Nonstop Database

HiRDB Version 9

UAP Development Guide

3020-6-456-10(E)

Relevant program products

List of program products:

For the Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T), Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T), or Linux 5 (AMD/Intel 64) operating system:

P-9W62-3592 HiRDB Server Version 9 09-01

P-F9W62-11925 HiRDB Non Recover Front End Server Version 9 09-00

P-F9W62-11926 HiRDB Advanced High Availability Version 9 09-00

This edition of the manual is released for the preceding program products, which have been developed under a quality management system that has been certified to comply with ISO9001 and TickIT. This manual may also apply to other program products; for details, see *Before Installing* or *Readme file* (for the UNIX version, see *Software Information* or *Before Installing*).

Trademarks

■ Restrictions

Information in this document is subject to change without notice and does not represent a commitment on the part of Hitachi. The software described in this manual is furnished according to a license agreement with Hitachi. The license agreement contains all of the terms and conditions governing your use of the software and documentation, including all warranty rights, limitations of liability, and disclaimers of warranty.

Material contained in this document may describe Hitachi products not available or features not available in your country.

Printed in Japan.

■ Issued

Dec. 2011: 3020-6-456-10(E)

■ Copyright

Preface

This manual describes the following items:

- Basic techniques needed to develop user application programs using the HiRDB Version 9 Nonstop Database program product with SQL as the database language.
- Environment setup for HiRDB Client

In this manual, a user application program is referred to as a *UAP*.

Intended readers

This manual is intended for users who will be constructing or operating *HiRDB Version 9* ("HiRDB") relational database systems.

It is assumed that readers of this manual have the following:

- For Windows systems, a basic knowledge of managing Windows
- For UNIX Systems, a basic knowledge of managing UNIX or Linux
- A basic knowledge of SQL
- A basic knowledge of programming in C language, COBOL, or Java

Organization of this manual

This manual consists of the following chapters and appendixes:

1. Overview

Chapter 1 explains the work flow for creating UAPs and the types of SQL statements to be used.

2. Database Operations

Chapter 2 explains the data expressions used in a HiRDB database and the basic database operations.

3. UAP Design

Chapter 3 explains issues to be taken into consideration in designing a UAP.

4. UAP Design for Improving Performance and Handling

Chapter 4 describes issues that UAP designers should consider to improve UAP performance and usability.

5. Notes about Creating UAPs that Access Object Relational Databases

Chapter 5 describes notes about creating UAPs that access object relational databases.

6. Client Environment Setup

Chapter 6 explains the procedure for installing a HiRDB client and describes the environment definition for creating and executing a UAP.

7. UAP Creation

This chapter explains the creation of embedded SQL UAPs written in C or COBOL.

8. Preparation for UAP Execution

Chapter 8 explains the flow from UAP preprocessing to execution and the methods used in those operations.

9. Java Stored Procedures and Java Stored Functions

Chapter 9 explains the development of stored procedures and stored functions with Java.

10. C Stored Procedures and C Stored Functions

Chapter 10 explains how to use C language to develop stored procedures and stored functions.

11. UAP Troubleshooting

Chapter 11 explains collection of historical information for UAP execution and error information; also explains the UAP error types and recovery methods.

12. Command Execution from UAPs

Chapter 12 explains the execution of commands from UAPs.

13. Connection from an XDS Client

Chapter 13 explains the development work flow for a UAP that connects to a HiRDB server through an XDS client.

14. HiRDB Access from ODBC Application Programs

Chapter 14 explains the ODBC driver installation procedure and ODBC functions.

15. HiRDB Access from OLE DB Application Programs

Chapter 15 explains HiRDB access from OLE DB application programs.

16. HiRDB Access from ADO.NET-compatible Application Programs

Chapter 16 describes how to access HiRDB from application programs that are compliant with ADO.NET.

17. Type2 JDBC Driver

Chapter 17 explains the Type2 JDBC driver installation and JDBC functions.

18. Type4 JDBC Driver

Chapter 18 explains the Type4 JDBC driver installation and JDBC functions.

19. SQLJ

Chapter 19 explains how to use SQLJ to develop a UAP.

A. SQL Communications Area

Appendix A explains the organization and contents of the SQL Communications Area, as well as expansion of the SQL Communications Areas.

B. SQL Descriptor Area

Appendix B explains the organization and contents of the SQL Descriptor Area, as well as expansion of the SQL Descriptor Area.

C. Column Name Descriptor Area

Appendix C explains the organization and contents of the Column Name Descriptor Area, as well as expansion of the Column Name Descriptor Area.

D. Type Name Descriptor Area

Appendix D explains the organization and contents of the Type Name Descriptor Area and expansion of the area.

E. Character Set Descriptor Areas

Appendix E describes the organization and content of the character set descriptor areas. This appendix also explains how to extend character set descriptor areas.

F. SQL Data Types and Data Descriptions

Appendix F explains the correspondence between the SQL data types and the C data descriptions, and the correspondence between the SQL data types and the COBOL data descriptions.

G. Data Dictionary Table Retrieval

Appendix G explains the contents of the data dictionary tables and how to reference them.

H. Functions provided by HiRDB

Appendix H explains the hash function for table partitioning, the space conversion function, the function for conversion to a DECIMAL signed normalized number, and the function that sets the character code classification.

I. Scalar Functions That Can Be Specified in the Escape Clause

Appendix I lists the scalar functions that can be specified in the escape clause.

J. Character Code Conversion Rules When Character Sets Are Used

Appendix J explains the rules for converting character codes between selected character sets.

K. HiRDB SQL Tuning Advisor Environment Setup

Appendix K explains how to set up an environment for HiRDB SQL Tuning Advisor.

L. Maximum and Minimum HiRDB Values

Appendix L explains the HiRDB maximum and minimum values.

M. List of Sample UAPs

Appendix M provides a list of references to where HiRDB sample UAPs can be found.

Related publications

This manual is related to the following manuals, which should be read as required.

HiRDB (for UNIX)

- *For UNIX Systems HiRDB Version 9 Description* (3000-6-451)#
- *For UNIX Systems HiRDB Version 9 Installation and Design Guide* (3000-6-452(E))
- *For UNIX Systems HiRDB Version 9 System Definition* (3000-6-453(E))
- *For UNIX Systems HiRDB Version 9 System Operation Guide* (3000-6-454(E))
- *For UNIX Systems HiRDB Version 9 Command Reference* (3000-6-455(E))
- *For UNIX Systems HiRDB Version 9 Staticizer Option Description and User's Guide* (3000-6-463)#
- *For UNIX Systems HiRDB Version 9 Disaster Recovery System Configuration and Operation Guide* (3000-6-464(E))
- *For UNIX Systems HiRDB Version 9 Memory Database Installation and Operation Guide* (3020-6-469)#
- *For UNIX Systems HiRDB First Step Guide* (3000-6-254)#

HiRDB (for both Windows and UNIX)

- *HiRDB Version 9 SQL Reference* (3020-6-457(E))

- *HiRDB Version 9 Messages* (3020-6-458(E))
- *HiRDB Version 9 XDM/RD E2 Connection Facility* (3020-6-465)#
- *HiRDB Version 9 Batch Job Accelerator* (3020-6-468)#
- *HiRDB Version 9 XML Extension* (3020-6-480)#
- *HiRDB Version 9 Text Search Plug-in* (3020-6-481)#
- *HiRDB Version 8 Security Guide* (3020-6-359)#
- *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide* (3020-6-360(E))
- *HiRDB Datareplicator Extension Version 8* (3020-6-361)#
- *HiRDB Dataextractor Version 8 Description, User's Guide and Operator's Guide* (3020-6-362(E))

In references to HiRDB Version 9 manuals, this manual omits the phrases *for UNIX systems* and *for Windows systems*. Refer to either the UNIX or Windows HiRDB manual, whichever is appropriate for your platform.

#: This manual has been published in Japanese only; it is not available in English.

For related products

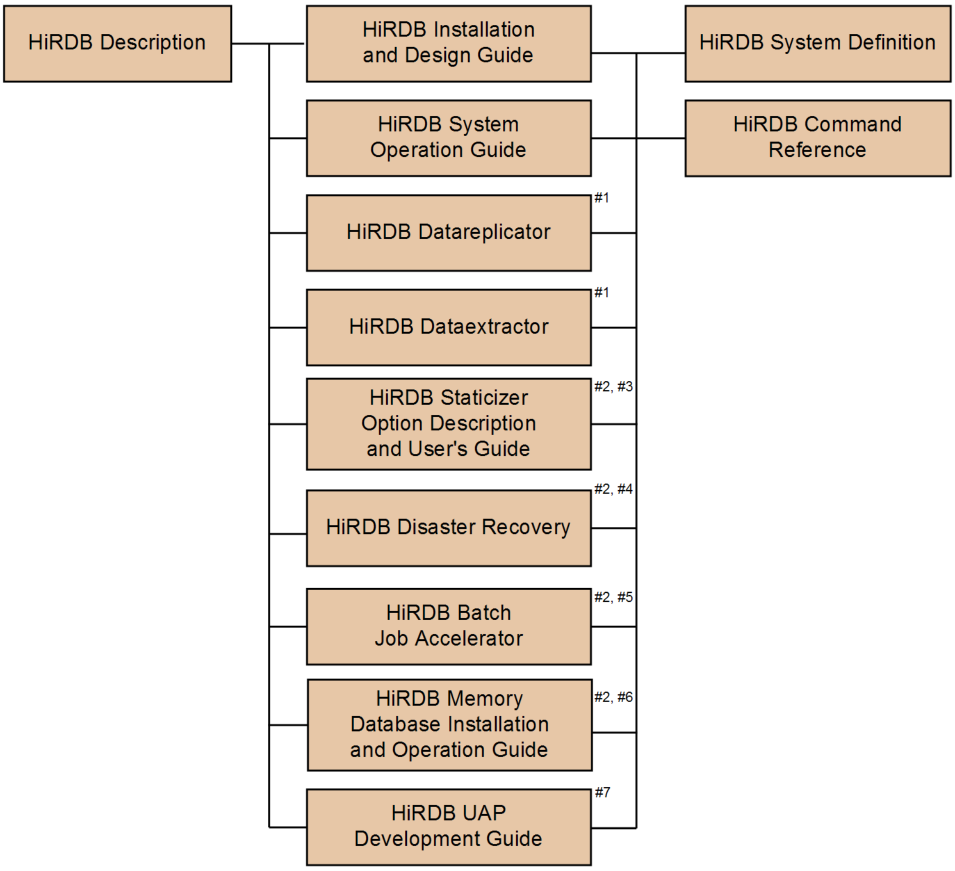
- *COBOL85 User's Guide* (3000-3-347)#
- *COBOL85 Operations Guide* (3020-3-747(E))
- *OpenTP1 Version 7 System Definition* (3000-3-D52(E))
- *OpenTP1 Version 7 Programming Reference C Language* (3000-3-D54(E))
- *OpenTP1 Version 7 Programming Reference COBOL Language* (3000-3-D55(E))
- *TP1/LINK USER'S GUIDE* (3000- 3-390(E))
- *TPBroker User's Guide* (3000-3-555(E))
- *Cosminexus Application Setup Guide* (3020-3-M08(E))

#: This manual has been published in Japanese only; it is not available in English.

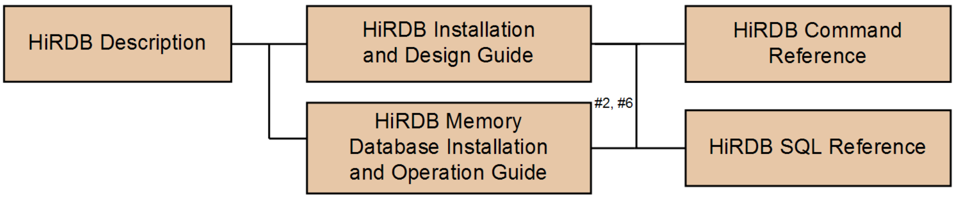
Organization of HiRDB manuals

The HiRDB manuals are organized as shown below. For the most efficient use of these manuals, it is suggested that they be read in the order they are shown, going from left to right.

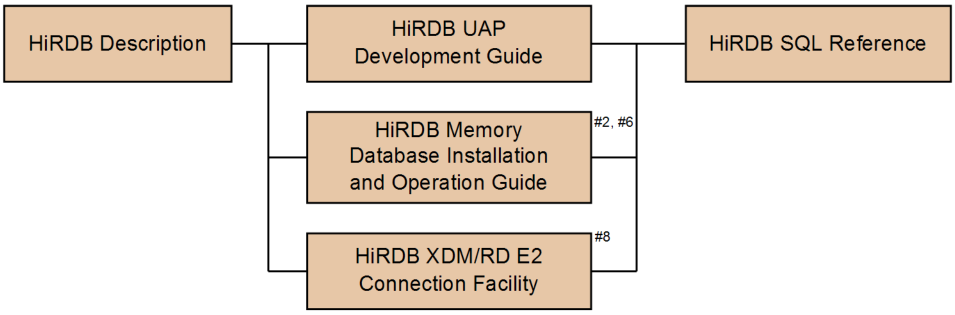
For system administrators:


HiRDB Description
HiRDB Installation and Design Guide
HiRDB System Operation Guide
HiRDB Datareplicator
#1
HiRDB Dataextractor
#1
HiRDB Staticizer Option Description and User's Guide
#2, #3
HiRDB Disaster Recovery
#2, #4
HiRDB Batch Job Accelerator
#2, #5
HiRDB Memory Database Installation and Operation Guide
#2, #6
HiRDB UAP Development Guide
#7
HiRDB System Definition
HiRDB Command Reference

For users who create tables:


HiRDB Description
HiRDB Installation and Design Guide
HiRDB Memory Database Installation and Operation Guide
#2, #6
HiRDB Command Reference
HiRDB SQL Reference

For users who create or execute UAPs:


HiRDB Description
HiRDB UAP Development Guide
HiRDB SQL Reference
HiRDB Memory Database Installation and Operation Guide
#2, #6
HiRDB XDM/RD E2 Connection Facility
#8

#1: Read if you intend to use the replication facility to link data.
#2: Published for UNIX only. There is no corresponding Windows manual.
#3: Read if you intend to use the inner replica facility.
#4: Read if you intend to configure a disaster recovery system.
#5: Read if you intend to use in-memory data processing to accelerate batch operations.
#6: Read if you intend to use the memory database facility.
#7: Read if you intend to link HiRDB to an OLTP system.
#8: Read if you intend to use the XDM/RD E2 connection facility to perform operations on XDM/RD E2 databases.

Conventions: Abbreviations for product names

This manual uses the following abbreviations for product names:

Full name or meaning	Abbreviation	
HiRDB Server Version 9	HiRDB/Single Server	HiRDB or HiRDB Server
	HiRDB/Parallel Server	
HiRDB/Developer's Kit Version 9	HiRDB/Developer's Kit	HiRDB Client
HiRDB/Developer's Kit Version 9 (64)		
HiRDB/Run Time Version 9	HiRDB/Run Time	
HiRDB/Run Time Version 9 (64)		
HiRDB Advanced High Availability Version 9	HiRDB Advanced High Availability	
HiRDB Accelerator Version 8	HiRDB Accelerator	
HiRDB Accelerator Version 9		
HiRDB Non Recover Front End Server Version 9	HiRDB Non Recover FES	

Full name or meaning	Abbreviation
HiRDB Staticizer Option Version 9	HiRDB Staticizer Option
HiRDB Disaster Recovery Light Edition Version 9	HiRDB Disaster Recovery Light Edition
HiRDB Text Search Plug-in Version 9	HiRDB Text Search Plug-in
HiRDB XML Extension Version 9	HiRDB XML Extension
HiRDB Datareplicator Version 8	HiRDB Datareplicator
HiRDB Dataextractor Version 8	HiRDB Dataextractor
HiRDB Adapter for XML - Standard Edition	HiRDB Adapter for XML
HiRDB Adapter for XML - Enterprise Edition	
HiRDB Control Manager	HiRDB CM
HiRDB Control Manager Agent	HiRDB CM Agent
Hitachi TrueCopy	TrueCopy
Hitachi TrueCopy basic	
TrueCopy	
TrueCopy remote replicator	
JP1/Automatic Job Management System 3	JP1/AJS3
JP1/Automatic Job Management System 2	
JP1/Automatic Job Management System 2 - Scenario Operation	JP1/AJS2-SO
JP1/Cm2/Extensible SNMP Agent	JP1/ESA
JP1/Cm2/Extensible SNMP Agent for Mib Runtime	
JP1/Cm2/Network Node Manager	JP1/NNM
JP1/Integrated Management - Manager	JP1/Integrated Management or JP1/IM
JP1/Integrated Management - View	
JP1/Magnetic Tape Access	EasyMT
EasyMT	
JP1/Magnetic Tape Library	MTguide
JP1/NETM/Audit - Manager	JP1/NETM/Audit

Full name or meaning	Abbreviation	
JP1/NETM/DM	JP1/NETM/DM	
JP1/NETM/DM Manager		
JP1/Performance Management	JP1/PFM	
JP1/Performance Management - Agent Option for HiRDB	JP1/PFM-Agent for HiRDB	
JP1/Performance Management - Agent Option for Platform	JP1/PFM-Agent for Platform	
JP1/Performance Management/SNMP System Observer	JP1/SSO	
JP1/VERITAS NetBackup BS v4.5	NetBackup	
JP1/VERITAS NetBackup v4.5		
JP1/VERITAS NetBackup BS V4.5 Agent for HiRDB License	JP1/VERITAS NetBackup Agent for HiRDB License	
JP1/VERITAS NetBackup V4.5 Agent for HiRDB License		
JP1/VERITAS NetBackup 5 Agent for HiRDB License		
OpenTP1/Server Base Enterprise Option	TP1/EE	
Virtual-storage Operating System 3/Forefront System Product	VOS3/FS	VOS3
Virtual-storage Operating System 3/Leading System Product	VOS3/LS	
Virtual-storage Operating System 3/Unific System Product	VOS3/US	
Extensible Data Manager/Base Extended Version 2 XDM Basic Program XDM/BASE E2	XDM/BASE E2	
XDM/Data Communication and Control Manager 3 XDM Data Communication Management System XDM/DCCM3	XDM/DCCM3	
XDM/Relational Database Relational Database System XDM/RD	XDM/RD	XDM/RD
XDM/Relational Database Extended Version 2 Relational Database System XDM/RD E2	XDM/RD E2	
VOS3 Database Connection Server	DB Connection Server	
Oracle WebLogic Server	WebLogic Server	
DB2 Universal Database for OS/390 Version 6	DB2	
$^{\text{DNCWARE}}$ ClusterPerfect (Linux Edition)	ClusterPerfect	
Java$^{\text{TM}}$	Java	

Full name or meaning	Abbreviation	
Microsoft(R) Office Excel	Microsoft Excel or Excel	
Microsoft(R) Visual C++(R)	Visual C++ or C++ language	
PowerHA for AIX, V5.5	PowerHA	
PowerHA SystemMirror V6.1		
HP-UX 11i V2 (IPF)	HP-UX or HP-UX (IPF)	
HP-UX 11i V3 (IPF)		
AIX 5L V5.2	AIX 5L	AIX
AIX 5L V5.3		
AIX V6.1	AIX V6.1	
Linux(R)	Linux	
Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T)	Linux AS 4	Linux
Red Hat Enterprise Linux AS 4 (x86)		
Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T)	Linux ES 4	
Red Hat Enterprise Linux ES 4 (x86)		
Red Hat Enterprise Linux 5.1 Advanced Platform (x86)	Linux 5.1	
Red Hat Enterprise Linux 5.1 (x86)		
Red Hat Enterprise Linux 5.1 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.1 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.2 Advanced Platform (AMD/Intel 64)	Linux 5.2	
Red Hat Enterprise Linux 5.2 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.3 Advanced Platform (AMD/Intel 64)	Linux 5.3	
Red Hat Enterprise Linux 5.3 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.4 Advanced Platform (AMD/Intel 64)	Linux 5.4	
Red Hat Enterprise Linux 5.4 (AMD/Intel 64)		
Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T)	Linux (EM64T)	
Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T)		
Red Hat Enterprise Linux 5.1 Advanced Platform (AMD/Intel 64)		

Full name or meaning	Abbreviation	
Red Hat Enterprise Linux 5.1 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.2 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.2 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.3 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.3 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.4 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.4 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.1 Advanced Platform (x86)	Linux 5 (x86)	Linux 5
Red Hat Enterprise Linux 5.1 (x86)		
Red Hat Enterprise Linux 5.1 Advanced Platform (AMD/Intel 64)	Linux 5 (AMD/ Intel 64)	
Red Hat Enterprise Linux 5.1 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.2 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.2 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.3 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.3 (AMD/Intel 64)		
Red Hat Enterprise Linux 5.4 Advanced Platform (AMD/Intel 64)		
Red Hat Enterprise Linux 5.4 (AMD/Intel 64)		
turbolinux 7 Server for AP8000	Linux for AP8000	
Microsoft(R) Windows NT(R) Workstation Operating System Version 4.0	Windows NT	
Microsoft(R) Windows NT(R) Server Network Operating System Version 4.0		
Microsoft(R) Windows(R) 2000 Professional Operating System	Windows 2000	
Microsoft(R) Windows(R) 2000 Server Operating System		
Microsoft(R) Windows(R) 2000 Datacenter Server Operating System		
Microsoft(R) Windows(R) 2000 Advanced Server Operating System		
Microsoft(R) Windows(R) 2000 Advanced Server Operating System	Windows 2000 Advanced Server	

Full name or meaning	Abbreviation	
Microsoft(R) Windows Server(R) 2003, Standard Edition	Windows Server 2003 Standard Edition	Windows Server 2003
Microsoft(R) Windows Server(R) 2003, Enterprise Edition	Windows Server 2003 Enterprise Edition	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition	Windows Server 2003 Standard x64 Edition	
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition	Windows Server 2003 Enterprise x64 Edition	
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition	Windows Server 2003 R2	
Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition		
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition	Windows Server 2003 R2 x64 Editions	
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition		
Microsoft(R) Windows Server(R) 2003, Enterprise Edition (64-bit version)	Windows Server 2003 (IPF)	
Microsoft(R) Windows Server(R) 2008 Standard	Windows Server 2008 Standard	Windows Server 2008
Microsoft(R) Windows Server(R) 2008 Enterprise	Windows Server 2008 Enterprise	
Microsoft(R) Windows Server(R) 2008 R2 Standard (x64)	Windows Server 2008 R2	
Microsoft(R) Windows Server(R) 2008 R2 Enterprise (x64)		
Microsoft(R) Windows Server(R) 2008 R2 Datacenter (x64)		
Microsoft(R) Windows Server(R) 2008 Standard (x64)	Windows Server 2008 (x64)	
Microsoft(R) Windows Server(R) 2008 Enterprise (x64)		

Full name or meaning	Abbreviation	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition	Windows Server 2003 x64 Editions	Windows (x64)
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition		
Microsoft(R) Windows(R) XP Professional x64 Edition	Windows XP x64 Edition	
Microsoft(R) Windows Server(R) 2003, Enterprise Edition (64-bit version)	Windows Server 2003 (IPF)	Windows (IPF)
Microsoft(R) Windows(R) XP Professional x64 Edition	Windows XP x64 Edition	Windows XP
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP Professional	
Microsoft(R) Windows(R) XP Home Edition Operating System	Windows XP Home Edition	
Microsoft(R) Windows Vista(R) Home Basic	Windows Vista Home Basic	Windows Vista
Microsoft(R) Windows Vista(R) Home Premium	Windows Vista Home Premium	
Microsoft(R) Windows Vista(R) Ultimate	Windows Vista Ultimate	
Microsoft(R) Windows Vista(R) Business	Windows Vista Business	
Microsoft(R) Windows Vista(R) Enterprise	Windows Vista Enterprise	
Microsoft(R) Windows Vista(R) Home Basic (x64)	Windows Vista (x64)	
Microsoft(R) Windows Vista(R) Home Premium (x64)		
Microsoft(R) Windows Vista(R) Ultimate (x64)		
Microsoft(R) Windows Vista(R) Business (x64)		
Microsoft(R) Windows Vista(R) Enterprise (x64)		
Microsoft(R) Windows(R) 7 Home Premium	Windows 7	
Microsoft(R) Windows(R) 7 Professional		

Full name or meaning	Abbreviation
Microsoft(R) Windows(R) 7 Enterprise	
Microsoft(R) Windows(R) 7 Ultimate	
Microsoft(R) Windows(R) 7 Home Premium (x64)	Windows 7 (x64)
Microsoft(R) Windows(R) 7 Professional (x64)	
Microsoft(R) Windows(R) 7 Enterprise (x64)	
Microsoft(R) Windows(R) 7 Ultimate (x64)	
Single server	SDS
System manager	MGR
Front-end server	FES
Dictionary server	DS
Back-end server	BES

- Windows Server 2003 and Windows Server 2008 may be referred to collectively as *Windows Server*. Windows 2000, Windows XP, Windows Server, Windows Vista, and Windows 7 may be referred to collectively as *Windows*.
- The hosts file means the `hosts` file stipulated by TCP/IP (including the `/etc/hosts` file). As a rule, a reference to the hosts file means the `%windir%\system32\drivers\etc\hosts` file.

This manual also uses the following acronyms:

Acronym	Full name or meaning
ACK	Acknowledgement
ADM	Adaptable Data Manager
ADO	ActiveX Data Objects
ADT	Abstract Data Type
AP	Application Program
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
BES	Back End Server
BLOB	Binary Large Object

Acronym	Full name or meaning
BMP	Basic Multilingual Plane
BOM	Byte Order Mark
CD-ROM	Compact Disc - Read Only Memory
CGI	Common Gateway Interface
CLOB	Character Large Object
CMT	Cassette Magnetic Tape
COBOL	Common Business Oriented Language
CORBA(R)	Common ORB Architecture
CPU	Central Processing Unit
CSV	Comma Separated Values
DAO	Data Access Object
DAT	Digital Audio Tape
DB	Database
DBM	Database Module
DBMS	Database Management System
DDL	Data Definition Language
DF for Windows NT	Distributing Facility for Windows NT
DF/UX	Distributing Facility/for UNIX
DIC	Dictionary Server
DLT	Digital Linear Tape
DML	Data Manipulate Language
DNS	Domain Name System
DOM	Document Object Model
DS	Dictionary Server
DTD	Document Type Definition
DTP	Distributed Transaction Processing
DWH	Data Warehouse

Acronym	Full name or meaning
EUC	Extended UNIX Code
EX	Exclusive
FAT	File Allocation Table
FD	Floppy Disk
FES	Front End Server
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GUI	Graphical User Interface
HBA	Host Bus Adapter
HD	Hard Disk
HTML	Hyper Text Markup Language
ID	Identification number
IP	Internet Protocol
IPF	Itanium(R) Processor Family
JAR	Java Archive File
Java VM	Java Virtual Machine
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFS	Journaled File System
JFS2	Enhanced Journaled File System
JIS	Japanese Industrial Standard code
JP1	Job Management Partner 1
JRE	Java Runtime Environment
JTA	Java Transaction API
JTS	Java Transaction Service
KEIS	Kanji processing Extended Information System
LAN	Local Area Network

Acronym	Full name or meaning
LDAP	Lightweight Directory Access Protocol
LIP	Loop Initialization Process
LOB	Large Object
LRU	Least Recently Used
LTO	Linear Tape-Open
LU	Logical Unit
LUN	Logical Unit Number
LVM	Logical Volume Manager
MGR	System Manager
MIB	Management Information Base
MRCF	Multiple RAID Coupling Feature
MSCS	Microsoft Cluster Server
MSFC	Microsoft Failover Cluster
NAFO	Network Adapter Fail Over
NAPT	Network Address Port Translation
NAT	Network Address Translation
NIC	Network Interface Card
NIS	Network Information Service
NTFS	New Technology File System
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLE	Object Linking and Embedding
OLTP	On-Line Transaction Processing
OOCOBOL	Object Oriented COBOL
ORB	Object Request Broker
OS	Operating System
OSI	Open Systems Interconnection

Acronym	Full name or meaning
OTS	Object Transaction Service
PC	Personal Computer
PDM II E2	Practical Data Manager II Extended Version 2
PIC	Plug-in Code
PNM	Public Network Management
POSIX	Portable Operating System Interface for UNIX
PP	Program Product
PR	Protected Retrieve
PU	Protected Update
RAID	Redundant Arrays of Inexpensive Disk
RD	Relational Database
RDB	Relational Database
RDB1	Relational Database Manager 1
RDB1 E2	Relational Database Manager 1 Extended Version 2
RDO	Remote Data Objects
RiSe	Real time SAN replication
RM	Resource Manager
RMM	Resource Manager Monitor
RPC	Remote Procedure Call
SAX	Simple API for XML
SDS	Single Database Server
SGML	Standard Generalized Markup Language
SJIS	Shift JIS
SNMP	Simple Network Management Protocol
SNTP	Simple Network Time Protocol
SQL	Structured Query Language
SQL/K	Structured Query Language / VOS K`

Acronym	Full name or meaning
SR	Shared Retrieve
SU	Shared Update
TCP/IP	Transmission Control Protocol / Internet Protocol
TM	Transaction Manager
TMS-4V/SP	Transaction Management System - 4V / System Product
UAP	User Application Program
UOC	User Own Coding
VOS K	Virtual-storage Operating System Kindness
VOS1	Virtual-storage Operating System 1
VOS3	Virtual-storage Operating System 3
WS	Workstation
WWW	World Wide Web
XDM/BASE E2	Extensible Data Manager / Base Extended Version 2
XDM/DF	Extensible Data Manager / Distributing Facility
XDM/DS	Extensible Data Manager / Data Spreader
XDM/RD E2	Extensible Data Manager / Relational Database Extended Version 2
XDM/SD E2	Extensible Data Manager / Structured Database Extended Version 2
XDM/XT	Extensible Data Manager / Data Extract
XDS	Extended Data Server
XFIT	Extended File Transmission program
XML	Extensible Markup Language

Path name representations

- The backslash (\) is used as the delimiter in path names. Readers who are using a UNIX version of HiRDB must replace the backslash with a forward slash (/). When the path names in the Windows and UNIX versions differ, both path names are given.
- The HiRDB directory path is represented as `%PDDIR%`. However, when the path names in the Windows and UNIX versions differ, the directory path in the UNIX

version is represented as `$PDDIR`, as shown in the following example:

Windows version: `%PDDIR%\CLIENT\UTL\`

UNIX version: `$PDDIR/client/lib/`

- `%windir%` refers to a Windows installation directory path.

Log representations

- Windows version

The application log that is displayed by Windows Event Viewer is referred to as the *event log*. The following procedure is used to view the event log.

To view the event log:

1. Choose **Start**, **Programs**, **Administrative Tools (Common)**, and then **Event Viewer**.

2. Choose **Log**, and then **Application**.

The application log is displayed. Messages with **HiRDBSingleServer** or **HiRDBParallelServer** displayed in the **Source** column were issued by HiRDB.

If you specified a setup identifier when you installed HiRDB, the specified setup identifier follows **HiRDBSingleServer** or **HiRDBParallelServer**.

- UNIX version

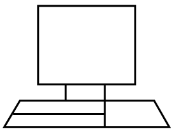
The OS log is referred to generically as *syslogfile*. syslogfile is the log output destination specified in `/etc/syslog.conf`. Typically, the following files are specified as syslogfile.

OS	File
HP-UX	`/var/adm/syslog/syslog.log`
Solaris	`/var/adm/messages` or `/var/log/syslog`
AIX	`/var/adm/ras/syslog`
Linux	`/var/log/messages`

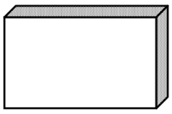
Conventions: Diagrams

This manual uses the following conventions in diagrams:

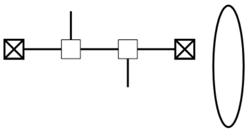
- Workstation or personal computer



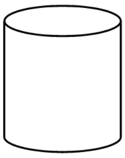
- Program



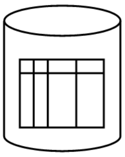
- Network (LAN)



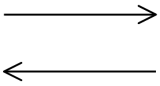
- File



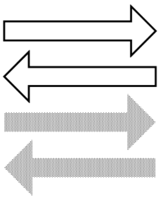
- Database



- Flow of control



- Flow of data



Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

Font	Convention
Bold	**Bold** type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italics	*Italics* are used to indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` *Italics* are also used for emphasis. For example: • Do *not* delete the configuration file.
`Code font`	A `code font` indicates text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`
`SD`	Bold code-font characters indicate the abbreviation for a command.
`perm`	Underlined characters indicate the default value.

The following table explains the symbols used in this table:

Symbol	Convention
|	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example: `A|B|C` means A, or B, or C.
{ }	In syntax explanations, curly brackets indicate that only one of the enclosed items is to be selected. For example: `A|B|C` means A, or B, or C.
[ ]	In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example: `[A]` means that you can specify `A` or nothing. `[B|C]` means that you can specify `B`, or `C`, or nothing.
. . .	In coding, an ellipsis (. . .) indicates that one or more lines of coding are not shown for purposes of brevity. In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: `A, B, B, ...` means that, after you specify `A, B,` you can specify B as many times as necessary.
()	Parentheses indicate the range of items to which the vertical bar (|) or ellipsis (. . .) is applicable.
~	A swung dash precedes the attributes of a user-specified value.
<< >>	Double angle brackets enclose the default value that the system assumes when the specification is omitted.
< >	A single pair of angle brackets encloses the syntax element notation for a user-specified value.
(())	A double pair of parentheses encloses the permitted range of values that can be specified.

Syntax element conventions

The following table explains the syntactical element symbols used in this manual:

Syntax element	Meaning
<alphabetics>	The alphabetic characters (`A` to `Z` and `a` to `z`) and the underscore (_)
<alphabetics and special characters>	The alphabetic characters (`A` to `Z` and `a` to `z`) and the special characters `#`, `@`, and \
<alphanumerics>	Alphabetic characters and the numeric characters (`0` to `9`)
<alphanumerics and special characters>	Alphabetic characters, special characters, and numeric characters

Syntax element	Meaning
<unsigned-integer>	Numeric values
<unsigned-decimal>#1	Numeric value (`0` to `9`), period (`.`), numeric value (`0` to `9`)
<identifier>#2	Alphanumeric character string beginning with an alphabetic character
<character-string>	String of any characters
<symbolic-name>	Alphanumeric character string beginning with an alphabetic character or a special character In the UNIX edition, symbolic names cannot include a backslash (`\`).
<path-name>#3	In the UNIX edition, path names can include forward slashes (`/`), alphanumeric characters, periods (`.`), hash marks (`#`), and at marks (`@`). In the Windows edition, path names can include backslashes (`\`), alphanumeric characters, periods (`.`), spaces, parentheses (`(` and `)`), hash marks (`#`), and at marks (`@`).

Use all single-byte characters. Alphabetic characters are case-sensitive. The path name depends on the OS in use.

#1

If all the numeric characters preceding the period are zeros (`0`), those zeros can be omitted. Similarly, if all the numeric characters following the period are zeros (`0`), those zeros can be omitted.

Example 1: `0.008` ➜ `.008`

Example 2: `15.000` ➜ `15`

#2

RDAREA names must begin with an alphabetic or special character, and can include alphanumeric and special characters, underscores (_), hyphens (`-`), and spaces. When an RDAREA name includes a space, the entire name must be enclosed in double quotation marks (`"`).

A host name is a character string that can include one or more alphabetic characters (`A` to `Z`, `a` to `z`), numeric characters, periods (`.`), hyphens (`-`), underscores (_), and at marks (`@`). A host name can begin with a numeric character.

#3

If you use a space or a parenthesis in a path name, you must enclose the entire path name in double quotation marks (`"`).

In the Windows edition, you can use a colon (`:`) in the drive name.

Notations used in computational expressions

The following notations are used in computational expressions

Symbol	Meaning
↑ ↑	Round up the result to the next integer. Example: The result of ↑34 ÷ 3↑ is 12.
↓ ↓	Discard digits following the decimal point. Example: The result of ↓34 ÷ 3↓ is 11.
MAX	Select the largest value as the result. Example: The result of Max(10, 2 x 4, 3 + 8) is 11.
MIN	Select the smallest value as the result. Example: The result of Min(10, 2 x 4, 3 + 8) is 8.

Notes on Windows path names

- In this manual, the Windows terms *directory* and *folder* are both referred to as *directory*.
- Include the drive name when you specify an absolute path name.

Example: `C:\win32app\hitachi\hirdb_s\spool\tmp`

- When you specify a path name in a command argument, in a control statement file, or in a HiRDB system definition file, and that path name includes a space or a parenthesis, you must enclose the entire path name in double quotation marks (").

Example: `pdinit -d "C:\Program Files(x86)\hitachi\hirdb_s\conf\mkinit"`

However, double quotation marks are not necessary when you use the `set` command in a batch file or at the command prompt to set an environment variable, or when you specify the installation directory. If you do use double quotation marks in such a case, the double quotation marks become part of the value assigned to the environment variable.

Example: `set PDCLTPATH=C:\Program Files\hitachi\hirdb_s\spool`

- HiRDB cannot use files on a networked drive, so you must install HiRDB and configure the HiRDB environment on a local drive. Files used by utilities, such as utility input and output files, must also be on the local drive.
- Do not use a short path name in place of the full path name (for example, do not use `C:\PROGRA~1`).

Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is 1,024^2 bytes.
- 1 GB (gigabyte) is 1,024^3 bytes.
- 1 TB (terabyte) is 1,024^4 bytes.

Conventions: Version numbers

The version numbers of Hitachi program products are usually written as two sets of two digits each, separated by a hyphen. For example:

- Version 1.00 (or 1.0) is written as 01-00.
- Version 2.05 is written as 02-05.
- Version 2.50 (or 2.5) is written as 02-50.
- Version 12.25 is written as 12-25.

The version number might be shown on the spine of a manual as *Ver. 2.00,* but the same version number would be written in the program as *02-00*.

HiRDB database language acknowledgements

The interpretations and specifications developed by Hitachi, Ltd. for the HiRDB database language specifications described in this manual are based on the standards listed below. In addition to citing the standards relevant to HiRDB database language specifications, we would like to take this opportunity to express our appreciation to the original developers of these standards.

- JIS X 3005 Information technology -- Database languages -- SQL
- ISO/IEC 9075 Information technology -- Database languages -- SQL

Legend:

JIS: Japanese Industrial Standards

ISO: International Organization for Standardization

IEC: International Electrotechnical Commission

Important notes on this manual

The following facilities are explained, but they are not supported:

- Distributed database facility
- Server mode system switchover facility

- User server hot standby
- Rapid system switchover facility
- Standby-less system switchover (1:1) facility
- Standby-less system switchover (effects distributed) facility
- HiRDB External Data Access facility
- Inner replica facility
- Updatable online reorganization
- Sun Java System Directory Server linkage facility
- Simple setup tool
- Extended syslog facility
- Rapid batch facility
- Memory database facility
- Linkage with JP1/NETM/Audit

The following products and option program products are explained, but they are not supported:

- HiRDB CM
- HiRDB Disaster Recovery Light Edition
- uCosminexus Grid Processing Server
- HiRDB Text Search Plug-in
- HiRDB XML Extension
- TP1/Server Base
- JP1/PFM-Agent Option for HiRDB
- JP1/VERITAS NetBackup Agent for HiRDB License
- HiRDB Dataextractor
- HiRDB Datareplicator
- XDM/RD
- HiRDB SQL Tuning Advisor
- COBOL2002

Notes on printed manuals

Please note that even though the printed manuals are separated into Part I and Part II, the chapters and page numbers sequentially continue from Part I to Part II. Also, please note that the index is only included in Part II.

Contents

Preface i

Intended readers ... i
Organization of this manual ... i
Related publications ... iv
Organization of HiRDB manuals ... v
Conventions: Abbreviations for product names ... vii
Path name representations ... xix
Log representations ... xx
Conventions: Diagrams ... xx
Conventions: Fonts and symbols ... xxi
Notes on Windows path names ... xxiv
Conventions: KB, MB, GB, and TB ... xxv
Conventions: Version numbers ... xxv
HiRDB database language acknowledgements ... xxv
Important notes on this manual ... xxv
Notes on printed manuals ... xxvi

1. Overview 1

1.1 UAP development flow ... 2
1.2 UAP characteristics ... 3
1.2.1 UAP format ... 3
1.2.2 List of SQL statements usable in HiRDB ... 4
1.3 Interface with HiRDB ... 17
1.4 UAP operation environment ... 18

2. Database Operations 27

2.1 Database data expressions ... 28
2.1.1 Relational database tables ... 28
2.1.2 Object relational database tables ... 30
2.2 Cursor usage ... 32
2.3 Data retrieval ... 35
2.3.1 Retrieval from a single table ... 35
2.3.2 Retrieval from multiple tables ... 39
2.3.3 Retrieval of a table with FIX attribute ... 41
2.4 Data updating ... 43
2.4.1 Updating using a cursor ... 43
2.4.2 Updating with a condition specified ... 44
2.4.3 Updating a table with the FIX attribute ... 45

2.4.4 Updating a table with repetition columns 46
2.5 Data deletion 49
2.5.1 Deletion using a cursor 49
2.5.2 Deletion with a condition specified 50
2.5.3 Deleting all rows in a table 51
2.6 Data insertion 53
2.6.1 Inserting rows on a column basis 53
2.6.2 Inserting rows on a row basis (to a table with the FIX attribute) 54
2.6.3 Inserting rows into a table with repetition columns 55
2.7 Specific data search 57
2.7.1 Searching for data within a specified range of values 57
2.7.2 Searching for a specific character pattern 60
2.7.3 Searching for non-NULL data 61
2.7.4 Searching for data that satisfies multiple conditions 62
2.7.5 Searching for data using a Boolean predicate 63
2.7.6 Searching for data using a structured repetition predicate 63
2.7.7 Searching for data using a subquery 64
2.8 Data operations 69
2.8.1 Arithmetic operations on numeric data 69
2.8.2 Date and time data operations 70
2.9 Data processing 72
2.9.1 Data grouping 72
2.9.2 Data sorting 73
2.9.3 Duplicated data elimination 74
2.10 Outer joining of tables 76
2.11 Defining and manipulating a view table 81
2.12 Manipulating data in a table with abstract data types 89
2.12.1 SGMLTEXT type 89
2.12.2 XML type 99
2.12.3 User-defined abstract data types 105
3. UAP Design 111
3.1 Basic SQL configuration in a UAP 112
3.2 Overview of UAPs 119
3.2.1 UAP descriptive languages 119
3.2.2 Interface areas 120
3.2.3 Integrity constraints 121
3.2.4 Retrieval methods using SQL statements 121
3.2.5 Static and dynamic SQLs 123
3.3 Transaction control 131
3.3.1 Connection to and disconnection from a HiRDB system 131
3.3.2 Transaction startup and termination 131
3.3.3 Synchronization point setting and rollback 131
3.3.4 UAP transaction management in an OLTP environment 132

3.3.5 Moving a transaction 134
3.4 Locking 136
3.4.1 Units of locking 136
3.4.2 Lock modes 137
3.4.3 Lock period 163
3.4.4 Deadlocks and corrective measures 164
3.4.5 Unlocked conditional search 172
3.4.6 Non-locking of index key values 175
3.4.7 Locking uncommitted data to be deleted 177
3.4.8 Locking by UAPs 184
3.4.9 Lock sequence based on SQL statement and index types 186
3.4.10 Lock sequence for rows 200
3.4.11 Creating locked resources for index key values 210
3.5 Use of a cursor 212
3.5.1 Notes on table operations when a cursor is used 212
3.5.2 FOR UPDATE and FOR READ ONLY clauses 215
3.5.3 Cursor declarations and locks 216
3.5.4 Holdable cursor 219
3.5.5 Examples of cursor use 223
3.6 SQL error identification and corrective measures 227
3.6.1 Error identification 227
3.6.2 Automatic error identification 230

4. UAP Design for Improving Performance and Handling 233

4.1 Using indexes 234
4.1.1 Indexes and processing time 234
4.1.2 Index priority 234
4.1.3 Changing indexes during retrieval 235
4.1.4 Notes about index searches 235
4.2 Manipulation of tables 244
4.2.1 Tables with the FIX attribute 244
4.2.2 Tables used in numbering 244
4.2.3 Tables using character sets 254
4.3 Stored procedures and stored functions 259
4.3.1 Defining a stored procedure 259
4.3.2 Defining a stored function 269
4.3.3 Defining and deleting stored functions 280
4.4 Triggers 283
4.5 SQL optimization 285
4.5.1 SQL optimizing modes 286
4.5.2 Optimization method types 300
4.5.3 Specifying SQL optimization 301
4.5.4 Allocating floatable servers (HiRDB/Parallel Server only) 302
4.5.5 Grouping processing methods (HiRDB/Parallel Server only) 309

4.5.6 Join methods 314
4.5.7 Search Methods 328
4.5.8 Execution of subqueries with no external references 335
4.5.9 Execution of subqueries with external references 341
4.5.10 Preparing for application of hash join and subquery hash execution 347
4.5.11 Deriving high-speed search conditions 355
4.6 Data guarantee levels 368
4.6.1 Specifying the data guarantee level 368
4.6.2 Data guarantee level types 369
4.6.3 Example of search results when a data guarantee level is specified 370
4.7 Block transfer facility 373
4.8 Facilities using arrays 377
4.8.1 FETCH facility using arrays 377
4.8.2 INSERT facility using arrays 386
4.8.3 UPDATE facility using arrays 399
4.8.4 DELETE facility using arrays 402
4.9 Rapid grouping facility 406
4.9.1 Overview 406
4.9.2 Application criteria 406
4.9.3 Specification method 407
4.9.4 Tuning method 407
4.10 Multi-connection facility 409
4.11 Narrowed search 424
4.11.1 What is a narrowed search? 424
4.11.2 Preparations for executing a narrowed search 424
4.11.3 Search using lists 425
4.11.4 Action if a rollback occurs for a transaction that uses a list 427
4.11.5 Automatic list deletion at HiRDB startup and termination 428
4.11.6 Notes about using lists 428
4.12 File output facility for BLOB data 431
4.12.1 What is the file output facility for BLOB data? 431
4.12.2 Application criteria 432
4.12.3 Specification method 433
4.12.4 Notes about using the file output facility for BLOB data 433
4.12.5 Examples of using the file output facility for BLOB data 433
4.13 Partial update and retrieval of BLOB and BINARY data 436
4.13.1 About partial update and retrieval of BLOB and BINARY data 436
4.13.2 Examples of using the addition update and partial extraction facility for BLOB data 436
4.13.3 Notes about performing partial updating and retrieval of BLOB and BINARY data 439
4.14 Retrieve first n records facility 441
4.14.1 Overview 441
4.14.2 Notes 441

4.14.3 Checking the access path....442
4.15 Automatic reconnect facility....443
4.15.1 Application criteria....443
4.15.2 Reconnect timings....443
4.15.3 CONNECT processing during automatic reconnect....446
4.15.4 Notes about using the automatic reconnect facility....446
4.16 Locator facility....448
4.16.1 What is the locator facility?....448
4.16.2 Application standard....450
4.16.3 Usage method....450
4.16.4 Usage example....450
4.17 Facility for returning the total number of hits....453
4.17.1 Overview....453
4.17.2 Usage examples....453
4.17.3 Note....454
4.18 Retrieval, updating, or deletion with an RDAREA name specified....455
4.18.1 Overview of function....455
4.18.2 Example....455
4.18.3 Notes....456
4.19 Automatic numbering facility....457
4.19.1 About sequence generators....457
4.19.2 Defining a sequence generator....458
4.19.3 Deleting a sequence generator....461
4.19.4 Acquiring the sequence numbers generated by a sequence generator....461
4.19.5 Examples....461
4.19.6 Notes....463

5. Notes about Creating UAPs that Access Object Relational Databases 471

5.1 Using abstract data types and user-defined functions....472
5.2 Restrictions on functions provided by plug-ins....474

6. Client Environment Setup 483

6.1 Types of HiRDB clients....484
6.2 Environment setup procedure for HiRDB clients....485
6.3 HiRDB client installation....486
6.3.1 Installing a HiRDB client on a UNIX client....486
6.3.2 Installing a HiRDB client on a Windows client....486
6.4 Organization of directories and files for a HiRDB client....490
6.4.1 Directories and files for UNIX clients....490
6.4.2 Directories and files for Windows clients....512
6.5 Setting the hosts file....532
6.6 Client environment definitions (setting environment variables)....533
6.6.1 Environment setup format....533
6.6.2 Specifications for using a UAP under OLTP as the client....545

6.6.3 Client environment definitions 562
6.6.4 Environment definition information 574
6.6.5 Environment variables and connection types for HiRDB servers 697
6.7 Registering an environment variable group 700
6.7.1 Registering an environment variable group in a UNIX environment 700
6.7.2 Registering an environment variable group in a Windows environment (registry registration) 701
6.7.3 Registering an environment variable group in a Windows environment (file registration) 708

7. UAP Creation 709

7.1 Overview 710
7.1.1 UAP basic configuration 710
7.1.2 UAP configuration elements 710
7.2 Writing a UAP in C 712
7.2.1 Coding rules 712
7.2.2 Program example 719
7.3 Writing a UAP in COBOL 750
7.3.1 Coding rules 750
7.3.2 Program example 755
7.4 Writing a UAP in C++ 786
7.4.1 Coding rules 786
7.5 Writing a UAP in OOCOBOL 787
7.5.1 Coding rules 787
7.6 Creating a UAP in 64-bit mode 788

8. Preparation for UAP Execution 793

8.1 UAP execution procedure 794
8.1.1 Executing a UAP written in C 794
8.1.2 Executing a UAP written in COBOL 795
8.2 Preprocessing 797
8.2.1 Overview 797
8.2.2 Preprocessing in UNIX 798
8.2.3 Preprocessing in Windows 816
8.2.4 Validating preprocessor declaration statements 831
8.2.5 Dispensing with the embedded SQL declare section 833
8.2.6 Specifying pointers as environment variables 834
8.2.7 Referencing structures 836
8.2.8 Use of pointers, structures, and structure qualifiers when the -E2 or -E3 option of the preprocessor is specified 839
8.3 Compiling and linking 842
8.3.1 Libraries for compiling and linking 842
8.3.2 Compiling and linking in UNIX 849
8.3.3 Compiling and linking in Windows 856

8.3.4 Compiling and linking when the multi-connection facility is used 859
8.4 Notes 866
8.4.1 Notes on UAP execution 866
8.4.2 Executing UAPs that use an X/Open-based API (TX_function) 871
8.4.3 UAP execution using the Unicode functionality of COBOL2002 880
8.4.4 Converting UAPs created with XDM/RD or UNIFY2000 882
8.4.5 Actions required when HiRDB is upgraded 882

9. Java Stored Procedures and Java Stored Functions 883

9.1 Overview 884
9.2 Procedure from external Java stored routine creation to execution 887
9.2.1 Creating an external Java stored routine 887
9.2.2 Registering the JAR file 890
9.2.3 Defining the external Java stored routine 891
9.2.4 Executing the external Java stored routine 892
9.3 Sample external Java stored routine programs 894
9.3.1 Sample program 894
9.3.2 Sample external Java stored routines provided with HiRDB 897
9.4 Notes about Java program creation 919
9.4.1 Using a Type 2 or 4 JDBC driver 919
9.4.2 Unsupported methods 919
9.4.3 Package, class, and method definitions 920
9.4.4 Parameter input/output mode mapping (Java stored procedures only) 921
9.4.5 Results-set return facility (Java stored procedures only) 922
9.4.6 Connection in a Java stored procedure 928
9.4.7 Releasing the result sets 928
9.5 Notes about testing and debugging 929
9.5.1 Java program for a Java stored procedure 929
9.5.2 Java program for a Java stored function 930
9.6 Notes about JAR file creation 932
9.6.1 Integrating Class files 933
9.6.2 Integrating Java files 933

10. C Stored Procedures and C Stored Functions 935

10.1 Overview 936
10.2 Procedure from external C stored routine creation to execution 937
10.2.1 Creating an external C stored routine 939
10.2.2 Registering the C library file 944
10.2.3 Defining the external C stored routine 945
10.2.4 Executing the external C stored routine 947
10.3 Sample external C stored routine programs 948
10.4 Limitations to C program creation 950

11. UAP Troubleshooting 953

11.1 Gathering error information ... 954
11.1.1 SQL tracing ... 954
11.1.2 Client error log facility ... 971
11.1.3 Facility for output of extended SQL error information ... 974
11.1.4 UAP statistical report facility ... 988
11.1.5 Command trace facility ... 1018
11.1.6 SQL trace dynamic acquisition facility ... 1020
11.1.7 Reconnect trace facility ... 1022
11.2 UAP error recovery ... 1025

12. Command Execution from UAPs 1027

12.1 Overview ... 1028
12.2 Preparations for executing commands from COMMAND EXECUTE ... 1030
12.3 Command executability ... 1037

13. Connection from an XDS Client 1045

13.1 Overview of format of connection from an XDS client ... 1046

14. HiRDB Access from ODBC Application Programs 1047

14.1 ODBC application programs ... 1048
14.2 Installing the ODBC2.0 driver ... 1049
14.3 Installing the ODBC 3.5 driver and setting the environment variables ... 1053
14.3.1 Installation ... 1053
14.3.2 Setting the environment variables (in the Windows edition) ... 1058
14.3.3 Determining the version number of the ODBC 3.5 driver ... 1059
14.4 ODBC functions provided by HiRDB ... 1060
14.5 ODBC function data types and HiRDB data types ... 1065
14.6 Specifiability of attributes in the ODBC functions ... 1070
14.7 Asynchronous execution of ODBC functions ... 1094
14.8 Setting cursor libraries ... 1098
14.9 File DSNs ... 1099
14.10 Executing a UAP in Unicode ... 1100
14.11 Tuning and troubleshooting ... 1103
14.12 Facilities that cannot be used when HiRDB is accessed with ODBC ... 1104
14.13 Notes about using the Linux edition of the HiRDB ODBC 3.5 driver ... 1105
14.14 Automatic SQL statement generation by using .NET Framework Data Provider for ODBC ... 1107

15. HiRDB Access from OLE DB Application Programs 1109

15.1 Overview ... 1110
15.2 Connection interface ... 1111
15.2.1 Registry information ... 1111

15.2.2 Connection properties 1112
15.3 Schema information........ 1114
15.4 Data type correspondences 1116
15.5 Error handling procedures 1117
15.5.1 Troubleshooting facility 1117
15.6 Notes........ 1118

16. HiRDB Access from ADO.NET-compatible Application Programs 1119

16.1 Overview 1120
16.1.1 HiRDB.NET Data Provider........ 1120
16.1.2 Prerequisite programs for HiRDB.NET Data Provider........ 1120
16.2 Installing HiRDB.NET Data Provider........ 1123
16.2.1 Installation procedure 1123
16.2.2 Files that are installed........ 1123
16.2.3 Checking the version information 1123
16.3 List of classes provided by HiRDB.NET Data Provider 1124
16.4 List of members provided by HiRDB.NET Data Provider 1126
16.4.1 List of HiRDBCommand members........ 1126
16.4.2 List of HiRDBCommandBuilder members........ 1127
16.4.3 List of HiRDBConnection members 1129
16.4.4 List of HiRDBDataAdapter members 1131
16.4.5 List of HiRDBDataReader members........ 1132
16.4.6 List of HiRDBException members 1135
16.4.7 List of HiRDBParameter members 1135
16.4.8 List of HiRDBParameterCollection members........ 1137
16.4.9 List of HiRDBProviderFactory members 1139
16.4.10 List of HiRDBRowUpdatedEventArgs members 1140
16.4.11 List of HiRDBRowUpdatingEventArgs members 1140
16.4.12 List of HiRDBTransaction members........ 1141
16.5 Interfaces of HiRDB.NET Data Provider........ 1142
16.5.1 HiRDBCommand........ 1142
16.5.2 HiRDBCommandBuilder 1146
16.5.3 HiRDBConnection 1152
16.5.4 HiRDBDataAdapter 1157
16.5.5 HiRDBDataReader........ 1159
16.5.6 HiRDBException 1169
16.5.7 HiRDBParameter 1169
16.5.8 HiRDBParameterCollection........ 1174
16.5.9 HiRDBProviderFactory........ 1181
16.5.10 HiRDBRowUpdatedEventArgs........ 1183
16.5.11 HiRDBRowUpdatingEventArgs 1183
16.5.12 HiRDBTransaction 1184
16.6 Notes about HiRDB.NET Data Provider........ 1186
16.6.1 Placing in global assembly cache........ 1186

16.6.2 Notes about individual methods and properties 1187
16.7 Data types of HiRDB.NET Data Provider..1190
16.7.1 DbType and HiRDBType properties 1190
16.7.2 Data types and accessories used by a UAP 1192
16.7.3 Type conversion by HiRDB.NET Data Provider 1193
16.8 Connection pooling function 1200
16.9 Provider-independent codes using DbProviderFactory 1201
16.10 Troubleshooting function of HiRDB.NET Data Provider 1205
16.11 Example of a UAP using HiRDB.NET Data Provider 1210
16.11.1 Connecting to the database 1210
16.11.2 Executing the SQL statement 1212
16.11.3 Executing a transaction... 1214
16.11.4 Executing a search statement.. 1217
16.11.5 Executing the INSERT facility using arrays.. 1218
16.11.6 Executing a repetition column 1219
16.11.7 Checking for an error in SQL statements and acquiring error information 1221

17. Type2 JDBC Driver 1223

17.1 Installation and environment setup 1224
17.1.1 Installing 1224
17.1.2 Environment setup 1224
17.1.3 Abbreviation of methods 1225
17.2 JDBC1.0 facility 1226
17.2.1 Driver class 1226
17.2.2 Connection class 1235
17.2.3 Statement class.. 1236
17.2.4 PreparedStatement class 1237
17.2.5 CallableStatement class 1237
17.2.6 ResultSet class 1238
17.2.7 ResultSetMetaData class 1239
17.2.8 DatabaseMetaData class 1242
17.2.9 SQLWarning class... 1242
17.3 JDBC2.0 basic facility 1244
17.3.1 Result set enhancements.. 1244
17.3.2 Batch updating... 1246
17.3.3 Added data types 1250
17.4 JDBC2.0 Optional Package 1259
17.4.1 Database connection using DataSource and JNDI 1259
17.4.2 Connection pooling.. 1262
17.4.3 Distributed transactions 1264
17.5 JAR file access facility 1267
17.5.1 Class name 1267
17.5.2 Method name 1267

17.6 Array class 1270
17.7 Specifying a value when using a repetition column as the ? parameter 1272
17.8 Functions provided by the HiRDB JDBC driver 1275
17.8.1 Provided class 1275
17.8.2 setBlockUpdate 1275
17.8.3 getBlockUpdate 1276
17.9 Notes on using the BLOB type 1278
17.10 Setting system properties 1280
17.10.1 Setting the array facility 1280
17.10.2 Setting the maximum number of SQL search items or ? parameters 1281
17.11 Connection information setup/acquisition interface 1284
17.11.1 setDescription 1286
17.11.2 getDescription 1288
17.11.3 setDBHostName 1288
17.11.4 getDBHostName 1289
17.11.5 setEncodeLang 1289
17.11.6 getEncodeLang 1291
17.11.7 setUser 1291
17.11.8 getUser 1292
17.11.9 setPassword 1293
17.11.10 getPassword 1293
17.11.11 setXAOpenString 1294
17.11.12 getXAOpenString 1295
17.11.13 setXACloseString 1296
17.11.14 getXACloseString 1296
17.11.15 setRMID 1297
17.11.16 getRMID 1297
17.11.17 setXAThreadMode 1298
17.11.18 getXAThreadMode 1298
17.11.19 setCommit_Behavior 1299
17.11.20 getCommit_Behavior 1301
17.11.21 setBlockUpdate 1301
17.11.22 getBlockUpdate 1302
17.11.23 setLONGVARBINARY_Access 1303
17.11.24 getLONGVARBINARY_Access 1304
17.11.25 setSQLInNum 1304
17.11.26 getSQLInNum 1305
17.11.27 setSQLOutNum 1306
17.11.28 getSQLOutNum 1307
17.11.29 setSQLWarningLevel 1307
17.11.30 getSQLWarningLevel 1308
17.11.31 setClear_Env 1309
17.11.32 getClear_Env 1309
17.12 Data types and character codes 1311

17.12.1 Data types 1311
17.12.2 Character code conversion facility 1312
17.13 Classes and methods with limitations 1314
17.13.1 Driver class 1314
17.13.2 Connection class 1314
17.13.3 Statement class 1315
17.13.4 PreparedStatement class 1316
17.13.5 CallableStatement class 1316
17.13.6 ResultSet class 1317
17.13.7 ResultSetMetaData class 1318
17.13.8 DatabaseMetaData class 1319
17.13.9 Blob class 1326
17.13.10 Array class 1326

18. Type4 JDBC Driver 1327

18.1 Installation and environment setup 1328
18.1.1 Installation 1328
18.1.2 Environment setup 1328
18.1.3 Abbreviation of methods 1329
18.2 Database connection using the DriverManager class 1330
18.2.1 Registering the Driver class 1330
18.2.2 Connecting to HiRDB with the getConnection method 1331
18.3 Database connection using a DataSource object and JNDI 1359
18.4 JDBC1.2 core API 1363
18.4.1 Driver interface 1363
18.4.2 Connection interface 1370
18.4.3 Statement interface 1403
18.4.4 PreparedStatement interface 1431
18.4.5 CallableStatement interface 1468
18.4.6 ResultSet interface 1561
18.4.7 DatabaseMetaData interface 1666
18.4.8 ResultSetMetaData interface 1811
18.4.9 Blob interface 1829
18.4.10 Array interface 1836
18.4.11 SQLException interface 1842
18.4.12 SQLWarning interface 1843
18.4.13 Unsupported interfaces 1844
18.5 JDBC2.1 Core API 1846
18.5.1 Expansion of the result set 1846
18.5.2 Batch update 1846
18.5.3 Added data types 1853
18.5.4 Unsupported interfaces 1853
18.6 JDBC2.0 Optional Package 1855
18.6.1 DataSource interface 1855

18.6.2 ConnectionPoolDataSource interface 1860
18.6.3 PooledConnection interface 1864
18.6.4 XAConnection interface...1868
18.6.5 XADataSource interface 1868
18.6.6 XAResource interface 1869
18.6.7 XAException interface...1870
18.6.8 Unsupported interfaces...1870
18.7 Connection information setup and acquisition interface...................................1871
18.7.1 setDescription...1874
18.7.2 getDescription 1876
18.7.3 setDBHostName...1877
18.7.4 getDBHostName 1878
18.7.5 setJDBC_IF_TRC 1878
18.7.6 getJDBC_IF_TRC...1879
18.7.7 setTRC_NO..1880
18.7.8 getTRC_NO 1880
18.7.9 setUapName 1881
18.7.10 getUapName...1882
18.7.11 setUser...1882
18.7.12 getUser 1883
18.7.13 setPassword 1884
18.7.14 getPassword...1885
18.7.15 setXAOpenString 1885
18.7.16 getXAOpenString..1887
18.7.17 setXACloseString..1887
18.7.18 getXACloseString 1888
18.7.19 setLONGVARBINARY_Access 1888
18.7.20 getLONGVARBINARY_Access...1889
18.7.21 setSQLInNum...1890
18.7.22 getSQLInNum 1891
18.7.23 setSQLOutNum...1891
18.7.24 getSQLOutNum 1892
18.7.25 setSQLWarningLevel 1893
18.7.26 getSQLWarningLevel..1894
18.7.27 setXALocalCommitMode 1894
18.7.28 getXALocalCommitMode..1896
18.7.29 setSQLWarningIgnore 1896
18.7.30 getSQLWarningIgnore...1897
18.7.31 setHiRDBCursorMode 1898
18.7.32 getHiRDBCursorMode...1899
18.7.33 setNotErrorOccurred 1899
18.7.34 getNotErrorOccurred...1900
18.7.35 setEnvironmentVariables...1901
18.7.36 getEnvironmentVariables 1902

18.7.37 setEncodeLang 1903
18.7.38 getEncodeLang 1904
18.7.39 setMaxBinarySize 1905
18.7.40 getMaxBinarySize 1906
18.7.41 setStatementCommitBehavior 1907
18.7.42 getStatementCommitBehavior 1908
18.7.43 setLONGVARBINARY_AccessSize 1909
18.7.44 getLONGVARBINARY_AccessSize 1910
18.7.45 setLONGVARBINARY_TruncError 1910
18.7.46 getLONGVARBINARY_TruncError 1911
18.7.47 setStatementCloseBehavior 1912
18.7.48 getStatementCloseBehavior 1913
18.7.49 setHiRDBINI 1914
18.7.50 getHiRDBINI 1914
18.7.51 setBatchExceptionBehavior 1915
18.7.52 getBatchExceptionBehavior 1916
18.8 Data types 1917
18.8.1 Mapping SQL data types 1917
18.8.2 Mapping during retrieval data acquisition 1918
18.8.3 Mapping when a ? parameter is set 1921
18.8.4 Data conversion of TIME, DATE, and TIMESTAMP columns 1926
18.8.5 Overflow handling 1929
18.9 Character conversion facility 1939
18.10 Supported client environment definitions 1940
18.11 Connection information priorities 1951
18.12 Migration from a Type2 JDBC driver 1959
18.13 Migration from DABroker for Java 1962
18.13.1 System properties related to the DABroker for Java-compatible facility 1962
18.13.2 Items that are not compatible with a Type4 JDBC driver 1964
18.14 JDBC interface method trace 1968
18.14.1 Setup for trace acquisition 1968
18.14.2 Acquisition rules 1968
18.14.3 Output example 1969
18.15 Exception trace log 1971
18.15.1 Methods to be acquired and setup for log acquisition 1971
18.15.2 Output formats 1979
18.15.3 Output example and analysis method 1986
18.15.4 Required memory size and file size 1991
18.15.5 Notes 1992
18.16 Example UAP that uses a JDBC driver 1995
18.17 Estimating the memory requirements for using a JDBC driver 1998
18.17.1 Estimating the Connection object size 1998
18.17.2 Estimating the Statement object size 2000

18.17.3 Estimating the PreparedStatement object size 2001
18.17.4 Estimating the CallableStatement object size 2002
18.17.5 Estimating the ResultSet object size 2003
18.17.6 Estimating the size of trace objects 2005

19. SQLJ 2007

19.1 Overview 2008
19.1.1 What is SQLJ? 2008
19.1.2 Environment settings 2010
19.2 SQLJ Translator 2012
19.3 UAP coding rule 2013
19.3.1 Labeling rule 2013
19.3.2 SQL coding rule 2013
19.3.3 SQL statements that can be used in SQLJ 2018
19.3.4 Correspondence between HiRDB data types and SQLJ data types 2020
19.3.5 Output variable settings (limited to the native interface edition) 2022
19.3.6 Using data types when a cursor is declared (limited to the native interface edition) 2024
19.3.7 Description of connection to and disconnection from a HiRDB server 2025
19.3.8 Description of cursor-based retrieval 2031
19.3.9 Receiving a dynamic result set 2036
19.3.10 Using JDBC and SQLJ together 2036
19.3.11 Creating and executing a UAP 2039
19.3.12 Migrating an SQLJ source from the standard interface edition to the native interface edition 2043
19.3.13 Notes about UAP development 2045
19.4 Native Runtime 2047
19.4.1 Package configuration 2047
19.4.2 Public classes of Native Runtime 2047
19.4.3 Cluster specifications 2048
19.4.4 Coding examples using the native interface 2054

Appendixes 2059

A. SQL Communications Area 2060
A.1 Organization and contents of the SQL Communications Area 2060
A.2 Expanding the SQL Communications Area 2066
B. SQL Descriptor Area 2069
B.1 Organization and contents of the SQL Descriptor Area 2069
B.2 Expanding the SQL Descriptor Area 2080
C. Column Name Descriptor Area 2093
C.1 Organization and contents of the Column Name Descriptor Area 2093
C.2 Expanding the Column Name Descriptor Area 2095
D. Type Name Descriptor Area 2097
D.1 Organization of the Type Name Descriptor Area 2097

D.2 Contents of the Type Name Descriptor Area 2097
D.3 Expanding the Type Name Descriptor Area 2098
E. Character Set Descriptor Area 2100
E.1 Organization and contents of the character set descriptor area 2100
E.2 Expanding the character set descriptor area 2110
F. SQL Data Types and Data Descriptions 2114
F.1 SQL data types and C data descriptions 2114
F.2 SQL data types and COBOL data descriptions 2135
G. Data Dictionary Table Retrieval 2159
G.1 Examples of SQL statements for retrieval 2163
G.2 Data dictionary table details 2166
H. Functions provided by HiRDB 2245
H.1 Hash function for table partitioning 2245
H.2 Space conversion function 2273
H.3 Function for conversion to a DECIMAL signed normalized number 2280
H.4 Character code type specification function 2282
I. Scalar Functions That Can Be Specified in the Escape Clause 2286
J. Character Code Conversion Rules When Character Sets Are Used 2292
J.1 Converting shift JIS kanji codes to EBCDIK 2292
J.2 Converting EBCDIK to shift JIS kanji codes 2294
K. HiRDB SQL Tuning Advisor Environment Setup 2297
L. Maximum and Minimum HiRDB Values 2298
M. List of Sample UAPs 2300
Index 2303

Chapter

1. Overview

This chapter explains the work flow for creating user application programs (UAPs), the characteristics of UAPs, and the interface between UAPs and the HiRDB system.

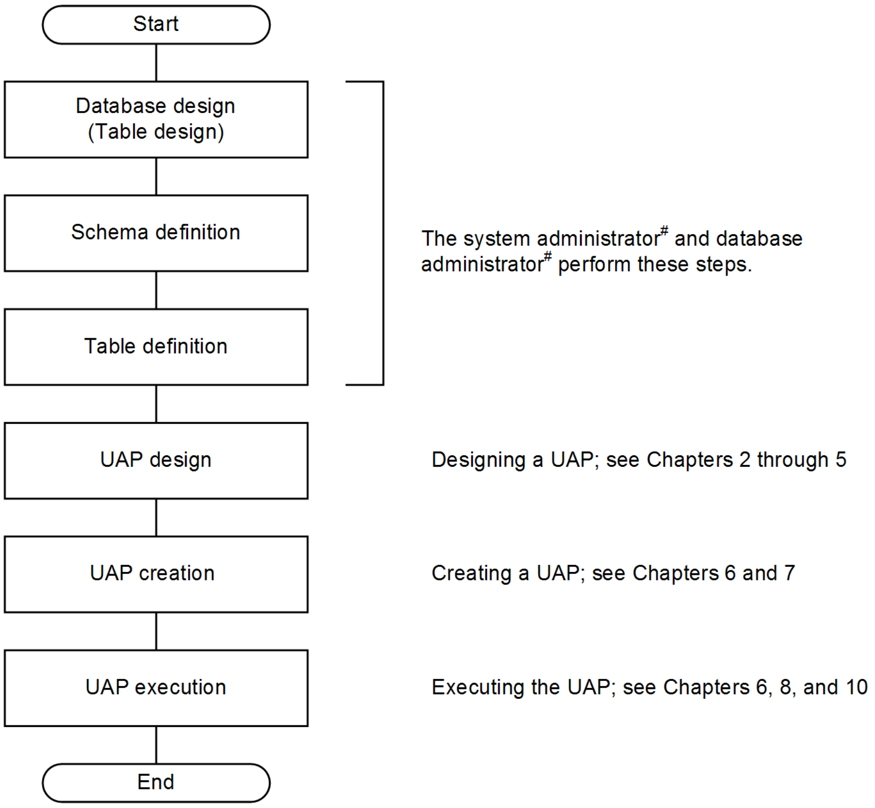
This chapter contains the following sections:

1.1 UAP development flow
1.2 UAP characteristics
1.3 Interface with HiRDB
1.4 UAP operation environment

1.1 UAP development flow

Before creating a user application program (UAP), the task requirements must be analyzed in order to create a database that is well suited to the data to be used in the task. Based on this analysis, you can estimate the overall database size and develop an outline of the UAP. The following figure shows the relationships between the elements of the UAP development tasks and the organization of this manual.

Figure 1-1: Relationships between UAP development task elements and the organization of this manual


Start
Database design
(Table design)
Schema definition
Table definition
UAP design
UAP creation
UAP execution
End
The system administrator# and database administrator# perform these steps.
Designing a UAP; see Chapters 2 through 5
Creating a UAP; see Chapters 6 and 7
Executing the UAP; see Chapters 6, 8, and 10

#: For work details, see *HiRDB Version 9 Installation and Design Guide*.

1.2 UAP characteristics

1.2.1 UAP format

To manipulate a HiRDB database, descriptions in the SQL language are embedded directly into a source program written in a high-level language.

The embedded method involves writing descriptions of a database language called SQL directly into a source program written in a high-level language. If you decide to create an embedded SQL UAP, program analysis becomes easy. All operations including the database operations (SQL) can be written as one program.

ODBC functions can also be specified in a UAP, and UAPs can also be created with Java^{TM} (SQLJ).

(1) Source program

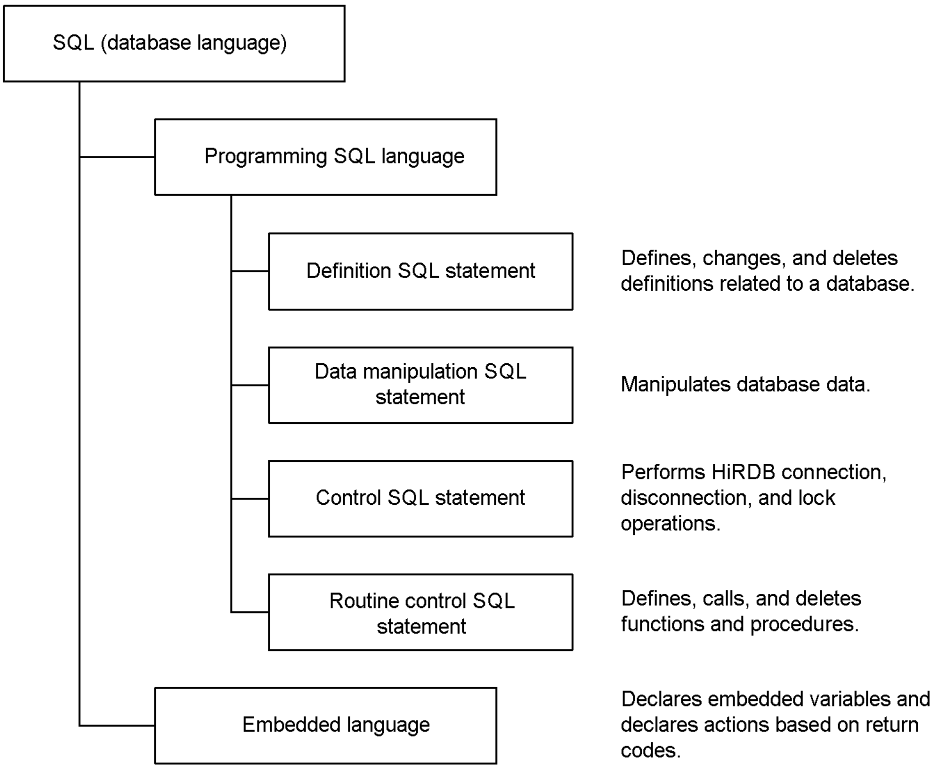
The following high-level languages can be used to write an embedded SQL UAP:

- C language
- C++ language
- COBOL language
- OOCOBOL language

(2) SQL

SQL is a database language for writing the definition, data manipulation, operation, and control instructions of a database. You can use these instructions by embedding them into a source program written in a high-level language. The following figure shows the SQL functional organization.

Figure 1-2: SQL functional organization


SQL (database language)
Programming SQL language
Definition SQL statement
Defines, changes, and deletes definitions related to a database.
Data manipulation SQL statement
Manipulates database data.
Control SQL statement
Performs HiRDB connection, disconnection, and lock operations.
Routine control SQL statement
Defines, calls, and deletes functions and procedures.
Embedded language
Declares embedded variables and declares actions based on return codes.

For an overview of the SQL language types and functions for programs, see *1.2.2 List of SQL statements usable in HiRDB*. For details about the embedded language, see the manual *HiRDB Version 9 SQL Reference*.

1.2.2 List of SQL statements usable in HiRDB

Tables 1-1 to *1-5* list the SQL statements that can be used in HiRDB. In the table headings, *OLTP* refers to an application program that complies with X/Open in the OLTP environment.

For details about the following items, refer to the indicated manuals or locations:

Details about the SQL coding formats

HiRDB Version 9 SQL Reference

Database definitions

HiRDB Version 9 Installation and Design Guide

Database operations

2. Database Operations

Database management

3. UAP Design

Embedded language

HiRDB Version 9 SQL Reference

Table 1-1: List of SQL statements (definition SQL)

SQL	**Function**	**Usability**		
		C	**COBOL**	**OLTP**
`ALLOCATE MEMORY TABLE` (specify a table to be expanded to the memory database)	Specifies a table defined by `CREATE TABLE` as a table to be expanded to the memory database.	U	U	--
`ALTER INDEX` (change index definition)	Renames an index.	U	U	--
`ALTER PROCEDURE` (re-create SQL object for procedure)	Re-creates an SQL object for a procedure.	U	U	--
`ALTER ROUTINE` (re-create SQL object for function, procedure, or trigger)	Re-creates an SQL object for a function or procedure.	U	U	--
`ALTER TABLE` (alter table definition)	• Adds a new column to end of a base table. • Changes a data type. • Increases the maximum length of an existing column of the variable-length data type. • Deletes a base table column that contains no data. • Changes the uniqueness constraint for cluster keys for a base table containing no data. • Renames table and columns.	U	U	--
`ALTER TRIGGER` (re-create SQL object for trigger)	Re-creates an SQL object for a trigger.	U	U	--

SQL	Function	Usability		
		C	COBOL	OLTP
COMMENT (add comment)	Provides a comment in a table or column.	U	U	--
CREATE AUDIT (define audit event)	Defines an audit event to be recorded as an audit trace and its target.	U	U	--
CREATE CONNECTION SECURITY (define connection security facility)	Defines the security item related to the connection security facility.	U	U	--
CREATE FUNCTION (define function)	Defines a function.	U	U	--
CREATE PUBLIC FUNCTION (define public function)	Defines a public function.	U	U	--
CREATE INDEX (define index)	Defines an index (ascending or descending order) for columns in a base table.	U	U	--
CREATE PROCEDURE (define procedure)	Defines a procedure.	U	U	--
CREATE PUBLIC PROCEDURE (define public procedure)	Defines a public procedure.	U	U	--
CREATE SCHEMA (define schema)	Defines a schema.	U	U	--
CREATE SEQUENCE (define sequence generator)	Defines a sequence generator.	U	U	--
CREATE TABLE (define base table)	Defines a base table.	U	U	--
CREATE TRIGGER (define trigger)	Defines a trigger.	U	U	--
CREATE TYPE (define type)	Defines an abstract data type.	U	U	--
CREATE VIEW (define view table)	Defines a view table.	U	U	--

SQL	Function	Usability		
		C	COBOL	OLTP
CREATE PUBLIC VIEW (define public view)	Defines a public view table.	U	U	--
DEALLOCATE MEMORY TABLE (release table expanded to memory database)	Releases a table expanded to the memory database.	U	U	--
DROP AUDIT (delete audit event)	Deletes the definition that matches the audit event and contents defined by CREATE AUDIT from the audit targets.	U	U	--
DROP CONNECTION SECURITY (delete connection security facility)	Deletes the security item related to the connection security facility.	U	U	--
DROP DATA TYPE (delete user-defined type)	Deletes a user-defined type.	U	U	--
DROP FUNCTION (delete function)	Deletes a function.	U	U	--
DROP PUBLIC FUNCTION (delete public function)	Deletes a public function.	U	U	--
DROP INDEX (delete index)	Deletes an index.	U	U	--
DROP PROCEDURE (delete procedure)	Deletes a procedure.	U	U	--
DROP PUBLIC PROCEDURE (delete public procedure)	Deletes a public procedure.	U	U	--
DROP SCHEMA (delete schema)	Deletes a schema.	U	U	--
DROP SEQUENCE (delete sequence generator)	Deletes a sequence generator.	U	U	--

SQL	Function	Usability		
		C	COBOL	OLTP
DROP TABLE (delete table)	Deletes a base table, as well as any indexes, comments, access privileges, view tables, and trigger associated with the base table.	U	U	--
DROP TRIGGER (delete trigger)	Deletes a trigger.	U	U	--
DROP VIEW (delete view table)	Deletes a view table.	U	U	--
DROP PUBLIC VIEW (delete public view table)	Deletes a public view table.	U	U	--
GRANT AUDIT (change auditor's password)	Changes the auditor's password.	U	U	--
GRANT CONNECT (grant CONNECT privilege)	Grants the CONNECT privilege to users.	U	U	--
GRANT DBA (grant DBA privilege)	Grants the DBA privilege to users.	U	U	--
GRANT RDAREA (grant RDAREA usage privilege)	Grants the RDAREA usage privilege to users.	U	U	--
GRANT SCHEMA (grant schema definition privilege)	Grants the schema definition privilege to users.	U	U	--
GRANT *access-privilege* (grant access privileges)	Grants access privileges to users.	U	U	--
REVOKE CONNECT (revoke CONNECT privilege)	Revokes previously granted CONNECT privileges.	U	U	--
REVOKE DBA (revoke DBA privilege)	Revokes previously granted DBA privileges.	U	U	--
REVOKE RDAREA (revoke RDAREA usage privilege)	Revokes previously granted RDAREA usage privileges.	U	U	--
REVOKE SCHEMA (revoke schema definition privilege)	Revokes previously granted schema definition privileges.	U	U	--

SQL	Function	Usability		
		C	COBOL	OLTP
REVOKE *access-privilege* (revoke access privilege)	Revokes previously granted access privileges.	U	U	--

U: Can be used.

--: Cannot be used.

Table 1-2: List of SQL statements (data manipulation SQL)

SQL	Function	Usability		
		C	COBOL	OLTP
ALLOCATE CURSOR statement (allocate cursor)	Allocates a cursor for a SELECT statement preprocessed by the PREPARE statement or for a group of result sets returned by a procedure.	U	U	U
ASSIGN LIST statement (create list)	Creates a list from a base table.	U	U	U
CALL statement# (call procedure)	Calls a procedure.	U	U	U
CLOSE statement (close cursor)	Closes a cursor.	U	U	U
DEALLOCATE PREPARE statement (release preprocessing)	Releases the allocation of an SQL statement preprocessed by the PREPARE statement.	U	U	U
DECLARE CURSOR (declare cursor)	Declares a cursor that the results of a retrieval by the SELECT statement can be fetched row by row with the FETCH statement.	U	U	U
DELETE statement (delete rows)	Deletes either the rows that satisfy specified search conditions or the row indicated by the cursor.	U	U	U
Preparable dynamic DELETE statement: locating (delete row that uses preprocessable cursor)	Deletes the row indicated by the specified cursor. This statement is used for dynamic execution.	U	U	U

SQL	Function	Usability		
		C	COBOL	OLTP
DESCRIBE statement (receive retrieval and I/O information)	Returns to the SQL Descriptor Area SQL retrieval information, output information, or input information that was preprocessed by the PREPARE statement.	U	U	U
DESCRIBE CURSOR statement (receive retrieval information for cursor)	Returns to the SQL Descriptor Area retrieval information for a cursor that can reference a result set returned by a procedure.	U	U	U
DESCRIBE TYPE statement (receive definition information for user-defined type)	Receives in the SQL Descriptor Area the definition information (including data codes for all attributes and data length) for a user-defined type. The user-defined type has been directly or indirectly included in the SQL retrieval item information that was preprocessed by the PREPARE statement.	U	U	U
DROP LIST statement (delete list)	Deletes a list.	U	U	U
EXECUTE statement (execute SQL)	Executes an SQL statement preprocessed by the PREPARE statement.	U	U	U
EXECUTE IMMEDIATE statement (preprocess and execute SQL)	Preprocesses and executes an SQL statement provided in a character string.	U	U	U
FETCH statement (fetch data)	Advances the cursor to the next row to be fetched, and reads column values in that row into the embedded variable specified in the INTO clause.	U	U	U
FREE LOCATOR statement (invalidate locator)	Invalidates a locator.	U	U	U

SQL	Function	Usability		
		C	COBOL	OLTP
INSERT statement (insert rows)	Inserts rows into a table. A single row can be inserted by direct specification of values; one or more rows can be inserted by using the SELECT statement.	U	U	U
OPEN statement (open cursor)	Opens a cursor. The cursor declared by DECLARE CURSOR or allocated by ALLOCATE CURSOR is positioned immediately preceding the first line of the retrieval results so that the retrieval results can be fetched.	U	U	U
PREPARE statement (preprocess SQL statement)	Preprocesses the SQL statement provided in a character string so that the statement can be executed and assigns a name (SQL statement identifier or extended statement name) to that SQL statement.	U	U	U
PURGE TABLE statement (delete all rows)	Deletes all rows in a base table.	U	U	--
Single-row SELECT statement (retrieve one row)	Searches table data. To fetch only one row of data from a table, the single-row SELECT statement can be used without having to declare a cursor.	U	U	U
Dynamic SELECT statement (retrieve dynamically)	Searches table data. The dynamic SELECT statement is preprocessed by the PREPARE statement. During the search, a cursor declared by DECLARE CURSOR or allocated by ALLOCATE CURSOR is used to fetch the retrieval results row by row.	U	U	U
UPDATE statement (update data)	Updates the values of columns in the rows that satisfy specified search conditions or in the row indicated by the cursor.	U	U	U

SQL	Function	Usability		
		C	COBOL	OLTP
Preparable dynamic UPDATE statement: locating (update data that uses preprocessable cursor)	Updates the value of the specified column in the row indicated by the specified cursor. This statement is used for dynamic execution.	U	U	U
Assignment statement (assign value)	Assigns a value to an SQL variable or SQL parameter.	U	U	U

U: Can be used.

--: Cannot be used.

#: A procedure that is called under OLTP cannot be executed if it contains a PURGE TABLE, COMMIT, or ROLLBACK statement.

Table 1-3: List of SQL statements (control SQL)

SQL	Function	Usability		
		C	COBOL	OLTP
CALL COMMAND statement (execute command or utility)	Executes a HiRDB command or utility.	U	U	U
COMMIT statement (terminate transaction normally)	Terminates the current transaction normally, sets synchronization points, generates one unit of commitment, and effects the database updates performed by the transaction.	U	U	--
CONNECT statement (connect UAP to HiRDB)	Passes the authorization identifier and password to HiRDB, and enables the UAP to use HiRDB.	U	U	--
DISCONNECT statement (disconnect UAP from HiRDB)	Terminates the current transaction normally, sets synchronization points, and generates one unit of commitment, then disconnects the UAP from HiRDB.	U	U	--
LOCK statement (lock control on tables)	Performs exclusive locks on specified tables.	U	U	U

SQL	Function	Usability		
		C	COBOL	OLTP
ROLLBACK statement (cancel transaction)	Cancels the current transaction and nullifies the database updating performed by the transaction.	U	U	--
SET SESSION AUTHORIZATION statement (change execution user)	Changes the user who is currently connected.	U	U	U

U: Can be used.

--: Cannot be used.

Table 1-4: List of SQL statements (embedded language)

SQL	Function	Usability		
		C	COBOL	OLTP
BEGIN DECLARE SECTION (declare beginning embedded SQL)	Indicates the beginning of an embedded SQL declare section, that specifies the embedded variables and indicator variables used in the SQL.	U	U	U
END DECLARE SECTION (declare end of embedded SQL)	Indicates the end of an embedded SQL declare section.	U	U	U
ALLOCATE CONNECTION HANDLE (allocate connection handle)	Allocates a connection handle to be used by the UAP in an environment that uses multiple connection functions.	U	U	--
FREE CONNECTION HANDLE (free connection handle)	Frees a connection handle that was allocated by ALLOCATE CONNECTION HANDLE.	U	U	--
DECLARE CONNECTION HANDLE SET (declare connection handle to be used)	Declares a connection handle to be used by the UAP SQL in an environment that uses the multi-connection facility.	U	U	U#
DECLARE CONNECTION HANDLE UNSET (cancel all connection handles being used)	Cancels all declarations of connection handle use specified with DECLARE CONNECTION HANDLE SET statements before this statement.	U	--	--

SQL	Function	Usability		
		C	COBOL	OLTP
GET CONNECTION HANDLE (get connection handle)	Allocates the connection handle to be used by the UAP when the multi-connection facility is to be used in an X/Open XA interface environment.	U	U	U#
COPY (copy cataloged text)	Copies cataloged text into a source program.	--	U	U
GET DIAGNOSTICS (get diagnostic information)	If the preceding SQL statement is CREATE PROCEDURE or CALL, obtains error information and diagnostic information from the diagnostics area.	U	U	U
COMMAND EXECUTE (execute commands from UAP)	Executes HiRDB and OS commands from inside the UAP.	U	--	--
SQL prefix	Indicates the beginning of SQL statements.	U	U	U
SQL terminator	Indicates the end of SQL statements.	U	U	U
WHENEVER (declare embedded exception)	Declares UAP processing, based on the return code set by HiRDB in the SQL Communications Areas after SQL statements have been executed.	U	U	U
SQLCODE variable	Receives the return code issued by HiRDB after an SQL statement has been executed.	U	U	U
SQLSTATE variable	Receives the return code issued by HiRDB after an SQL statement has been executed.	U	U	U
Declaration of PDCNCTHDL-type variable	Declares a connection handle type variable to be used in an environment that uses the multi-connection facility.	U	--	--
INSTALL JAR (register JAR file)	Installs the JAR file in the HiRDB server.	U	--	--

SQL	Function	Usability		
		C	COBOL	OLTP
REPLACE JAR (re-register JAR file)	Replaces the JAR file in the HiRDB server.	U	--	--
REMOVE JAR (delete JAR file)	Uninstalls the JAR file from the HiRDB server.	U	--	--

U: Can be used.

--: Cannot be used.

#: The statement can be used if a connection handle was allocated with the GET CONNECTION HANDLE statement.

Table 1-5: List of SQL statements (routine control SQL)

SQL	Function	Usability		
		C	COBOL	OLTP
Compound statement (execute multiple statements)	Executes a group of SQL statements as a single SQL statement.	PFT	PFT	--
IF statement (execute by conditional branching)	Executes the SQL statement that satisfies a set of specified conditions.	PFT	PFT	--
RETURN statement (return return value)	Returns the return value of a function.	PFT#1	PFT#1	--
WHILE statement (repeat statements)	Executes a set of SQL statements repetitively.	PFT	PFT	--
FOR statement (repeat execution of each row)	Repeats execution of an SQL statement for each row in a table.	PFT#3	PFT#3	--
LEAVE statement (leave statement)	Exits from a compound statement or the WHILE statement and terminates execution of the statement.	PFT	PFT	--
WRITE LINE statement (output character string to file)	Outputs a character string of the specified value expression to a file.	PFT	PFT	--
SIGNAL statement (report error)	Triggers an error and reports it.	PFT#2	PFT#2	--

SQL	Function	Usability		
		C	COBOL	OLTP
RESIGNAL statement (re-report error)	Triggers an error and reports it again.	PFT[#2]	PFT[#2]	--

PFT: The statement cannot be used directly in the UAP. However, the statement can be used to define an SQL procedure, SQL function, or trigger operation in the CREATE PROCEDURE, CREATE FUNCTION, or CREATE TRIGGER statement.

--: Cannot be used.

Note

In procedure definitions, the SQL statements that can be specified in addition to the routine control SQL statements are the CALL, CLOSE, DECLARE CURSOR, DELETE, FETCH, INSERT, OPEN, PURGE TABLE, single-row SELECT, UPDATE, COMMIT, LOCK, and ROLLBACK statements. In functions, SQL statements other than routine control SQL statements cannot be specified.

#1: This statement cannot be used if an SQL procedure or a trigger operation is defined in the CREATE PROCEDURE or CREATE TRIGGER statement.

#2: This statement cannot be used if an SQL function is defined in the CREATE FUNCTION statement.

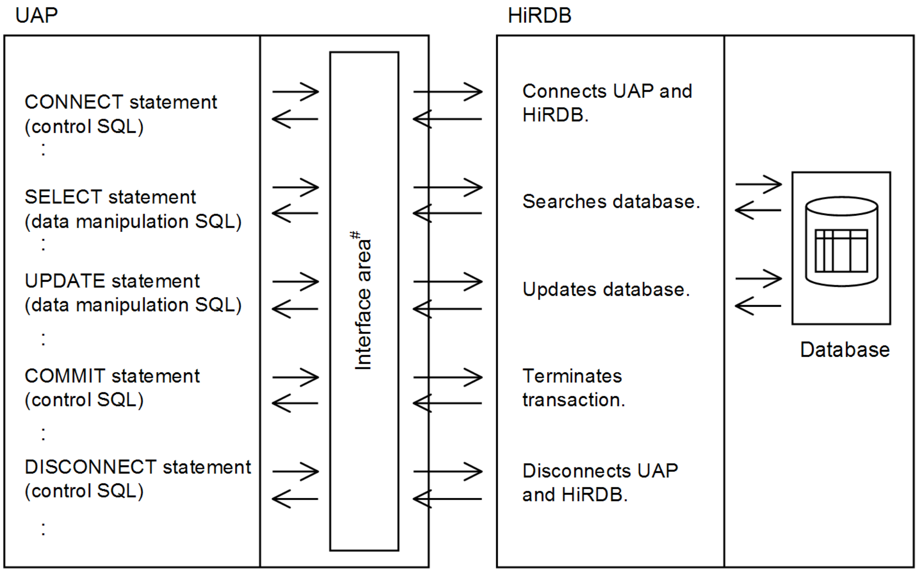
#3: This statement cannot be used in the CREATE FUNCTION statement.

1.3 Interface with HiRDB

To manipulate a HiRDB database, create a UAP. The UAP issues SQL statements and uses the interface area to exchange information with HiRDB.

The following figure shows the interface between a UAP and HiRDB.

Figure 1-3: Interface between a UAP and HiRDB



#: For details about the interface area, see *3.2.2 Interface areas*.

1.4 UAP operation environment

HiRDB operates in a client/server network environment. The unit used to send a request for executing a UAP is called the *client*, and the unit used to receive a request is called the *server*. The system used as the server is the HiRDB server.

A client can operate in any combination of these eight modes:

- Operating mode in which a machine other than the server machine is used as the client
- Operating mode in which the same server machine as the HiRDB server is used as the client
- Operating mode in which a UAP under On-Line Transaction Processing (OLTP) is used as the client
- Operating mode in which an ODBC[#1]-compatible application program is used as the client
- Operating mode in which an OLE DB[#2]-compatible application program is used as the client
- Operating mode in which an ADO.NET-compatible application program is used as the client
- Operating mode in which a Java (JDBC-compatible) application program is used as the client
- Operating mode in which a VOS3 system or Linux for AP8000[#3] UAP is used as the client

#1: ODBC refers to a database access mechanism advocated by Microsoft Corporation. For details about how to access HiRDB from an ODBC-compatible UAP, see *14. HiRDB Access from ODBC Application Programs*.

#2: Like ODBC, OLE DB is an API for accessing a wide range of data sources. Unlike ODBC, OLE DB also defines interfaces for accessing non-SQL data. For details about how to access HiRDB from an OLE DB-compatible UAP, see *15. HiRDB Access from OLE DB Application Programs*.

#3: Linux for AP8000 operates with HiRDB/Developer's Kit Version 6.

Figures 1-4 to *1-11* show the client operation modes.

Use the same platform for the HiRDB/Developer's Kit used to create the UAP and the HiRDB/Developer's Kit used to execute the UAP.

Figure 1-4: Operating mode using a machine other than the server machine as a client

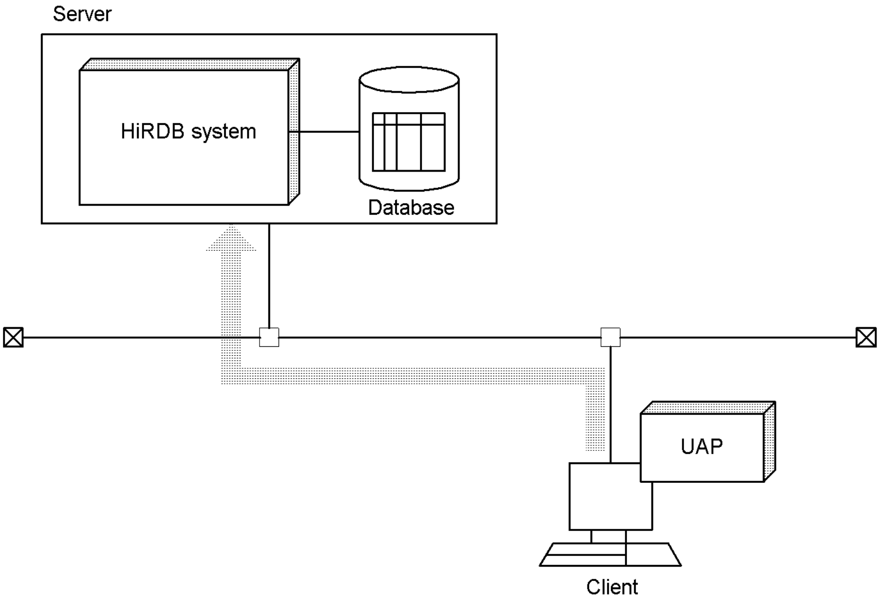


Figure 1-5: Operating mode using the same server machine as the HiRDB server as the client

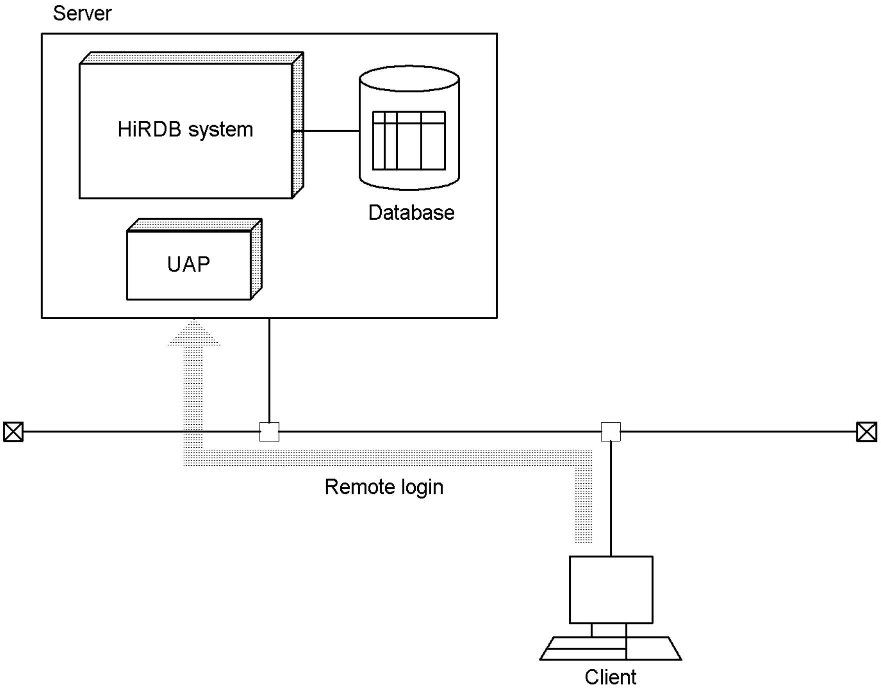


Figure 1-6: Operating mode using a UAP under OLTP as a client

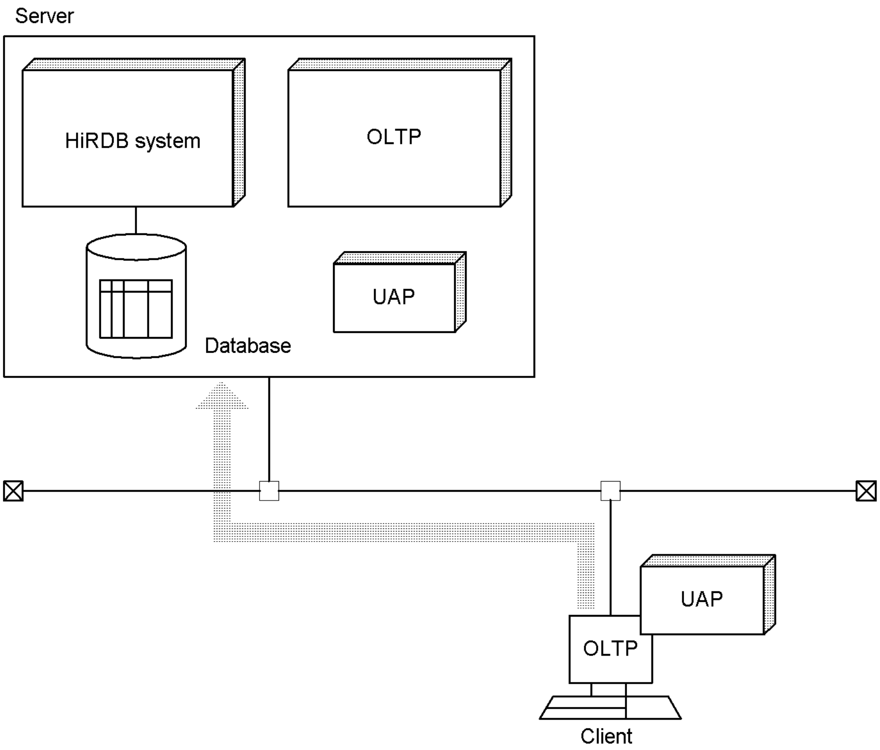
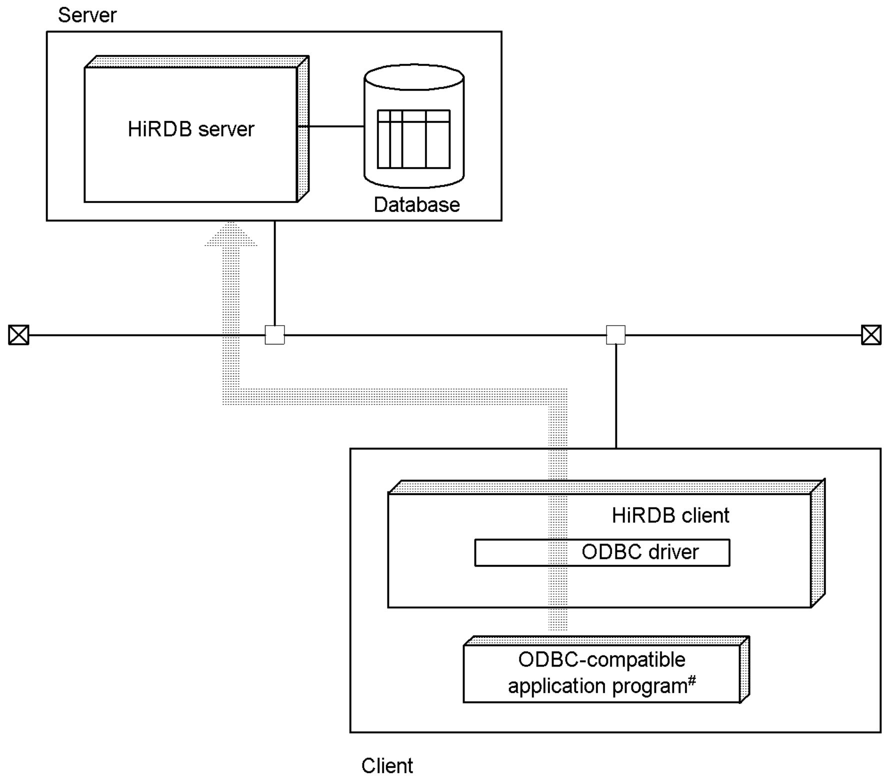
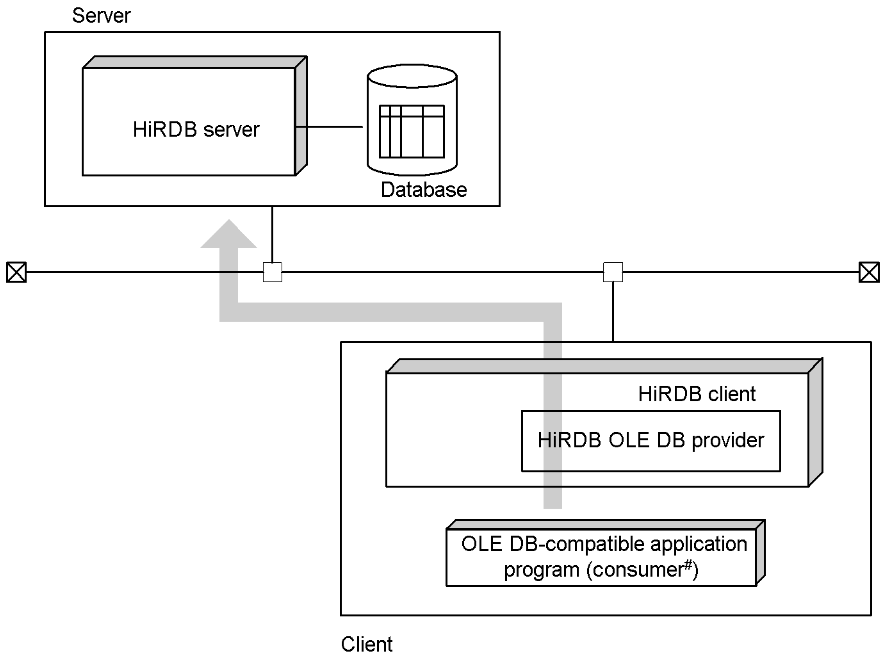


Figure 1-7: Operating mode using an ODBC-compatible UAP as a client



ODBC compatible UAPs are included.

Figure 1-8: Operating mode using an OLE DB-compatible UAP as a client



A consumer refers to a program that calls the OLE DB method and interface.

Figure 1-9: Operating mode using an ADO.NET-compatible UAP as a client

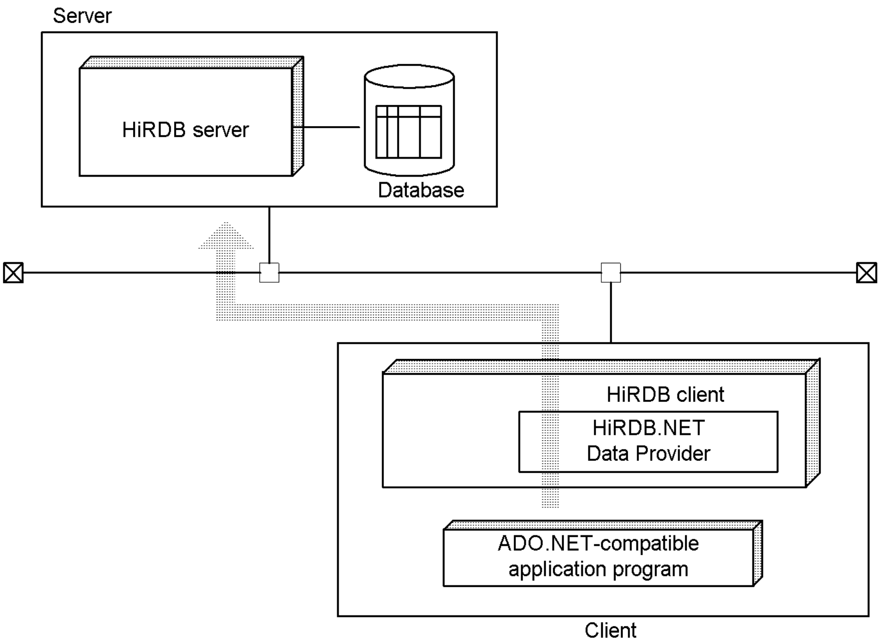


Figure 1-10: Operating mode using a Java (JDBC-compatible) application program as a client

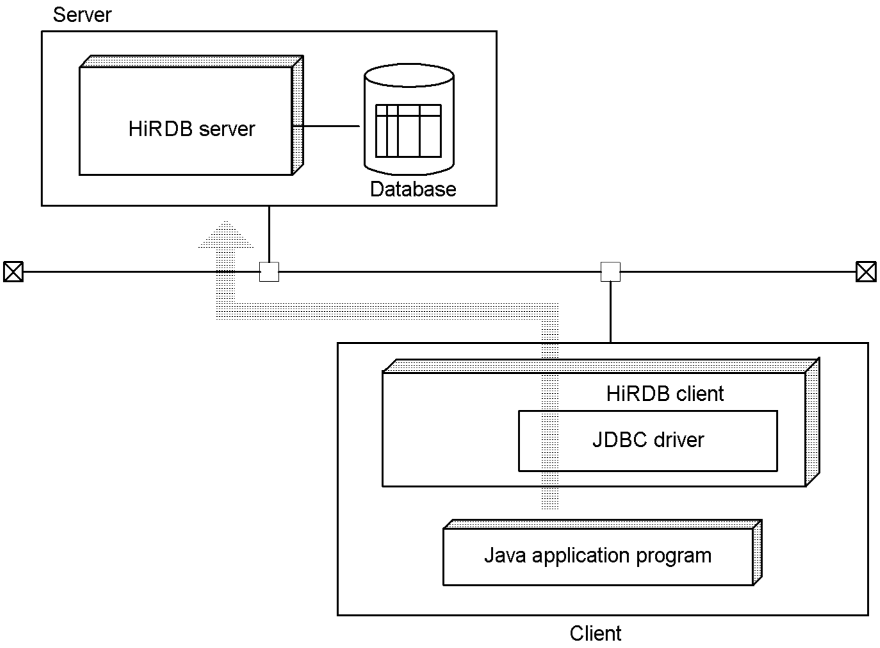
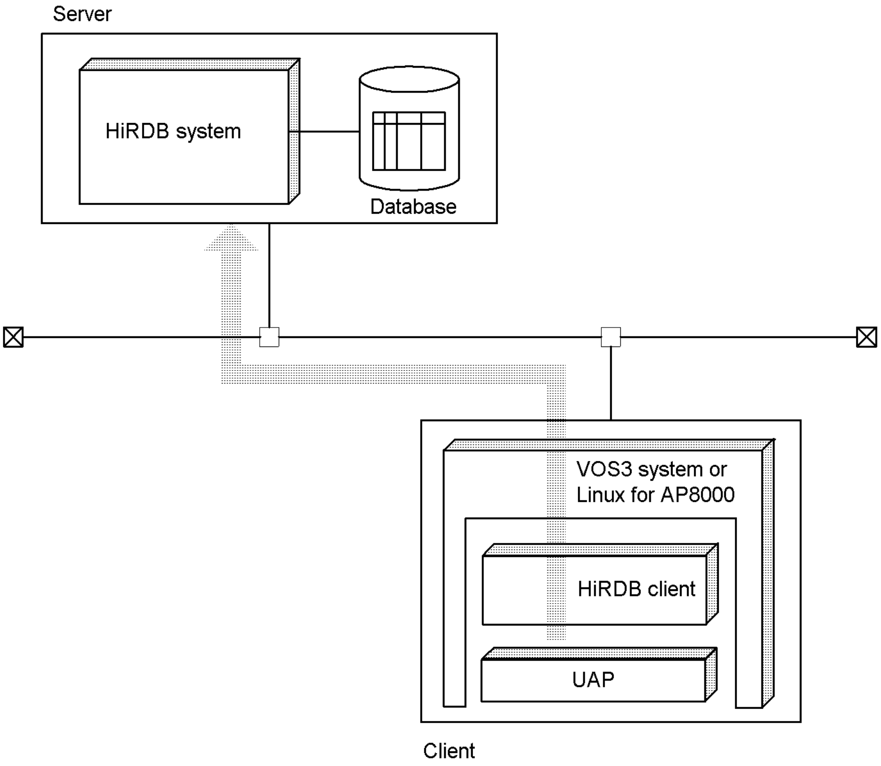


Figure 1-11: Operating mode using a VOS3 system or Linux for AP8000 UAP as a client



Chapter

2. Database Operations

This chapter explains the data expressions used in a database and provides examples of basic database operations.

The SQL statements used in the examples are excerpts from the complete SQL statements written according to the prescribed syntax; for details about SQL statements, see the *HiRDB Version 9 SQL Reference* manual.

This chapter contains the following sections:

2.1 Database data expressions
2.2 Cursor usage
2.3 Data retrieval
2.4 Data updating
2.5 Data deletion
2.6 Data insertion
2.7 Specific data search
2.8 Data operations
2.9 Data processing
2.10 Outer joining of tables
2.11 Defining and manipulating a view table
2.12 Manipulating data in a table with abstract data types

2.1 Database data expressions

2.1.1 Relational database tables

A HiRDB database is a relational database whose logical structure is expressed by tables. This section explains tables.

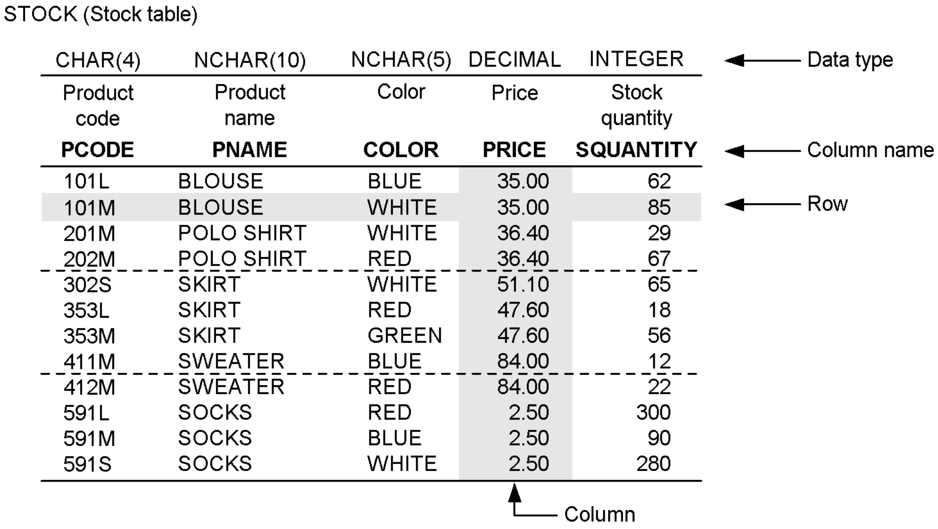
(1) Basic table configuration

A relational database is expressed logically by tables.

The values in the vertical and horizontal directions of a table are called *columns* and *rows*, respectively. The values within a column represent data with the same attribute, that is, the same data type. A table consists of a set of rows; the row is the basic unit for retrievals. Each column is assigned a name (column name) that is used for database manipulations.

The following figure shows an example of a basic table configuration. Ending zeros in the `PRICE` column (in this example and throughout the manual) are not displayed on the actual screen.

Figure 2-1: Basic table configuration example



STOCK (Stock table)

CHAR(4)	NCHAR(10)	NCHAR(5)	DECIMAL	INTEGER
Product code	Product name	Color	Price	Stock quantity
PCODE	**PNAME**	**COLOR**	**PRICE**	**SQUANTITY**
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

(2) Tables that use repetition columns

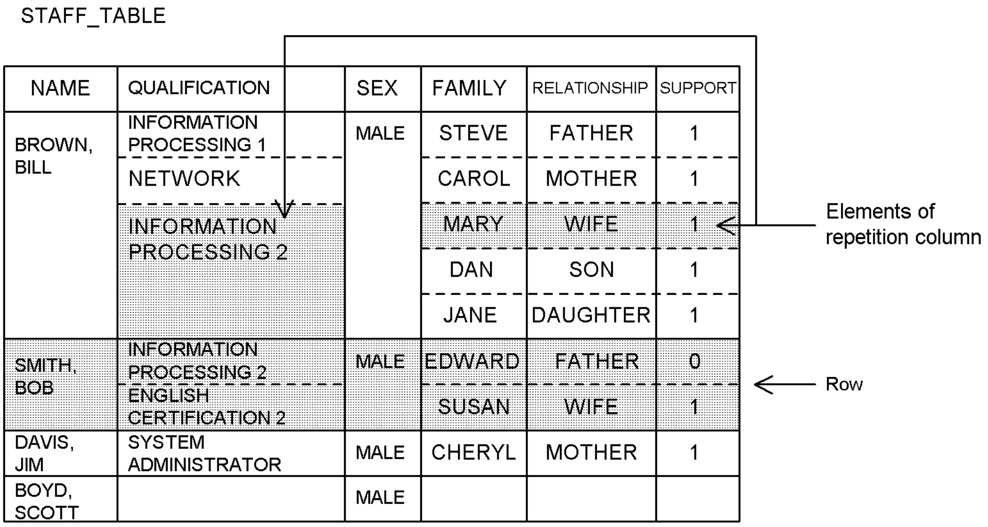
A repetition column refers to a column that consists of multiple elements. Using repetition columns has the following advantages:

- Multiple tables do not have to be joined.

- Less disk space is used because many duplicate information items are eliminated.
- Access performance is better because related data items (repetition data items) are stored near each other rather than in separate tables.

The following figure shows a configuration example of a table that has repetition columns.

Figure 2-2: Configuration example of a table with repetition columns



STAFF_TABLE

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 1 NETWORK INFORMATION PROCESSING 2	MALE	STEVE CAROL MARY DAN JANE	FATHER MOTHER WIFE SON DAUGHTER	1 1 1 1 1
SMITH, BOB	INFORMATION PROCESSING 2 ENGLISH CERTIFICATION 2	MALE	EDWARD SUSAN	FATHER WIFE	0 1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

Note: The blank locations represent null values.

(3) View table

A virtual table that limits the range of columns or rows that can be manipulated by the user can be created based on an actual table (referred to hereafter as a *base table*). Such a virtual table is called a *view table*. A view table can be defined for the following purposes, thus restricting the manipulation range and simplifying operations:

- To retrieve only certain columns of a table
- To change the order of the columns in a table
- To retrieve only certain rows of a table

Although a view table is usually defined to view only selected columns or rows of a table, it can be retrieved in the same way as a base table. Because use of a view table restricts the range of manipulations that are possible, precise security measures can be implemented by means of view tables.

The following figure shows an example of a view table created from a base table.

For details about how to define and manipulate a view table, see *2.11 Defining and* *manipulating a view table.*

Figure 2-3: Example of a base table and view table

Base table

ORDER (Order table)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received date	Order received time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

View table

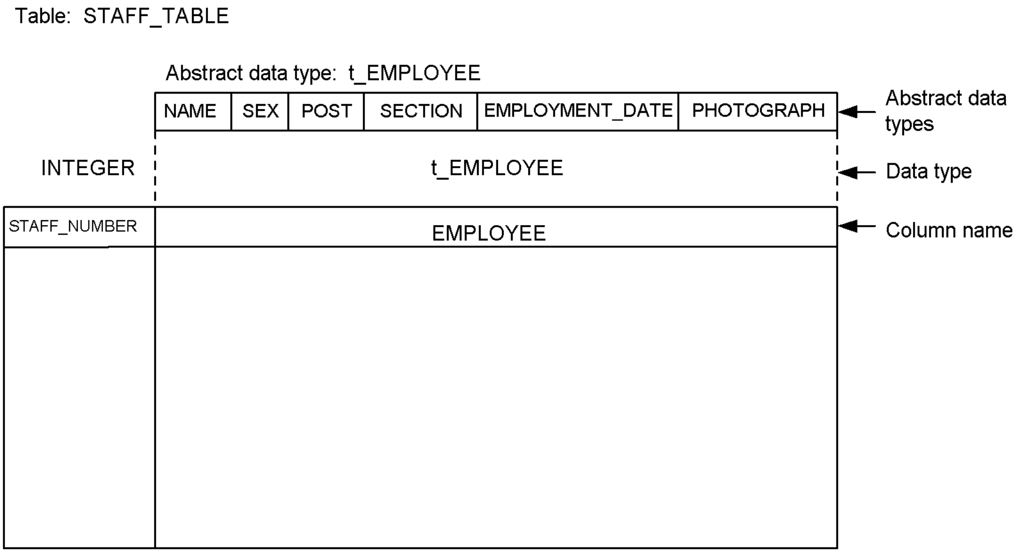
FNO	PCODE	OQUANTITY
Form number	Product code	Ordered quantity
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10
026557	411M	5
026558	412M	4
026559	591M	80
026560	591L	10

2.1.2 Object relational database tables

The HiRDB database can also be defined as an *object relational database*. An object relational database table can be created by defining abstract data types in the table columns.

The following figure shows a basic configuration example of a table that has abstract data types.

Figure 2-4: Basic configuration example of a table with abstract data types



2.2 Cursor usage

Table retrieval results usually consist of multiple rows. A cursor is used by the UAP to retrieve rows one at a time from the entire set of retrieved rows.

This section explains how to retrieve data using a cursor and how to use the cursor to update a retrieved row.

For details about how to use a cursor, see *3.5 Use of a cursor*.

(1) Retrieval using a cursor

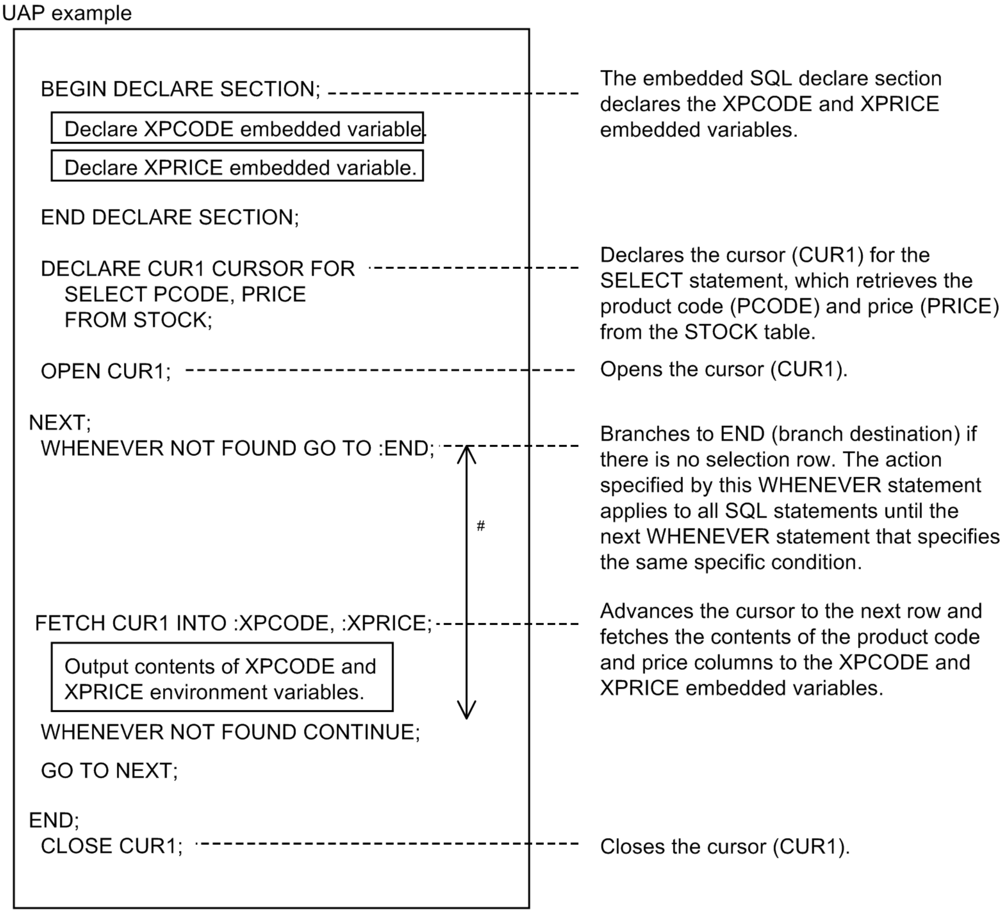
When table retrieval results consist of multiple rows or when retrieving data dynamically after preprocessing the SQL statement with the `PREPARE` statement, a cursor is used to retrieve the individual rows.

When retrieval results consist of one or fewer rows, it is possible to use the single row `SELECT` statement for retrieval instead of a cursor.

For details about the `PREPARE` and single row `SELECT` statements, see the *HiRDB Version 9 SQL Reference* manual.

As an example of using a cursor to retrieve multiple rows, the UAP below retrieves product codes and unit prices from a stock table:

UAP example



```
BEGIN DECLARE SECTION;
  Declare XPCODE embedded variable.
  Declare XPRICE embedded variable.
END DECLARE SECTION;

DECLARE CUR1 CURSOR FOR
  SELECT PCODE, PRICE
  FROM STOCK;

OPEN CUR1;

NEXT;
 WHENEVER NOT FOUND GO TO :END;

 FETCH CUR1 INTO :XPCODE, :XPRICE;
  Output contents of XPCODE and
  XPRICE environment variables.
 WHENEVER NOT FOUND CONTINUE;

 GO TO NEXT;

END;
 CLOSE CUR1;
```

The embedded SQL declare section declares the XPCODE and XPRICE embedded variables.

Declares the cursor (CUR1) for the SELECT statement, which retrieves the product code (PCODE) and price (PRICE) from the STOCK table.

Opens the cursor (CUR1).

Branches to END (branch destination) if there is no selection row. The action specified by this WHENEVER statement applies to all SQL statements until the next WHENEVER statement that specifies the same specific condition.

Advances the cursor to the next row and fetches the contents of the product code and price columns to the XPCODE and XPRICE embedded variables.

Closes the cursor (CUR1).

Note
The SQL prefix and the SQL terminator have been omitted.
This is the effective range of WHENEVER NOT FOUND GO TO :END;.

(2) Using a cursor to update the row retrieved

When multiple rows are retrieved, a cursor is used to update the rows one at a time.

Although the single row `SELECT` statement can be used to update a retrieval that consists of one row or less, use of a cursor results in better processing efficiency.

As an example of using a cursor to update rows one at a time, the UAP below reduces the unit price of each product in the stock table by 10% (multiplies by 0.9):

UAP example

```
BEGIN DECLARE SECTION; ----------------- The embedded SQL declare section
                                          declares the XPRICE and PPRICE
  Declare XPRICE embedded variable.       embedded variables.
  Declare PPRICE embedded variable.
END DECLARE SECTION;

DECLARE CUR1 CURSOR FOR ---------------- Declares the cursor (CUR1) for fetching the
  SELECT PRICE FROM STOCK                 price column retrieval results one row at a
  FOR UPDATE OF PRICE;                    time from the STOCK table.

OPEN CUR1;------------------------------ Opens the cursor (CUR1).

NEXT;
WHENEVER NOT FOUND GO TO :END; --------- Branches to END (branch destination) if
                                          there is no selection row.

FETCH CUR1 INTO :XPRICE; --------------- Advances the cursor to the next row and
                                          reads the value of the price column to the
                                          XPRICE embedded variable.

WHENEVER NOT FOUND CONTINUE; ----------- Executes the next instruction if there is no
                                          row to process.
  Multiply price of XPRICE embedded
  variable by 0.9 and set result to
  PPRICE embedded variable.

UPDATE STOCK SET PRICE=:PPRICE --------- Updates the price column value in the row
  WHERE CURRENT OF CUR1;                  where the cursor is positioned with the value
GOTO NEXT;                                of the PPRICE embedded variable.

END;
CLOSE CUR1; ---------------------------- Closes the cursor (CUR1).
```

Note
The SQL prefix and the SQL terminator have been omitted.

(3) Retrieval without using a cursor (single row retrieval)

As an example of a retrieval that does not use a cursor, the UAP below makes a count of the items in the stock table and sets the results in an embedded variable.

UAP example

```
SELECT COUNT(*) INTO :PCOUNT ----------- Fetches the count from the
  FROM STOCK                              stock table (STOCK) into
                                          the PCOUNT embedded
                                          variable.
```

2.3 Data retrieval

Selecting those rows that satisfy a condition specified for a particular column is called a *retrieval*. You can also join and search two or more tables based on the values of a specific column and obtain a single set of retrieval results.

This section describes retrieval from one or more tables.

2.3.1 Retrieval from a single table

The following figure shows, as an example of retrieval from a single table, the use of a SELECT statement to retrieve from a stock table the rows that contain SKIRT as the product name.

Figure 2-5: Retrieval from a single table

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT * FROM STOCK
   WHERE PNAME=N'SKIRT'
```

Retrieval results

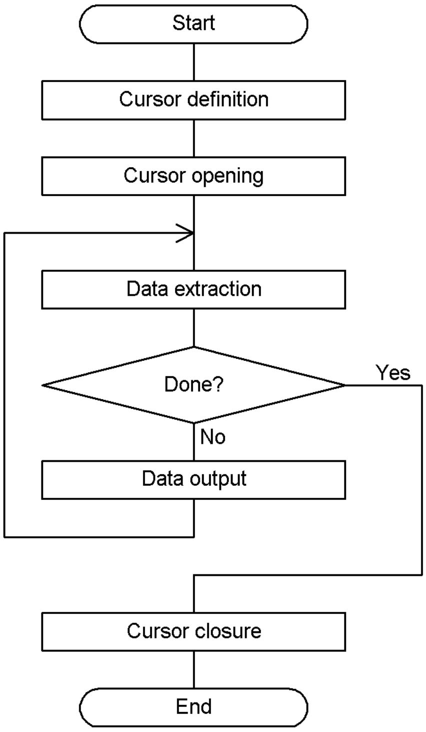
PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

The table retrieval results are expressed as a table and passed to the UAP that requested the processing.

A cursor is then used by the UAP to reference the retrieval results table. Because a cursor can point to a specific row in the retrieval results table, the UAP is able to read the contents of the row being indicated by the cursor and process that row's contents.

The following figure shows a sequence in which a UAP processes data from a retrieval results table.

Figure 2-6: UAP data processing sequence for a retrieval results table



The processing steps shown in *Figure 2-6* are explained as follows.

1. Cursor definition

   To use a cursor, a cursor name, the name of the table to be retrieved using the cursor, and retrieval conditions are defined. For the example in *Figure 2-5*, the following definitions define the cursor name as `CUR1` and specify that `SKIRT` only is to be retrieved from the stock table:

```
DECLARE CUR1 CURSOR FOR
SELECT PNAME,COLOR,PRICE FROM STOCK
  WHERE PNAME=N'SKIRT'
```

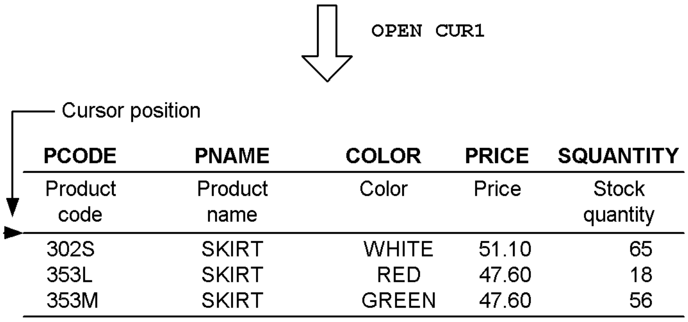
2. Cursor opening

When a cursor is opened, retrieval results can be extracted in accordance with the defined conditions. The retrieval results are stored in a table format in the system and remain valid until the cursor is closed.

The following specification opens the cursor:
`OPEN CUR1`

As soon as a cursor is opened, it is positioned at the first column above the first row of the retrieval results table. The following table shows a cursor immediately after it has been opened; in this example, the cursor name is `CUR1` and the retrieval condition for the stock table is `SKIRT`.

Figure 2-7: Cursor position immediately following cursor opening



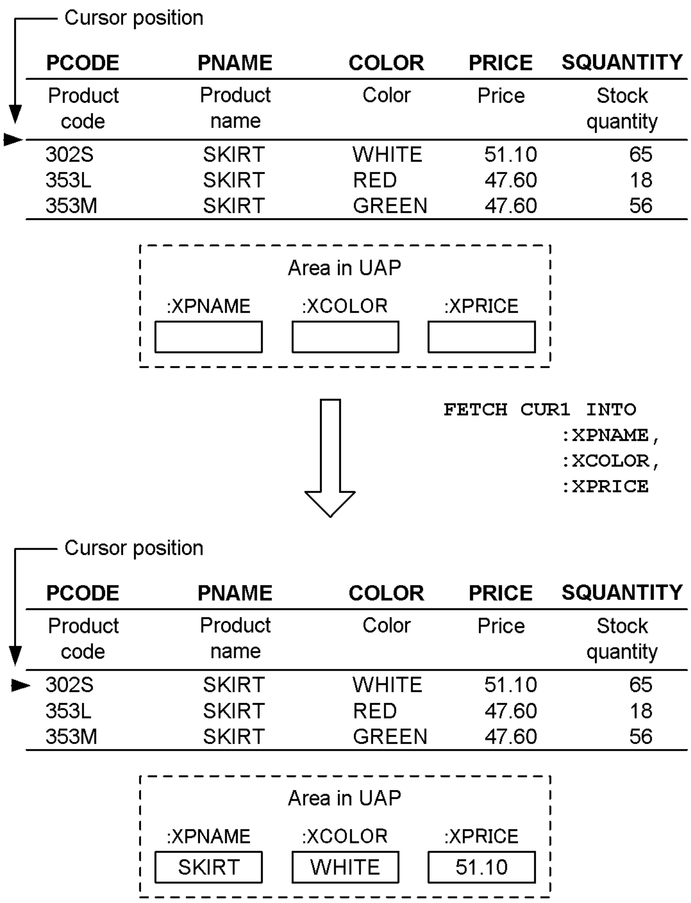
PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

3. Data extraction

   The `FETCH` statement advances the cursor by one row (to the next row). The contents of that row are then stored in a specified area in the UAP.

   The following figure shows the cursor immediately after it has been opened and how the retrieved contents are stored in the UAP.

Figure 2-8: Example of extracting retrieved contents and storing them in the UAP



PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

```
FETCH CUR1 INTO
         :XPNAME,
         :XCOLOR,
         :XPRICE
```

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

:XPNAME	:XCOLOR	:XPRICE
SKIRT	WHITE	51.10

4. Data output

   The data stored in the area in the UAP is output as necessary.

5. Cursor closure

   When processing of retrieved data by the UAP has been completed, close the cursor.

   Once the cursor is closed, the retrieval results table stored in the system is deleted. The following specification closes the cursor:
   `CLOSE CUR1`

2.3.2 Retrieval from multiple tables

The `FROM` clause of the `SELECT` statement is used to retrieve data from two or more tables. The following figure shows, as an example of obtaining a single result from multiple tables, the procedure for creating a table consisting of form numbers and product names for products with fewer than 60 units in stock and fewer than 30 units ordered.

Figure 2-9: Example of retrieval from two tables

STOCK(Stocktable)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Productname	Color	Price	Stockquantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

ORDER(Ordertable)

FNO	CCODE	PCODE	OQUANTITY	...
Form number	Customer code	Product code	Ordered quantity	...
026551	TT002	101M	10	...
026552	TT002	591M	25	
026553	TH001	353M	8	
026554	TK001	411M	6	
026555	TA001	591M	30	...
026556	TT002	202M	10	
026557	TZ001	411M	5	
026558	TZ001	412M	4	
026559	TH001	591M	80	...
026560	TT001	591L	10	

```
SELECT FNO, PNAME FROM STOCK, ORDER
        WHERE STOCK.PCODE=ORDER.PCODE
       AND STOCK.SQUANTITY<60
       AND ORDER.OQUANTITY<30
```

Retriev alresults

FNO	PNAME
Form number	Productname
026553	SKIRT
026554	SWEATER
026557	SWEATER
026558	SWEATER

2.3.3 Retrieval of a table with FIX attribute

When retrieving data from a table with the `FIX` attribute, an entire row can be retrieved as a fixed-length record. In a sense, the entire row is manipulated as a single column. This is called *retrieval on a row basis*; `ROW` is specified in the selection clause of the `SELECT` statement.

Retrieval on a row basis reduces retrieving overhead for each column, so it enhances access performance.

The following figure shows, as an example of retrieval on a row basis, the procedure for using a cursor (`CUR1`) to retrieve the product name `POLO SHIRT` from the stock table and then setting it into an embedded variable (`:XROW`).

Figure 2-10: Example of retrieval on a row basis

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
DECLARE CUR1 CURSOR FOR SELECT
  ROW FROM STOCK
  WHERE PNAME= 'POLO SHIRT'

OPEN CUR1
```

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	BLUE	36.40	67

:XROW


```
FETCH CUR1 INTO :XROW
```

:XROW

201M	POLO SHIRT	WHITE	36.40	29

2.4 Data updating

The following three methods can be used to update information in a table:

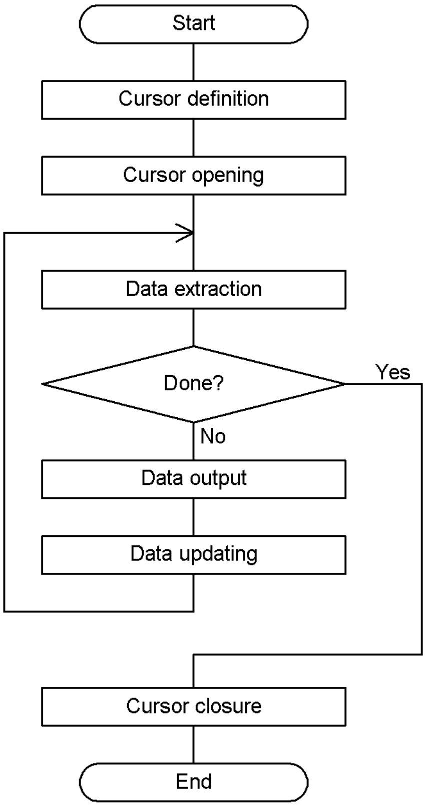
- Updating the row indicated by the cursor
- Updating only those rows that satisfy a condition
- Updating on a row basis (for table with `FIX` attribute only)

If the table is partitioned by key ranges, the values in the column being used as the key cannot be updated.

2.4.1 Updating using a cursor

Multiple retrieved rows are updated by using a cursor to retrieve one row at a time. The following figure shows how to use a cursor to update a table.

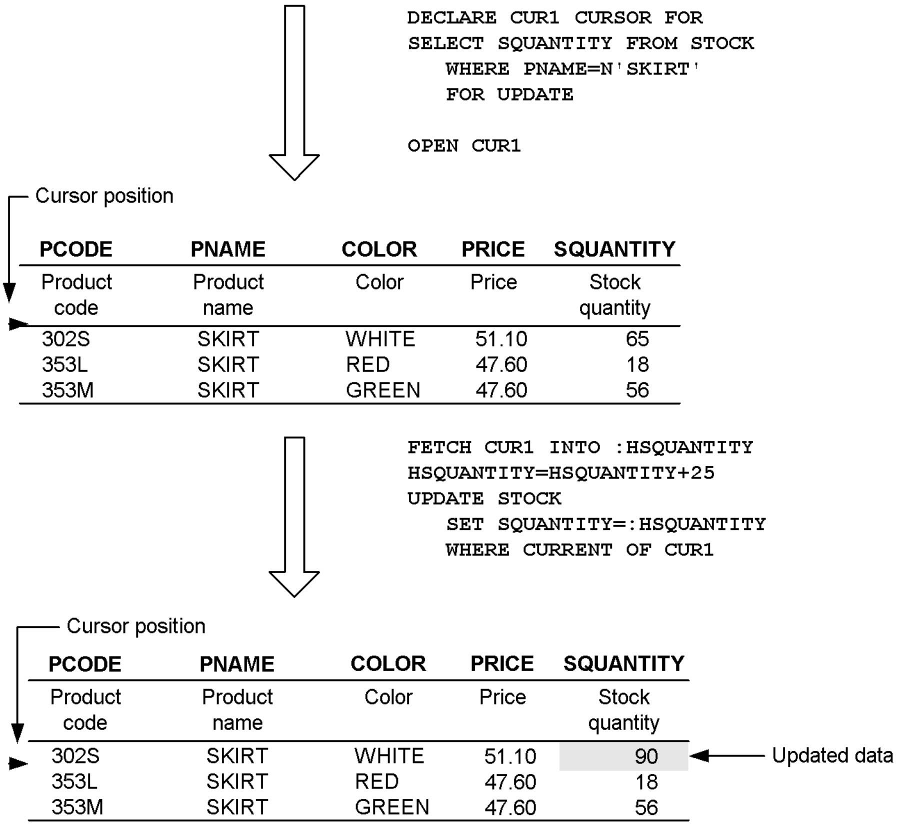
Figure 2-11: Procedure for updating a table



The steps of the processing procedure shown in *Figure 2-11* are basically the same as the steps in *Figure 2-6*, except for data updating.

The following figure shows an example of using a cursor to update a table. It is assumed that the steps up to data fetching have been completed.

Figure 2-12: Example of using a cursor to update a table



```
DECLARE CUR1 CURSOR FOR
SELECT SQUANTITY FROM STOCK
   WHERE PNAME=N'SKIRT'
   FOR UPDATE

OPEN CUR1
```

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

```
FETCH CUR1 INTO :HSQUANTITY
HSQUANTITY=HSQUANTITY+25
UPDATE STOCK
   SET SQUANTITY=:HSQUANTITY
   WHERE CURRENT OF CUR1
```

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	90
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

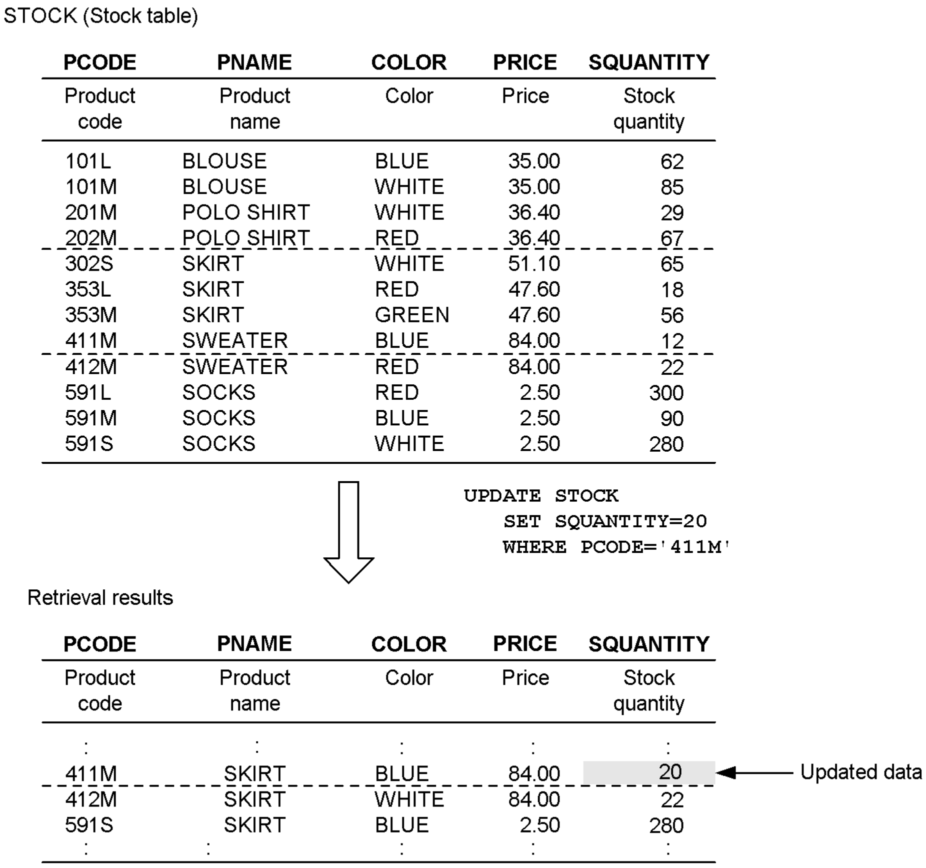
2.4.2 Updating with a condition specified

When a condition is specified for data updating, all rows that satisfy the condition are updated. To update by specifying a condition, the `WHERE` clause must be specified in the `UPDATE` statement.

If the table is partitioned by key ranges, the values in the column being used as the key cannot be updated.

The following figure shows, as an example of updating with a condition specified, the procedure for updating to `20` the quantity of each item whose product code in the stock table is `411M`.

Figure 2-13: Example of updating with condition specified



STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
UPDATE STOCK
   SET SQUANTITY=20
   WHERE PCODE='411M'
```

Retrieval results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
:	:	:	:	:
411M	SKIRT	BLUE	84.00	20
412M	SKIRT	WHITE	84.00	22
591S	SKIRT	BLUE	2.50	280
:	:	:	:	:

2.4.3 Updating a table with the FIX attribute

When a table with the FIX attribute is updated, an entire row can be updated as fixed-length data. To update on a row basis, ROW must be specified in the SET clause of the UPDATE statement.

Updating on a row basis reduces updating overhead for each column, so it enhances access performance.

The following figure shows, as an example of updating a table on a row basis, the procedure for updating from 12 to 20 the quantity of each item in the stock table whose product code is 411M by specifying the new value in an embedded variable (:YROW).

Figure 2-14: Example of updating on a row basis

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

:YROW

411M	SWEATER	BLUE	84.00	20

```
UPDATE STOCK
   SET ROW=:YROW
   WHERE PCODE='411M'
```

Retrieval results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
:	:	:	:	:
411M	SKIRT	BLUE	84.00	20
412M	SKIRT	WHITE	84.00	22
591S	SKIRT	BLUE	2.50	280
:	:	:	:	:

Updated data

2.4.4 Updating a table with repetition columns

The following three methods are provided for updating a table that has repetition columns:

- Updating an existing element (`SET` clause)
- Adding a new element (`ADD` clause)
- Deleting an existing element (`DELETE` clause)

To update a table that has repetition columns, specify the repetition column elements to be updated using the following format: *repetition-column-name* `[{`*subscript*`| *}]`. *subscript* indicates the element position.

This section describes the method for adding a new element.

The figure below shows an example of updating a table that has repetition columns. This example adds in the staff table the element DATABASE to the qualifications of the employee named SMITH, BOB.

Figure 2-15: Example of updating a table with repetition columns

STAFF_TABLE

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 1 NETWORK INFORMATION PROCESSING 2	MALE	STEVE CAROL MARY DAN JANE	FATHER MOTHER WIFE SON DAUGHTER	1 1 1 1 1
SMITH, BOB	INFORMATION PROCESSING 2 ENGLISH CERTIFICATION 2	MALE	EDWARD SUSAN	FATHER WIFE	0 1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

```
UPDATE STAFF_TABLE
  ADD QUALIFICATION[*]=ARRAY[N'DATABASE']
  WHERE NAME=N'SMITH, BOB'
```

Updated results

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 1 NETWORK INFORMATION PROCESSING 2	MALE	STEVE CAROL MARY DAN JANE	FATHER MOTHER WIFE SON DAUGHTER	1 1 1 1 1
SMITH, BOB	INFORMATION PROCESSING 2 ENGLISH CERTIFICATION 2 DATABASE	MALE	EDWARD SUSAN	FATHER WIFE	0 1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

Updated data
(added element)

2.5 Data deletion

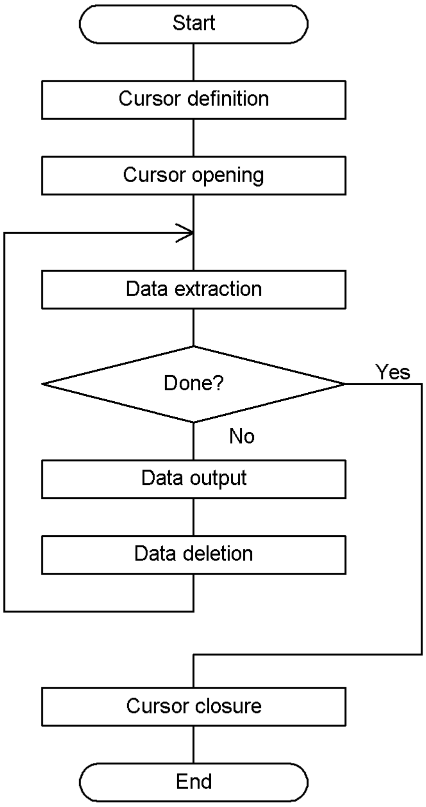
The following three methods are provided for deleting information in a table:

- Deleting the row indicated by the cursor
- Deleting only those rows that satisfy a condition
- Deleting all rows

2.5.1 Deletion using a cursor

To delete rows in a table, a cursor can be used to verify each row's contents and delete the rows one row at a time. The following figure shows the procedure for using a cursor to delete rows in a table.

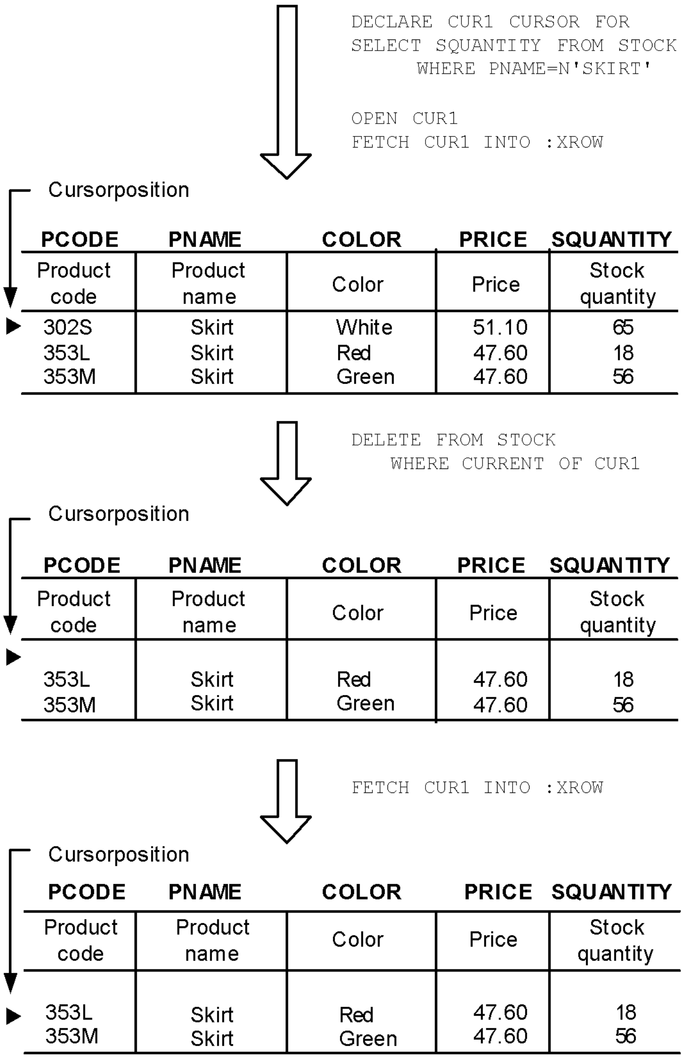
Figure 2-16: Procedure for deleting a table



The steps of the processing procedure shown in *Figure 2-16* are basically the same as the steps in *Figure 2-6*, except for data deletion.

The figure below shows an example of using a cursor to delete data one row at a time. It is assumed that the steps up to data fetching have been completed.

Figure 2-17: Example of using a cursor to delete rows



```
DECLARE CUR1 CURSOR FOR
SELECT SQUANTITY FROM STOCK
     WHERE PNAME=N'SKIRT'

OPEN CUR1
FETCH CUR1 INTO :XROW
```

Cursorposition

	PCODE	PNAME	COLOR	PRICE	SQUANTITY
	Product code	Product name	Color	Price	Stock quantity
▶	302S	Skirt	White	51.10	65
	353L	Skirt	Red	47.60	18
	353M	Skirt	Green	47.60	56

```
DELETE FROM STOCK
   WHERE CURRENT OF CUR1
```

Cursorposition

	PCODE	PNAME	COLOR	PRICE	SQUANTITY
	Product code	Product name	Color	Price	Stock quantity
▶					
	353L	Skirt	Red	47.60	18
	353M	Skirt	Green	47.60	56

```
FETCH CUR1 INTO :XROW
```

Cursorposition

	PCODE	PNAME	COLOR	PRICE	SQUANTITY
	Product code	Product name	Color	Price	Stock quantity
▶	353L	Skirt	Red	47.60	18
	353M	Skirt	Green	47.60	56

2.5.2 Deletion with a condition specified

If a condition is specified for data deletion, all rows that satisfy the condition are deleted. To delete by specifying a condition, the `WHERE` clause must be specified in the `DELETE` statement.

The following figure shows, as an example of deletion with a condition specified, the procedure for deleting from the stock table only the items whose product name is SKIRT.

Figure 2-18: Example of deletion with a condition specified

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
DELETE FROM STOCK
  WHERE PNAME=N'SKIRT'
```

Deletion results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

2.5.3 Deleting all rows in a table

When the target of data deletion is a base table, it is possible with the PURGE TABLE statement to delete all rows in the table in one step. Deleting all rows in a table in one step is more efficient than deleting them by using the DELETE statement with the WHERE clause omitted (without specifying a condition).

The PURGE TABLE statement cannot be executed if the application program is compliant with X/Open in the On-Line Transaction Processing (OLTP) environment.

The following figure shows, as an example of deleting all rows in a table, the procedure for deleting all data from the stock table.

Figure 2-19: Example of deleting all rows in a table

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
PURGE TABLE STOCK
```

Deletion results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity

2.6 Data insertion

Two methods are provided for inserting rows into a table:

- Inserting rows on a column basis
- Inserting rows on a row basis (to a table with `FIX` attribute)

2.6.1 Inserting rows on a column basis

To insert a single row by directly specifying values in each column, use the `INSERT` statement.

The following figure shows, as an example of inserting rows on a column basis, the procedure for inserting the values set in embedded variables (`:ZPCODE` to `:ZSQUANTITY`) into columns in the stock table.

Figure 2-20: Example of row insertion on a column basis

ZPCODE	ZPNAME	ZCOLOR	ZPRICE	ZSQUANTITY
671L	PULLOVER	WHITE	45.00	45

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
:	:	:	:	:
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
INSERT INTO STOCK (PCODE,PNAME,
                   COLOR,PRICE,SQUANTITY)
     VALUES (:ZPCODE,:ZPNAME,:ZCOLOR,
             :ZPRICE,:ZSQUANTITY)
```

Insertion results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
:	:	:	:	:
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280
671L	PULLOVER	WHITE	45.00	45

Inserted row

2.6.2 Inserting rows on a row basis (to a table with the FIX attribute)

When rows are inserted into a table with the `FIX` attribute, an entire row can be inserted as a fixed-length record. To insert a row on a row basis, `ROW` must be specified in the `INSERT` statement.

Rows can be inserted on a row basis only into a base table.

The following figure shows, as an example of inserting rows on a row basis, the procedure for inserting into the stock table the values set in an embedded variable (`:ZROW`).

Figure 2-21: Example of row insertion on a row basis

:ZROW

671L	PULLOVER	WHITE	45.00	45

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
:	:	:	:	:
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
INSERT INTO STOCK (ROW)
   VALUES (:ZROW)
```

Insertion results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
:	:	:	:	:
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280
671L	PULLOVER	WHITE	45.00	45

Inserted row

2.6.3 Inserting rows into a table with repetition columns

When inserting rows into a table that has repetition columns, specify the insertion values for the repetition columns using the following format:
`ARRAY[`*element-value*`[,`*element-value*`]...]`.

The figure below shows an example of inserting rows into a table that has repetition columns. This example inserts a row into the staff table.

Figure 2-22: Example of inserting a row into a table with repetition columns

STAFF_TABLE

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 2 NETWORK INFORMATION PROCESSING 2	MALE	STEVE CAROL MARY DAN JANE	FATHER MOTHER WIFE SON DAUGHTER	1 1 1 1 1
SMITH, BOB	INFORMATION PROCESSING 2 ENGLISH CERTIFICATION 2	MALE	EDWARD SUSAN	FATHER WIFE	0 1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

```
INSERT INTO STAFF_TABLE
    VALUES (N'HUTTON, ERIC',
    ARRAY[N'INFORMATION PROCESSING 1',
    N'INFORMATION PROCESSING 2'],
    N'FEMALE',
    ARRAY[N'SARAH', N'RICHARD'],
    ARRAY[N'HUSBAND', N'SON'],
    ARRAY[1, 1])
```

Insertion result

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 2 NETWORK INFORMATION PROCESSING 2	MALE	STEVE CAROL MARY DAN JANE	FATHER MOTHER WIFE SON DAUGHTER	1 1 1 1 1
SMITH, BOB	INFORMATION PROCESSING 2 ENGLISH CERTIFICATION 2	MALE	EDWARD SUSAN	FATHER WIFE	0 1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			
HUTTON, SARAH	INFORMATION PROCESSING 1 INFORMATION PROCESSING 2	FEMALE	ERIC RICHARD	HUSBAND SON	1 1

Inserted row

2.7 Specific data search

Retrieving specific data with conditions is called a *search*. A search condition is specified to manipulate data in a table on the basis of a condition. A search condition selects the rows to be manipulated; multiple conditions can be combined using logical operators. The following four methods are provided for searching for data in a table:

- Searching for data within a specified range of values
- Searching for a specified character pattern
- Searching for non-NULL data
- Searching for data that satisfies multiple conditions

2.7.1 Searching for data within a specified range of values

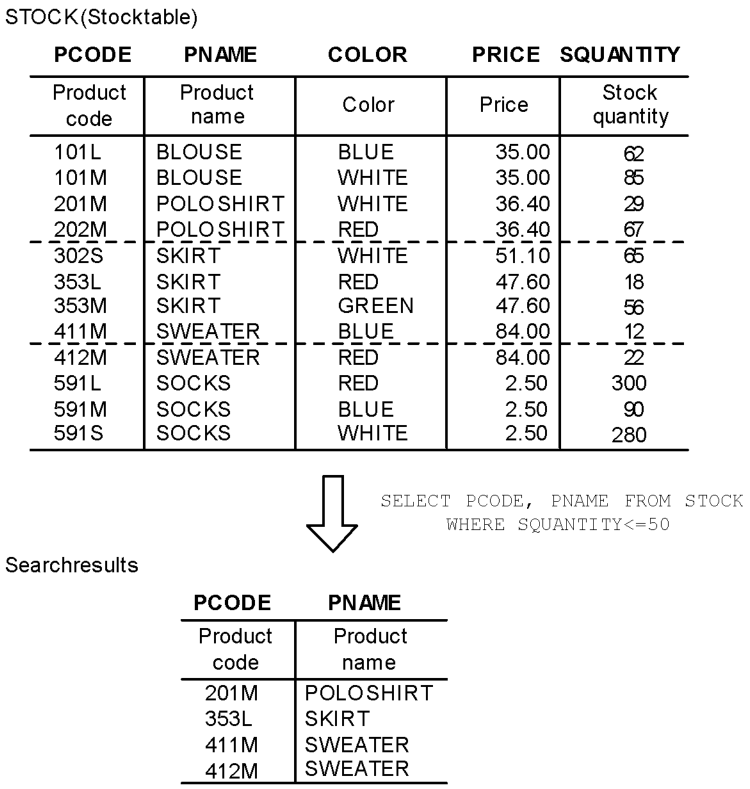
To manipulate rows by specifying a range of values, a `comparison` predicate, a `BETWEEN` predicate, or an `IN` predicate is used to set a condition.

(1) Comparison predicate

A `comparison` predicate is used to specify an equivalence or size comparison as the search condition.

The following figure shows, as an example of a data search using a comparison predicate, the procedure for searching the stock table for the product codes and product names of products with 50 or fewer units in stock.

Figure 2-23: Data search example using a comparison predicate



STOCK(Stocktable)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE, PNAME FROM STOCK
     WHERE SQUANTITY<=50
```

Searchresults

PCODE	PNAME
Product code	Product name
201M	POLOSHIRT
353L	SKIRT
411M	SWEATER
412M	SWEATER

(2) BETWEEN predicate

A BETWEEN predicate extracts only the data within a specified range of values.

The following figure shows, as an example of a data search using a BETWEEN predicate, the procedure for searching the stock table for the product codes and product names of products with 200 to 300 units in stock.

Figure 2-24: Data search example using a BETWEEN predicate

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE,PNAME FROM STOCK
   WHERE SQUANTITY BETWEEN 200 AND 300
```

Search results

PCODE	PNAME
Product code	Product name
591L	SOCKS
591S	SOCKS

(3) IN predicate

An IN predicate extracts only those items with data that matches specified multiple values.

The following figure shows, as an example of a data search using an IN predicate, the procedure for searching the stock table for the product codes and product names of products whose unit price is either 36.40 or 47.60.

Figure 2-25: Data search example using an IN predicate

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE,PNAME FROM STOCK
   WHERE PRICE IN(36.40,47.60)
```

Search results

PCODE	PNAME
Product code	Product name
201M	POLO SHIRT
202M	POLO SHIRT
353L	SKIRT
353M	SKIRT

2.7.2 Searching for a specific character pattern

The LIKE predicate manipulates rows that have a specified character pattern in their column.

The following figure shows, as an example of a data search using a LIKE predicate, the procedure for searching the order table for the form numbers, product codes, and quantities ordered for those customer codes with T as the second character.

Figure 2-26: Data search example using a LIKE predicate

ORDER (Order table)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received date	Order received time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT FNO, PCODE, OQUANTITY
    FROM ORDER
    WHERE CCODE LIKE'_T%'
```

Search results

FNO	PCODE	OQUANTITY
Form number	Product code	Ordered quantity
026551	101M	10
026552	591M	25
026556	202M	10
026560	591L	10

2.7.3 Searching for non-NULL data

The NULL predicate combined with NOT manipulates rows that do not contain any null values in their table columns.

When NOT is not combined with the NULL predicate, the rows containing the null value become the target of manipulation.

The following figure shows, as an example of a data search using the NULL predicate with NOT specified, the procedure for searching the order table for the form numbers, product codes, and quantities ordered for all customer codes that have been set (non-null values).

Figure 2-27: Data search example using a NULL predicate with NOT

ORDER(Ordertable)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Orderreceiv ed date	Orderreceiv ed time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT FNO, PCODE, OQUANTITY FROM ORDER
     WHERE CCODE IS NOT NULL
```

Searchresults

FNO	PCODE	OQUANTITY
Form number	Product code	Ordered quantity
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10
026557	411M	5
026558	412M	4
026559	591M	80
026560	591L	10

2.7.4 Searching for data that satisfies multiple conditions

The logical operators AND, OR, and NOT manipulate rows containing data that satisfies multiple conditions.

The following figure shows, as an example of a data search that satisfies multiple conditions, the procedure for searching the stock table for the product codes and quantities ordered for products whose name is either BLOUSE or POLO SHIRT, and that have 50 or more units in stock.

Figure 2-28: Data search example involving multiple conditions

STOCK(Stocktable)

PCODE Product code	PNAME Product name	COLOR Color	PRICE Price	SQUANTITY Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE, SQUANTITY FROM STOCK
     WHERE (PNAME=N'BLOUSE'
OR PNAME=N'POLO SHIRT')
AND SQUANTITY>=50
```

Searchresults

PCODE Product code	SQUANTITY Stock quantity
101L	62
101M	85
202M	67

2.7.5 Searching for data using a Boolean predicate

If the result of a function defined by an abstract data type or the result of a user-defined function is a Boolean value (TRUE, FALSE, or UNKNOWN), use a Boolean predicate for the true/false decision. For details about data retrieval strings that use a Boolean predicate, see *2.12 Manipulating data in a table with abstract data types*.

2.7.6 Searching for data using a structured repetition predicate

When searching for data by specifying conditions for multiple repetition columns in a table that has repetition columns, use a structured repetition predicate.

The following figure shows, as an example of a data search using a structured repetition predicate, the procedure for searching the staff table for those employees who list a father as a family member.

Figure 2-29: Data search example using a structured repetition predicate

STAFF_TABLE

NAME	QUALIFICATION	SEX	FAMILY	RELATIONSHIP	SUPPORT
BROWN, BILL	INFORMATION PROCESSING 1	MALE	STEVE	FATHER	1
	NETWORK		CAROL	MOTHER	1
	INFORMATION PROCESSING 2		MARY	WIFE	1
			DAN	SON	1
			JANE	DAUGHTER	1
SMITH, BOB	INFORMATION PROCESSING 2	MALE	EDWARD	FATHER	0
	ENGLISH CERTIFICATION 2		SUSAN	WIFE	1
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

```
SELECT NAME FROM STAFF_TABLE
    WHERE ARRAY (RELATIONSHIP, SUPPORT)
        [ANY](RELATIONSHIP=N'FATHER' AND
         SUPPORT=1)
```

Search results

NAME
BROWN, BILL

Note: A multicolumn index that consists of RELATIONSHIP and SUPPORT must be defined in the RELATIONSHIP and SUPPORT columns.

2.7.7 Searching for data using a subquery

A query can be represented structurally by specifying values of the query results in a search condition. A subquery allows an easy access to complex queries in a database.

The following figure shows, as an example of a data search using a subquery, the procedure for searching the stock table for the product codes of products whose price equals or exceeds the average price.

Figure 2-30: Data search example using a subquery

STOCK (Stock table)

PCODE Product code	PNAME Product name	COLOR Color	PRICE Price	SQUANTITY Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE FROM STOCK
   WHERE PRICE>=
      (SELECT AVG(PRICE) FROM STOCK)
```

Subquery results

```
(SELECT AVG(PRICE)
            FROM STOCK)
```

Average AVG(PRICE)
38.71

Search results

PCODE Product code
302S
353L
353M
411M
412M

(1) Subquery using a quantified predicate

A `quantified` predicate can be used to determine whether or not the results of a subquery satisfy a specified set of comparison conditions and to further narrow the retrieval results.

The following figure shows, as an example of a subquery using a qualified predicate, the procedure for retrieving from the stock table the product codes and names of all products whose quantity in stock is greater than the quantity in stock for any BLOUSE (regardless of the product code).

Figure 2-31: Data search example using a subquery and a quantified predicate

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE,PNAME  FROM STOCK
   WHERE SQUANTITY>ALL
      (SELECT SQUANTITY FROM STOCK
            WHERE PNAME=N'BLOUSE')
```

Subquery results

```
(SELECT SQUANTITY
          FROM STOCK
      WHERE PNAME=
             N'BLOUSE')
```

BLOUSE IN STOCK
85
62

Search results

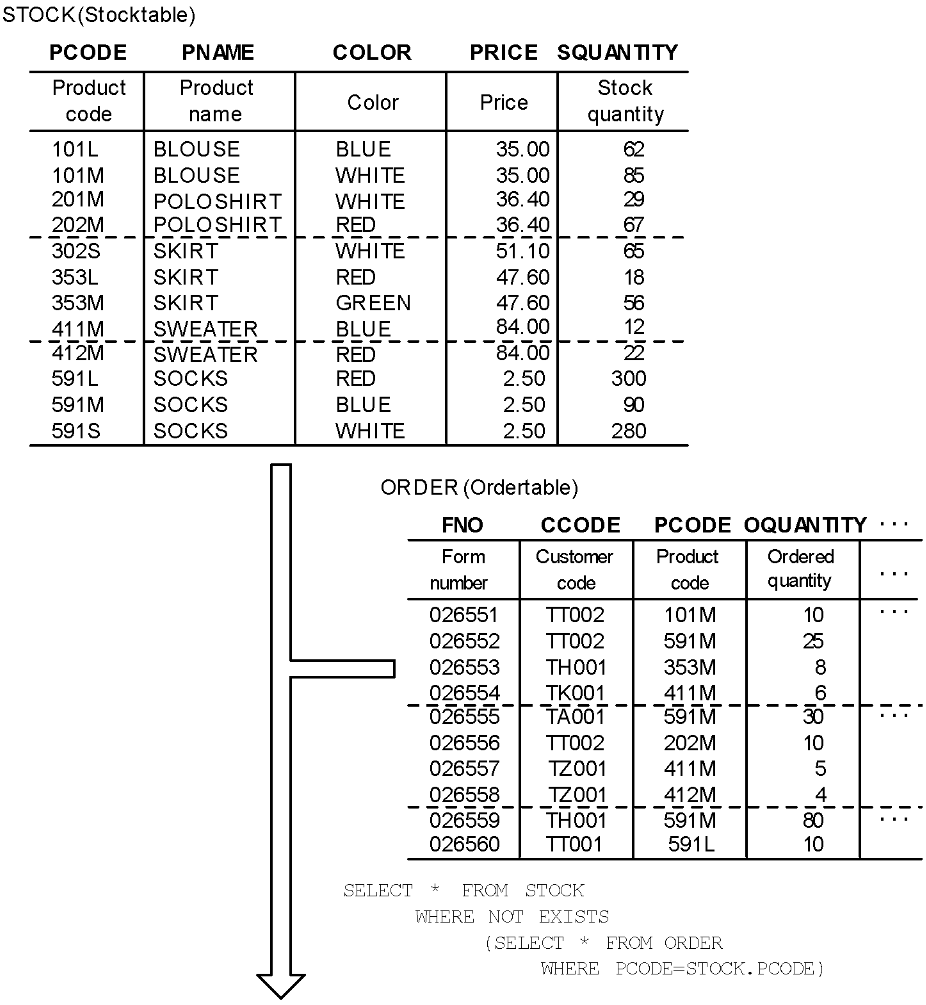
PCODE	PNAME
Product code	Product name
591L	SOCKS
591M	SOCKS
591S	SOCKS

(2) Subquery using the EXISTS predicate

The `EXISTS` predicate is used to test whether or not the results of a subquery are an empty set.

The following figure shows, as an example of a subquery using the `EXISTS` predicate, the procedure for retrieving from the stock and order tables the names of all products for which no orders have been received.

Figure 2-32: Example of a subquery using the EXISTS predicate



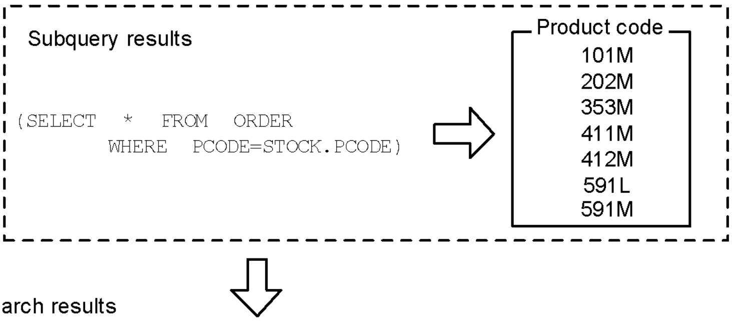
STOCK(Stocktable)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

ORDER (Ordertable)

FNO	CCODE	PCODE	OQUANTITY	. . .
Form number	Customer code	Product code	Ordered quantity	. . .
026551	TT002	101M	10	. . .
026552	TT002	591M	25	
026553	TH001	353M	8	
026554	TK001	411M	6	
026555	TA001	591M	30	. . .
026556	TT002	202M	10	
026557	TZ001	411M	5	
026558	TZ001	412M	4	
026559	TH001	591M	80	. . .
026560	TT001	591L	10	

```
SELECT * FROM STOCK
     WHERE NOT EXISTS
          (SELECT * FROM ORDER
              WHERE PCODE=STOCK.PCODE)
```



Subquery results

```
(SELECT  *  FROM  ORDER
       WHERE  PCODE=STOCK.PCODE)
```

Search results

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
201M	POLOSHIRT	WHITE	36.40	29
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
591S	SOCKS	WHITE	2.50	280

2.8 Data operations

It is possible to search for numeric values, dates, and times in table columns and to extract the results of operations on such values.

The following types of operations can be performed on data in a table:

- Four types of arithmetic operations on numeric data
- Operations on date and time data

2.8.1 Arithmetic operations on numeric data

Four types of arithmetic operations can be performed on numeric values in specified columns, and the results of such operations can be extracted.

The four types of arithmetic operations are addition, subtraction, multiplication, and division.

The following figure shows, as an example of performing arithmetic operations on numeric data, the procedure for calculating projected sales revenue from the unit prices and stock quantities of the products named `SOCKS`, and then retrieving the product code and calculation result for each product (in units of $1 million).

Figure 2-33: Example of numeric data operations

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	8.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
SELECT PCODE,PRICE*SQUANTITY/100000,
          N'Million dollars' FROM STOCK
   WHERE MODEL=N'SOCKS'
```

Operation results

PCODE	PRICE*SQUANTITY/100000
Product code	Projected revenue
591L	7.50 Million dollars
591M	2.25 Million dollars
591S	7.00 Million dollars

2.8.2 Date and time data operations

Operations can be performed on date and time data in a table, and retrieval results based on a specific period of time can be extracted.

Scalar functions are used to operate date or time data. Date operations are used for date data, and time operations are used for time data.

The following figure shows, as an example of performing a time operation on data, the procedure for extracting from the sales order table the form numbers, product codes, and quantities ordered for all orders received before noon (12:00:00).

Figure 2-34: Example of time data operation

ORDER (Ordertable)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Orderreceiv ed date	Orderreceiv ed time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT  FNO,  PCODE,  OQUANTITY
              FROM ORDER
   WHERE  OTIME<TIME('12:00:00')
```

Operationresults

FNO	PCODE	OQUANTITY
Form number	Product code	Ordered quantity
026551	101M	10
026552	591M	25
026553	353M	8
026554	411M	6
026555	591M	30
026556	202M	10

2.9 Data processing

Data to be extracted from a table can be processed in various ways, such as by grouping or by sorting in ascending or descending order. HiRDB provides three types of data processing:

- Grouping data
- Sorting in ascending or descending order
- Eliminating duplicated data

2.9.1 Data grouping

If a value is repeated in a specified column, all the items with that value can be grouped as a single item in the retrieval results. The `GROUP BY` clause performs this grouping. The `AVG`, `SUM`, `MAX`, `MIN`, and `COUNT` set functions, respectively, can be used to obtain the average value, total value, maximum value, minimum value, and rows count of each group.

The following figure shows, as an example of data grouping, the procedure for grouping sets of product codes in the `ORDER1` table and then extracting the total quantity ordered for each group.

Figure 2-35: Data grouping example

ORDER1 (Order table)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received date	Order received time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT PCODE, SUM(OQUANTITY)
     FROM ORDER1
     GROUP BY PCODE
```

Grouping results

PCODE	SUM (OQUANTITY)
Product code	Sum of ordered quantities
101M	10
591M	135
353M	8
411M	11
202M	10
412M	4
591L	10

2.9.2 Data sorting

The data in a specified column of a table can be sorted in ascending or descending order of the values.

The following figure shows, as an example of data sorting, the procedure for retrieving form numbers, product codes, and quantities ordered from the stock table and then sorting the retrieved items in ascending order of the product codes.

Figure 2-36: Data sorting example

ORDER (Order table)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received date	Order received time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT FNO, PCODE, OQUANTITY
     FROM ORDER
     ORDER BY PCODE
```

Search results

FNO	PCODE	OQUANTITY
Form number	Product code	Ordered quantity
026551	101M	10
026556	202M	10
026553	353M	8
026554	411M	6
026557	411M	5
026558	412M	4
026560	591L	10
026552	591M	25
026555	591M	30
026559	591M	80

2.9.3 Duplicated data elimination

When two or more tables are manipulated, duplicated data can be eliminated from retrieval results. UNION or DISTINCT specifies duplicated data elimination.

The following figure shows, as an example of eliminating duplicate data, the procedure for retrieving the product codes of products (with at least 10 units ordered) from two order tables and then eliminating the duplicated data.

Figure 2-37: Duplicated data elimination example

ORDERS1 (Order table 1)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received data	Order received time
026551	TT002	101M	10	1995-06-14	09:23:11
026552	TT002	591M	25	1995-06-14	09:23:11
026553	TH001	353M	8	1995-06-14	10:10:55
026554	TK001	411M	6	1995-06-14	10:15:47
026555	TA001	591M	30	1995-06-14	10:15:47
026556	TT002	202M	10	1995-06-14	11:48:09

ORDERS2 (Order table 2)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Order received data	Order received time
026557	TZ001	411M	5	1995-06-14	13:02:00
026558	TZ001	412M	4	1995-06-14	13:02:00
026559	TH001	591M	80	1995-06-14	14:04:16
026560	TT001	591L	10	1995-06-14	15:31:20

```
SELECT PCODE FROM ORDERS1
   WHERE OQUANTITY>=10
   UNION
SELECT PCODE FROM ORDERS2
   WHERE OQUANTITY>=10
```

Search results

PCODE
Product code
101M
591M
202M
591L

2.10 Outer joining of tables

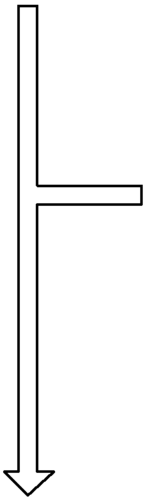
When it is necessary to join an outer table that contains general information and an inner table that contains partial information to obtain information on all rows of the outer table, in addition to the information that can be obtained from normal joining (inner joining), outer joining provides a method of fetching the retrieval results. In outer joining, any inner table columns that do not meet a specified set of joining conditions are assigned null values. One use of outer joining is to join tables that have missing values.

The following figure shows, as an example of an outer join, the procedure for performing an outer join on the stock table and order table to retrieve product codes, produce names, colors, and form numbers of those products whose stock quantity is less than 100.

Figure 2-38: Example of outer joining

STOCK(Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Productname	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280



ORDER (Ordertable)

FNO	CCODE	PCODE	OQUANTITY	· · ·
Form number	Customercode	Product code	Ordered quantity	· · ·
026551	TT002	101M	10	· · ·
026552	TT002	591M	25	· · ·
026553	TH001	353M	8	· · ·
026554	TK001	411M	6	· · ·
026555	TA001	591M	30	· · ·
026556	TT002	202M	10	· · ·
026557	TZ001	411M	5	· · ·
026558	TZ001	412M	4	· · ·
026559	TH001	591M	80	· · ·
026560	TT001	591L	10	· · ·

```
SELECT   STOCK.PCODE,   PNAME,   COLOR,   FNO
           FROM  STOCK  LEFT  OUTER  JOIN  ORDER
                  ON  STOCK.PCODE=ORDER.PCODE
           WHERE  SQUANTITY<100
```

Retriev alresults

PCODE	PNAME	COLOR	FNO
Productcode	Productname	Color	Formnumber
101L	BLOUSE	BLUE	
101M	BLOUSE	WHITE	026551
201M	POLO SHIRT	WHITE	
202M	POLO SHIRT	RED	026556
302S	SKIRT	WHITE	
353L	SKIRT	RED	
353M	SKIRT	GREEN	026553
411M	SWEATER	BLUE	026554
411M	SWEATER	BLUE	026557
412M	SWEATER	RED	026558
591M	SOCKS	BLUE	026552
591M	SOCKS	BLUE	026555
591M	SOCKS	BLUE	026559

Note
The f orm numberf or products with noorders becomes the nullv alue.

The following figure shows an example of an outer join with three or more tables. This example performs an outer join on the stock table, the current month's order table, and the previous month's order table to retrieve the product names and unit prices, as well as the current and previous months' order quantities for all products whose price is $50.00 or more.

Figure 2-39: Example of outer joining with three or more tables

STOCK(Stocktable)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Productcode	Productname	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

ORDER (Ordertable)

. . .	PCODE	OQUANTITY	. . .
. . .	Product code	Ordered quantity	. . .
. . .	101M	10	. . .
. . .	591M	25	. . .
. . .	353M	8	. . .
. . .	411M	6	. . .
. . .	591M	30	. . .
. . .	202M	10	. . .
. . .	411M	5	. . .
. . .	412M	4	. . .
. . .	591M	80	. . .
. . .	591L	10	. . .

PMORDER(Prev iousmonth'sordertable)

. . .	PCODE	PMOQUANTITY	. . .
. . .	Product code	Previousmonth's orderedquantity	. . .
. . .	101M	10	. . .
. . .	591M	20	. . .
. . .	591L	40	. . .
. . .	353L	6	. . .
. . .	591M	30	. . .
. . .	202M	20	. . .
. . .	302S	8	. . .
. . .	412M	4	. . .

```
SELECT A.PCODE, A.PNAME, A.PRICE, B.OQUANTITY, C.PMOQUANTITY
    FROM STOCK A LEFT OUTER JOIN ORDER B ON A.PCODE=B.PCODE
             AND A.PRICE>=50.00
           LEFT OUTER JOIN PMOQUANTITY ON A.PCODE=C.PCODE
             AND A.PRICE>=50.00
```

Retrieval results

PCODE	PNAME	PRICE	OQUANITY	PMOQUANTITY
Product code	Product name	Price	Ordered quantity	Previous month's ordered quantity
101L	BLOUSE	35.00		
101M	BLOUSE	35.00		
201M	POLOSHIRT	36.40		
202M	POLOSHIRT	36.40		
302S	SKIRT	51.10		8
353L	SKIRT	47.60		
353M	SKIRT	47.60		
411M	SWEATER	84.00	6	
411M	SWEATER	84.00	5	
412M	SWEATER	84.00	4	4
591L	SOCKS	2.50		
591M	SOCKS	2.50		
591S	SOCKS	2.50		

Note
If there are no orders or if the price is less than 50.00, the ordered quantity values (OQUANTITY and PMOQUANTITY) become null values.

2.11 Defining and manipulating a view table

Defining a view table derived from other tables to view specific columns and rows allows you to restrict the table data that can be manipulated.

This section describes definition and manipulation of view tables, using as examples the stock table and sales table shown in the following figure.

Figure 2-40: Tables used in examples of manipulating view tables

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

SALES (Sales table)

PCODE	BRANCH	QUANTITY	TOTAL
Product code	Branch	Quantity	Total
101M	NEW YORK	5	175.00
202M	HAWAII	10	364.00
202M	NEW YORK	3	109.20
302S	CALIFORNIA	5	255.50
411M	ALASKA	2	168.00
591M	TEXAS	8	20.00

(1) Defining view tables

Five examples of defining view tables are provided in this section:

- Defining a view table to limit the columns to be searched
- Using search conditions to define a view table
- Defining a read-only view table

- Defining a view table from which duplications are eliminated
- Defining a view table from another view table

(a) Defining a view table to limit the columns to be searched

The following figure shows an example of defining a view table to limit the columns to be searched. In this example, view table `V1` is derived from the stock table so that the columns that can be searched do not include the color column.

Figure 2-41: Example of defining a view table for limiting the columns to be searched

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
CREATE VIEW V1(PCODE,PNAME,PRICE,
                         SQUANTITY)
   AS SELECT PCODE,PNAME,PRICE,
             SQUANTITY FROM STOCK
```

V1 (Viewed table 1)

PCODE	PNAME	PRICE	SQUANTITY
Product code	Product name	Price	Stock quantity
101L	BLOUSE	35.00	62
101M	BLOUSE	35.00	85
201M	POLO SHIRT	36.40	29
202M	POLO SHIRT	36.40	67
302S	SKIRT	51.10	65
353L	SKIRT	47.60	18
353M	SKIRT	47.60	56
411M	SWEATER	84.00	12
412M	SWEATER	84.00	22
591L	SOCKS	2.50	300
591M	SOCKS	2.50	90
591S	SOCKS	2.50	280

(b) Using search conditions to define a view table

The figure below shows an example of using search conditions to define a view table. In this example, a query is used to create view table V2 from the stock and sales tables in order to determine which products at each branch have sold fewer than 10 items.

Figure 2-42: Example of using search conditions to define a view table

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

SALES (Sales table)

PCODE	BRANCH	QUANTITY	TOTAL
Product code	Branch	Quantity	Total
101M	NEW YORK	5	175.00
202M	HAWAII	10	364.00
202M	NEW YORK	3	109.20
302S	CALIFORNIA	5	255.50
411M	ALASKA	2	168.00
591M	TEXAS	8	20.00

```
CREATE VIEW V2
   AS SELECT PNAME,BRANCH,TOTAL
             FROM STOCK,SALES
   WHERE SQUANTITY.PCODE=SALES.PCODE
         AND TOTAL<10
```

V2 (Viewed table 2)

PNAME	BRANCH	TOTAL
Product name	Branch	Total
BLOUSE	NEW YORK	175.00
POLO SHIRT	NEW YORK	109.20
SKIRT	CALIFORNIA	255.50
SOCKS	TEXAS	20.00
SWEATER	ALASKA	168.00

(c) Defining a read-only view table

The figure below shows an example of defining a read-only view table. In this example, a query is used to create read-only view table V3 from the stock table to determine those products whose price is higher than the average price of all products; the retrieved information includes the product codes, product names, prices, and stock quantities.

Figure 2-43: Example of defining a read-only view table

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
CREATE READ ONLY VIEW V3
   AS SELECT PCODE, PNAME, PRICE, SQUANTITY
   FROM STOCK X
   WHERE PRICE>(SELECT AVG(PRICE)
                FROM STOCK Y)
```

V3 (Viewed table 3)

PCODE	PNAME	PRICE	SQUANTITY
Product code	Product name	Price	Stock quantity
302S	SKIRT	51.10	65
353L	SKIRT	47.60	18
353M	SKIRT	47.60	56
411M	SWEATER	84.00	12
412M	SWEATER	84.00	22

(d) Defining a view table from which duplications are eliminated

The figure below shows an example of defining a view table from which duplications are eliminated. In this example, view table V4 is created from the stock table; in view table V4, duplicated product names and prices are eliminated.

Figure 2-44: Example of defining a view table from which duplications are eliminated

STOCK (Stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

```
CREATE VIEW V4(PNAME,PRICE)
   AS SELECT DISTINCT PNAME,PRICE
   FROM STOCK
```

V4 (Viewed table 4)

PNAME	PRICE
Product name	Price
BLOUSE	35.00
POLO SHIRT	36.40
SKIRT	51.10
SKIRT	47.60
SWEATER	84.00
SOCKS	2.50

(e) Defining a view table from another view table

The figure below shows an example of defining a view table from another view table. In this example, a query is used to define view table V5, which is to consist of the rows in view table V1, defined in (a), that contain SKIRT as the product name.

Figure 2-45: Example of defining a view table from another view table

V1 (Viewed table 1)

PCODE Product code	PNAME Product name	PRICE Price	SQUANTITY Stock quantity
101L	BLOUSE	35.00	62
101M	BLOUSE	35.00	85
201M	POLO SHIRT	36.40	29
202M	POLO SHIRT	36.40	67
302S	SKIRT	51.10	65
353L	SKIRT	47.60	18
353M	SKIRT	47.60	56
411M	SWEATER	84.00	12
412M	SWEATER	84.00	22
591L	SOCKS	2.50	300
591M	SOCKS	2.50	90
591S	SOCKS	2.50	280

```
CREATE VIEW V5
   AS SELECT * FROM V1
   WHERE PNAME=N'SKIRT'
```

V5 (Viewed table 5)

PCODE Product code	PNAME Product name	PRICE Price	SQUANTITY Stock quantity
302S	SKIRT	51.10	65
353L	SKIRT	47.60	18
353M	SKIRT	47.60	56

(2) Manipulating a view table

The figure below shows an example of manipulating a view table. In this example, the product name, branch, and sales total of the product with the highest sales total are retrieved from the view table (V2) defined in *(1)(b) Using search conditions to define a view table*. (The view table is specified in the SQL statement that specifies the subquery.)

Figure 2-46: Example of manipulating a view table

V2 (Viewed table 2)

PNAME	BRANCH	TOTAL
Product name	Branch	Total
BLOUSE	NEW YORK	175.00
POLO SHIRT	NEW YORK	109.20
SKIRT	CALIFORNIA	255.50
SOCKS	TEXAS	20.00
SWEATER	ALASKA	168.00

```
SELECT * FROM V2
   WHERE SALES=
        (SELECT MAX(SALES) FROM V2)
```

Retrieval results

PNAME	BRANCH	TOTAL
Product name	Branch	Total
SKIRT	CALIFORNIA	255.50

2.12 Manipulating data in a table with abstract data types

To manipulate data in a table that has abstract data types, use functions or component specifications. The functions include constructor functions (or default constructor functions), which are created automatically when abstract data types are defined, and user-defined functions, which are any functions that the user defines. Component specifications manipulate attributes that make up abstract data types.

Some abstract data types are provided by plug-ins and some are defined by the user. This section discusses the `SGMLTEXT` and `XML` types that are provided by plug-ins.

2.12.1 SGMLTEXT type

This section describes examples that use the HiRDB Text Search Plug-in. The HiRDB Text Search Plug-in provides the abstract data type functions shown in the following table. For details about the abstract data type functions provided by the plug-in, refer to the individual plug-in manuals.

Table 2-1: Descriptions of abstract data type functions provided by the HiRDB Text Search Plug-in

Function name	Description
`SGML TEXT`	SGML text registration
`Contains`	Structure specification retrieval
`contains_with_score, score`	Score retrieval

This section also explains examples that use SGML text to manage an operation manual for medicines. The examples use tables that were defined in the database creation section (for tables that include abstract data types provided by the plug-in) of the *HiRDB Version 9 Installation and Design Guide*.

(1) Retrieving data

(a) Example retrieval with the SGMLTEXT type (1)

Figure 2-47 shows an example of data retrieval with the `SGMLTEXT` type. This example searches for medicines that are indicated for relief of `headaches`. The SQL statement for retrieving the data can be specified as follows:

```
SELECT MEDICINE_ID FROM MEDICINE_MGMT_TABLE
  WHERE contains (OPERATION_MANUAL, 'attached text data
[indications {"headaches"}]')
  IS TRUE
```

This example uses the `contains` abstract data type function to retrieve medicines that include the character string `headaches` in the indications structure section of the

`OPERATION_MANUAL` column.

Figure 2-47: Example of retrieval with the SGMLTEXT type (1)

MEDICINE_MANAGEMENT_TABLE

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning,...</indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children.... Store in the refrigerator .... </warnings></attached text data>
MEDICINE 2	<attached text data><indications>Headaches, toothaches, neuralgia, backaches, ...</indications> <use-dosage>Adults (20 years or older): Take 5 wrappers every 24 hours or more. ...</use-dosage> ⋮ <warnings>Keep out of reach of children .... For aches and pains other than headaches, ...</warnings></attached text data>
⋮	⋮

⇩

Search results

MEDICINE_ID
MEDICINE 2
MEDICINE 7
MEDICINE 8
MEDICINE 16
MEDICINE 19

(b) Example retrieval with the SGMLTEXT type (2)

Figure 2-48 shows another example of retrieval with the `SGMLTEXT` type. This example retrieves the medicine ID and inventory quantity of medicines that are indicated for `food poisoning`. The SQL statement for retrieving the data can be specified as follows:

```
SELECT MEDICINE_MGMT_TABLE.MEDICINE_ID,SQUANTITY
  FROM MEDICINE_MGMT_TABLE LEFT OUTER JOIN STOCK
    ON MEDICINE_MGMT_TABLE.MEDICINE_ID=STOCK.MEDICINE_ID
    WHERE contains (OPERATION_MANUAL, 'attached text data
[indications {"food poisoning"}]')
```

```
    IS TRUE
```

In this example, a medicine management table and a stock table are outer joined and searched. The example uses the `contains` abstract data type function to retrieve medicine IDs that include the character string `food poisoning` in the indications structure section of the `OPERATION_MANUAL` column and find out the stock quantity for those medicine IDs.

Figure 2-48: Example of retrieval with the SGMLTEXT type (2)

MEDICINE_MANAGEMENT_TABLE

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ...</indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator ....</warnings></attached text data>
MEDICINE 2	<attached text data><indications>Headaches, toothaches, neuralgia, backaches, ...</indications> <use-dosage>Adults (20 years or older): Take 5 wrappers every 24 hours or more. ...</use-dosage> ⋮ <warnings>Keep out of reach of children .... For aches and pains other than headaches, ...</warnings></attached text data>
MEDICINE 3	<attached text data><indications>Abdominal pains, food poisoning, nausea, ...</indications> <use-dosage>Adults (20 years or older): Take 5 tablets ....</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator ....</warnings></attached text data>
⋮	⋮

STOCK

MEDICINE_ID	PRICE	SQUANTITY
MEDICINE 1	15.00	150
MEDICINE 2	9.00	60
MEDICINE 4	12.00	200
⋮	⋮	⋮

Search results

MEDICINE_ID	SQUANTITY
MEDICINE 1	150
MEDICINE 3	
⋮	⋮

(2) Updating a table

Figure 2-49 shows an example of updating with the `SGMLTEXT` type. This example updates the operation manual for `MEDICINE 2`. The SQL statement for updating the table can be specified as follows:

```
UPDATE MEDICINE_MGMT_TABLE SET OPERATION_MANUAL =
SGMLTEXT(:sgml AS BLOB(1M))
WHERE MEDICINE_ID = 'MEDICINE 2'
```

This example uses the `SGMLTEXT` abstract data type function to update the operation manual data for `MEDICINE 2`.

The `sgml` BLOB-type embedded variable must be defined beforehand in front of the `UPDATE` statement:

```
EXEC SQL BEGIN DECLARE SECTION;                                  1
       SQL TYPE IS BLOB(300K)sgml;                               1
EXEC SQL END DECLARE SECTION;                                    1
strcpy (sgml. sgml_data,char_ptr_pointing_to_a_sgml_text);         2
sgml.sgml_length=strlen(char_ptr_pointing_to_a_sgml_text);       3
```

1. Define the `sgml` BLOB-type embedded variable.
2. Store the new update data in the `sgml` embedded variable.
3. Set the `sgml_length` attribute value for the BLOB data that was created to the length of the stored data.

Figure 2-49: Example of updating with the SGMLTEXT type

MEDICINE_MGMT_TABLE

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ... </indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator .... </warnings></attached text data>
MEDICINE 2	<attached text data><indications>Headaches, toothaches, neuralgia, backaches, ...</indications> <use-dosage>Adults (20 years or older): Take 5 wrappers every 24 hours or more. ...</use-dosage> ⋮ <warnings>Keep out of reach of children .... For aches and pains other than headaches, ...</warnings></attached text data>
⋮	⋮

Updated results

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ... </indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator .... </warnings></attached text data>
MEDICINE 2	<attached text data><indications>Stomachaches, heartburn, hangovers, ...</indications> <use-dosage>Take the following dosage five times a day. 20 years or older: 1 wrapper. 11 to 19 years: 1/3 wrapper. ...</use-dosage> ⋮ <warnings>Keep out of reach of children ...</warnings></attached text data>
⋮	⋮

Updated data

(3) Deleting rows

Figure 2-50 shows an example of row deletion with the `SGMLTEXT` type. This example deletes the row for MEDICINE 2. The SQL statement for deleting the row can be specified as follows:

```
DELETE FROM MEDICINE_MGMT_TABLE
  WHERE MEDICINE_ID = 'MEDICINE 2'
```

This examples deletes the row for MEDICINE 2 from the medicine management table.

Figure 2-50: Example of deletion with the SGMLTEXT type

MEDICINE_MGMT_TABLE

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ...</indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator ....</warnings></attached text data>
MEDICINE 2	<attached text data><indications>Headaches, toothaches neuralgia, backaches, ...</indications> <use-dosage>Adults (20 years or older): Take 5 wrappers every 24 hours or more. ...</use-dosage> ⋮ <warnings>Keep out of reach of children .... For aches and pains other than headaches, ...</warnings></attached text data>
⋮	⋮

⇩

Deletion results

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ...</indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator ....</warnings></attached text data>
MEDICINE 3	<attached text data><indications>Colds, cold-like symptoms, ...</indications> <use-dosage>Adults (20 years or older): Take 30 tablets five times a day, preferably within five minutes after eating .... Youths (10 to 19): Take 15 tablets five times a day, preferably within five minutes after eating ....</use-dosage> ⋮ <warnings>Keep away from direct sunlight .... Do no drive after taking this medicine ....</warnings></attached text data>
⋮	⋮

(4) Inserting rows

Figure 2-51 shows an example of row insertion with the SGMLTEXT type. This example inserts a row for MEDICINE 25. The SQL statement for inserting the row can be specified as follows:

```
INSERT INTO MEDICINE_MGMT_TABLE(MEDICINE_ID,OPERATION_MANUAL)
  VALUES(MEDICINE 25,SGMLTEXT(:sgml AS BLOB(1M)))
```

This example uses the SGMLTEXT abstract data type function to add a row for MEDICINE 25 to the medicine management table.

The sgml BLOB-type embedded variable must be defined beforehand in front of the INSERT statement:

```
EXEC SQL BEGIN DECLARE SECTION;                         1
     SQL TYPE IS BLOB(300K) sgml;                       1
EXEC SQL END DECLARE SECTION;                           1
strcpy(sgml.sgml_data,char_ptr_pointing_to_a_sgml_text); 2
sgml.sgml_length=strlen(char_ptr_pointing_to_a_sgml_text);    3
```

1. Define the sgml BLOB-type embedded variable.
2. Store the insertion data in the sgml embedded variable.
3. Set the sgml_length attribute value for the BLOB data that was created to the length of the stored data.

Figure 2-51: Example of insertion with the SGMLTEXT type

MEDICINE_MGMT_TABLE

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ... </indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator .... </warnings></attached text data>
⋮	⋮
MEDICINE 24	<attached text data><indications>Colds, cold-like symptoms, ... </indications> <use-dosage>Adults (20 years or older): Take 30 tablets five times a day, preferably within five minutes after eating .... Youths (10 to 19): Take 15 tablets five times a day, preferably within five minutes after eating ....</use-dosage> ⋮ <warnings>Keep away from direct sunlight .... Do not drive after taking this medicine ....</warnings></attached text data>

⇩

Insertion results

MEDICINE_ID	OPERATION_MANUAL
MEDICINE 1	<attached text data><indications>Diarrhea, food poisoning, ... </indications> <use-dosage>Adults (20 years or older): Take 10 tablets after each meal. Youths (11 to 19): Take 7 tablets after each meal.</use-dosage> ⋮ <warnings>Keep out of reach of children .... Store in the refrigerator .... </warnings></attached text data>
⋮	⋮
MEDICINE 25	<attached text data><indications>Eye fatigue, stiff shoulders, ... </indications> <use-dosage>Adults (20 years or older): Take 10 tablets three times a day, after breakfast, lunch, and dinner.</use-dosage> ⋮ <warnings>Keep out of reach of children ....</warnings></attached text data>

Inserted row

2.12.2 XML type

This section describes examples that use HiRDB XML Extension. For details about the abstract data type functions provided by HiRDB XML Extension, see the manual *HiRDB Version 9 SQL Reference*.

This section explains examples that use XML documents to manage book information.

The examples presented below are based on a table defined for the discussion of database creation in the *HiRDB Version 9 Installation and Design Guide* (table containing abstract data types provided by plug-ins).

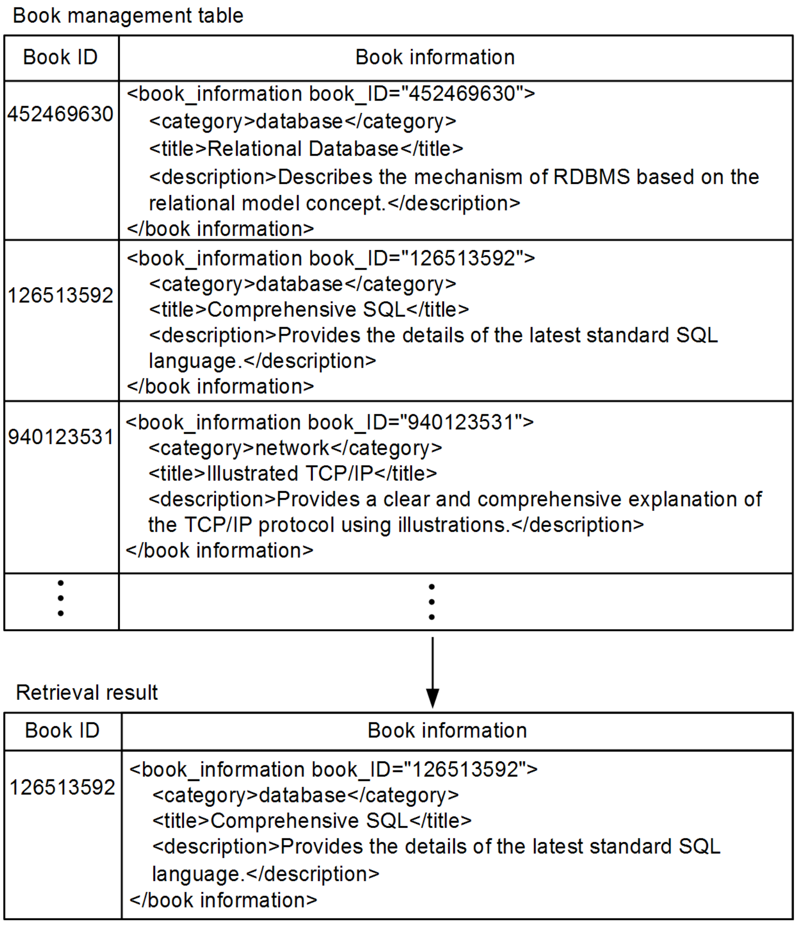
(1) Retrieving data

(a) Example retrieval with the XML type (1)

Figure 2-52 shows an example of data retrieval with the XML type. This example retrieves the book information for book ID `126513592` as `VARCHAR`-type values. The SQL statement for retrieving the data can be specified as follows:

```
SELECT book_ID, XMLSERIALIZE(book_information AS
VARCHAR(32000))
    FROM book_management_table
    WHERE book_ID = 126513592
```

Figure 2-52: Example of retrieval with the XML type (1)



Book management table

Book ID	Book information
452469630	<book_information book_ID="452469630"> <category>database</category> <title>Relational Database</title> <description>Describes the mechanism of RDBMS based on the relational model concept.</description> </book information>
126513592	<book_information book_ID="126513592"> <category>database</category> <title>Comprehensive SQL</title> <description>Provides the details of the latest standard SQL language.</description> </book information>
940123531	<book_information book_ID="940123531"> <category>network</category> <title>Illustrated TCP/IP</title> <description>Provides a clear and comprehensive explanation of the TCP/IP protocol using illustrations.</description> </book information>
⋮	⋮

Retrieval result

Book ID	Book information
126513592	<book_information book_ID="126513592"> <category>database</category> <title>Comprehensive SQL</title> <description>Provides the details of the latest standard SQL language.</description> </book information>

(b) Example of retrieval with the XML type (2)

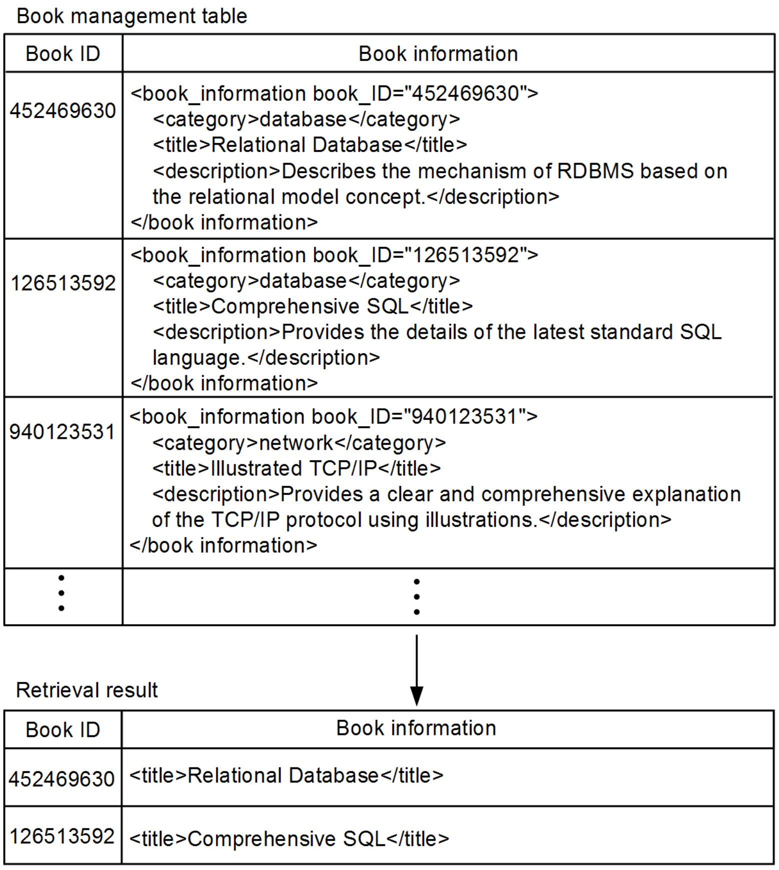
This example retrieves results of evaluation by the `XQuery` expression. *Figure 2-53* shows an example of retrieving the titles of books whose category is `database`. The SQL statement for retrieving the data can be specified as follows:

```
  SELECT book_ID,
         XMLSERIALIZE(
           XMLQUERY('/book_information/title'
                     PASSING BY VALUE book_information
                     RETURNING SEQUENCE EMPTY ON EMPTY)
           AS VARCHAR(32000))
    FROM book_management_table
    WHERE XMLEXISTS('/book_information[category="database"]'
                     PASSING BY VALUE book_information)
```

Explanation:

This example uses the `XMLQUERY` function to retrieve the results of evaluation by the `XQuery` expression. It uses the `XMLEXISTS` predicate to eliminate output of rows in which the result of evaluation by the `XQuery` expression is a null sequence.

Figure 2-53: Example of retrieval with the XML type (2)



Book management table

Book ID	Book information
452469630	<book_information book_ID="452469630"> <category>database</category> <title>Relational Database</title> <description>Describes the mechanism of RDBMS based on the relational model concept.</description> </book information>
126513592	<book_information book_ID="126513592"> <category>database</category> <title>Comprehensive SQL</title> <description>Provides the details of the latest standard SQL language.</description> </book information>
940123531	<book_information book_ID="940123531"> <category>network</category> <title>Illustrated TCP/IP</title> <description>Provides a clear and comprehensive explanation of the TCP/IP protocol using illustrations.</description> </book information>
⋮	⋮

Retrieval result

Book ID	Book information
452469630	<title>Relational Database</title>
126513592	<title>Comprehensive SQL</title>

(c) Example of retrieval with the XML type (3)

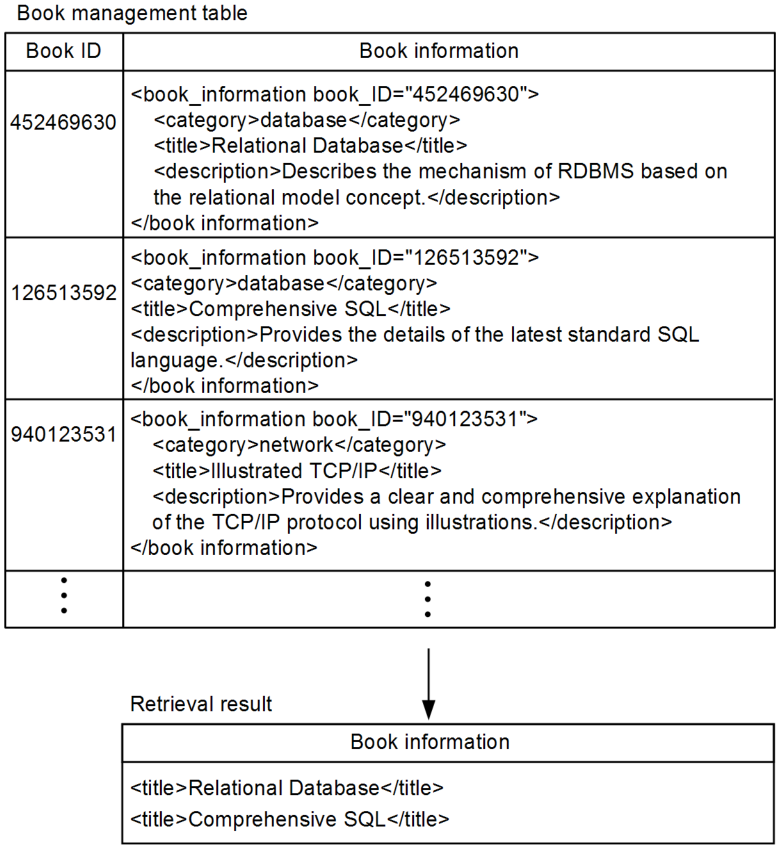
This example outputs values of the XML type as single XML-type values. *Figure 2-54* shows an example of joining the titles of books whose category is `database` and then retrieving the result. The SQL statement for retrieving the data can be specified as follows:

```
SELECT XMLSERIALIZE(
          XMLAGG(
            XMLQUERY('/book_information/title'
               PASSING BY VALUE book_information
               RETURNING SEQUENCE EMPTY ON EMPTY)
  )
  AS VARCHAR(32000))
FROM book_management_table
WHERE XMLEXISTS('/book_information[category="database"]'
                PASSING BY VALUE book_information)
```

Explanation:

You use the XMLAGG set function to output a value of the XML type on each row as a single XML-type value.

Figure 2-54: Example of retrieval with the XML type (3)



Book management table

Book ID	Book information
452469630	<book_information book_ID="452469630"> <category>database&lt;/category> &lt;title&gt;Relational Database&lt;/title&gt; &lt;description&gt;Describes the mechanism of RDBMS based on the relational model concept.&lt;/description&gt; </book information>
126513592	<book_information book_ID="126513592"> <category>database&lt;/category> &lt;title&gt;Comprehensive SQL&lt;/title&gt; &lt;description&gt;Provides the details of the latest standard SQL language.&lt;/description&gt; </book information>
940123531	<book_information book_ID="940123531"> <category&gt;network&lt;/category> &lt;title&gt;Illustrated TCP/IP&lt;/title&gt; &lt;description&gt;Provides a clear and comprehensive explanation of the TCP/IP protocol using illustrations.&lt;/description&gt; </book information>
⋮	⋮

Retrieval result

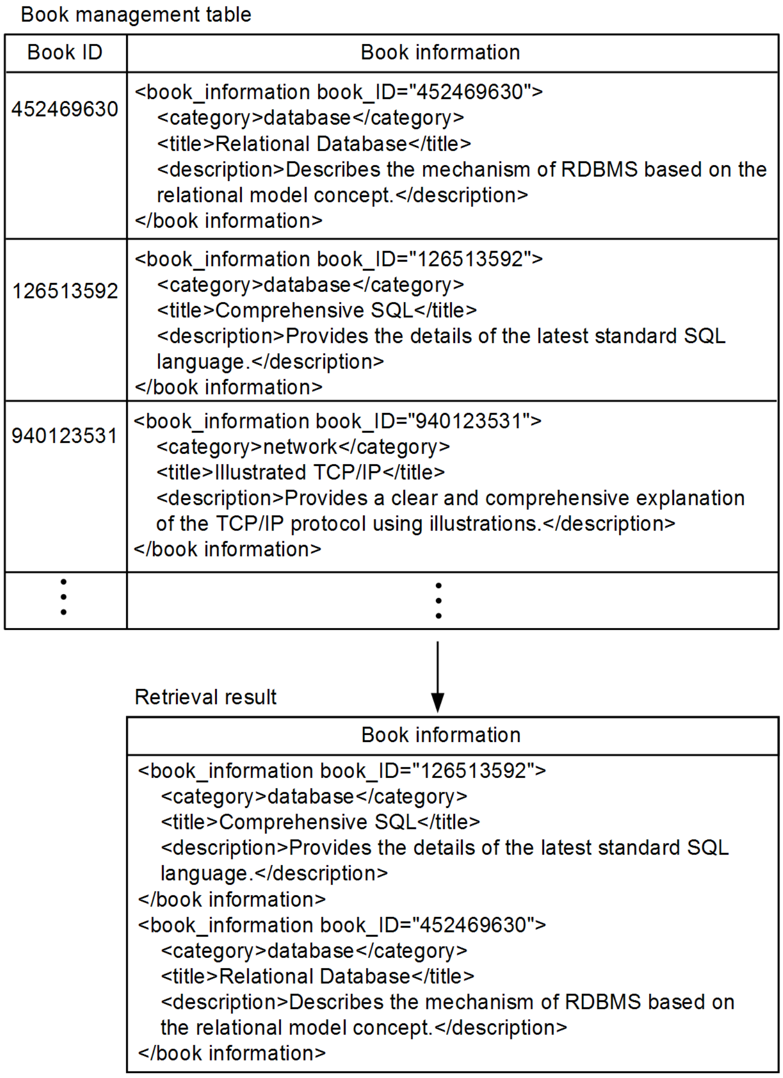
Book information
<title>Relational Database</title> <title>Comprehensive SQL</title>

(d) Example of retrieval with the XML type (4)

This example evaluates the XQuery expression for a value of the XML type as a single XML-type value. *Figure 2-55* shows an example of retrieving all book information that has the title Comprehensive SQL and with the same category. The SQL statement for retrieving the data can be specified as follows:

```
SELECT
  XMLSERIALIZE(
    XMLQUERY(
      '$BOOKS/book_information[category=$BOOKS/
book_information[title="Comprehensive SQL"]/category]'
      PASSING BY VALUE XMLAGG(book_information) AS BOOKS
      RETURNING SEQUENCE EMPTY ON EMPTY))
    AS VARCHAR(32000))
  FROM book_management_table
```

Figure 2-55: Example of retrieval with the XML type (4)



Book management table

Book ID	Book information
452469630	<book_information book_ID="452469630"> <category>database</category> <title>Relational Database</title> <description>Describes the mechanism of RDBMS based on the relational model concept.</description> </book information>
126513592	<book_information book_ID="126513592"> <category>database</category> <title>Comprehensive SQL</title> <description>Provides the details of the latest standard SQL language.</description> </book information>
940123531	<book_information book_ID="940123531"> <category>network</category> <title>Illustrated TCP/IP</title> <description>Provides a clear and comprehensive explanation of the TCP/IP protocol using illustrations.</description> </book information>
⋮	⋮

Retrieval result

Book information
<book_information book_ID="126513592">   <category>database</category>   <title>Comprehensive SQL</title>   <description>Provides the details of the latest standard SQL language.</description> </book information> <book_information book_ID="452469630">   <category>database</category>   <title>Relational Database</title>   <description>Describes the mechanism of RDBMS based on the relational model concept.</description> </book information>

(e) Example of retrieval with the XML type (5)

This example uses a substructure index to retrieve data. The following shows an example definition of an index that uses the `category` element in the `book_information` as the `VARCHAR` type key value.

Example definition of substructure index

```
CREATE INDEX INDX1 ON
book_management_table(book_information)
```

```
  IN (RDAREA02) KEY FROM '/book_information/category' AS
VARCHAR(100)
```

Use of this index enables the SQL statement shown below to reduce the time required for narrowing down the rows. The following example retrieves book information whose category is `network` from `book_management_table`.

Example retrieval using a substructure index

```
SELECT book_ID,
       XMLSERIALIZE(book_information AS VARCHAR(32000))
  FROM book_management_table
  WHERE XMLEXISTS('/book_information[category="network"]'
                PASSING BY VALUE book_information)
```

(f) Example of retrieval with the XML type (6)

This example retrieves data by using an XML-type full-text search index. The following shows an example definition of an XML-type full-text search index for the `book_information` column.

Example definition of XML-type full-text search index

```
CREATE INDEX INDX1
  USING TYPE IXXML ON
book_management_table(book_information)
  IN (LOBAREA01)
```

Use of this index enables the SQL statement shown below to reduce the time required for narrowing down the rows. The following example retrieves book information that contains `RDBMS` in the description from `book_management_table`.

Example retrieval using an XML-type full-text search index

```
SELECT book_ID,
       XMLSERIALIZE(book_information AS VARCHAR(32000))
  FROM book_management_table
  WHERE XMLEXISTS('/book_information/description/
text()[contains( . ,"RDBMS")]'
                 PASSING BY VALUE book_information)
```

2.12.3 User-defined abstract data types

This section describes examples of manipulating tables with user-defined abstract data types. The examples use tables that were defined in the database creation section (for tables that include user-defined abstract data types) of the *HiRDB Version 9 Installation and Design Guide*.

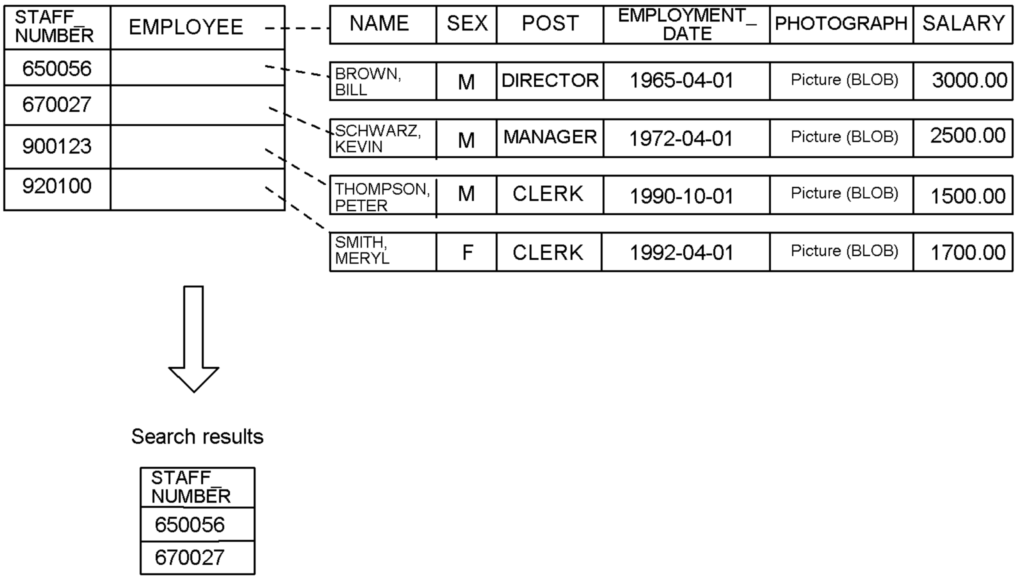
(1) Retrieving data from a table with abstract data types

Figure 2-56 shows an example of retrieval from a table that has abstract data types. The example retrieves staff numbers of employees who have worked for at least 20 years in the company. The SQL statement for retrieving the data can be specified as follows:

```
SELECT STAFF_NUMBER
    FROM STAFF_TABLE
    WHERE YearsOfService(EMPLOYEE)>=20
```

This example uses the user-defined function `YearsOfService` to retrieve staff numbers of employees whose years of service are 20 years or longer. The argument for the user-defined function `YearsOfService` is `EMPLOYEE`.

Figure 2-56: Example of retrieval from a table with abstract data types



STAFF_NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	

NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00

Search results

STAFF_NUMBER
650056
670027

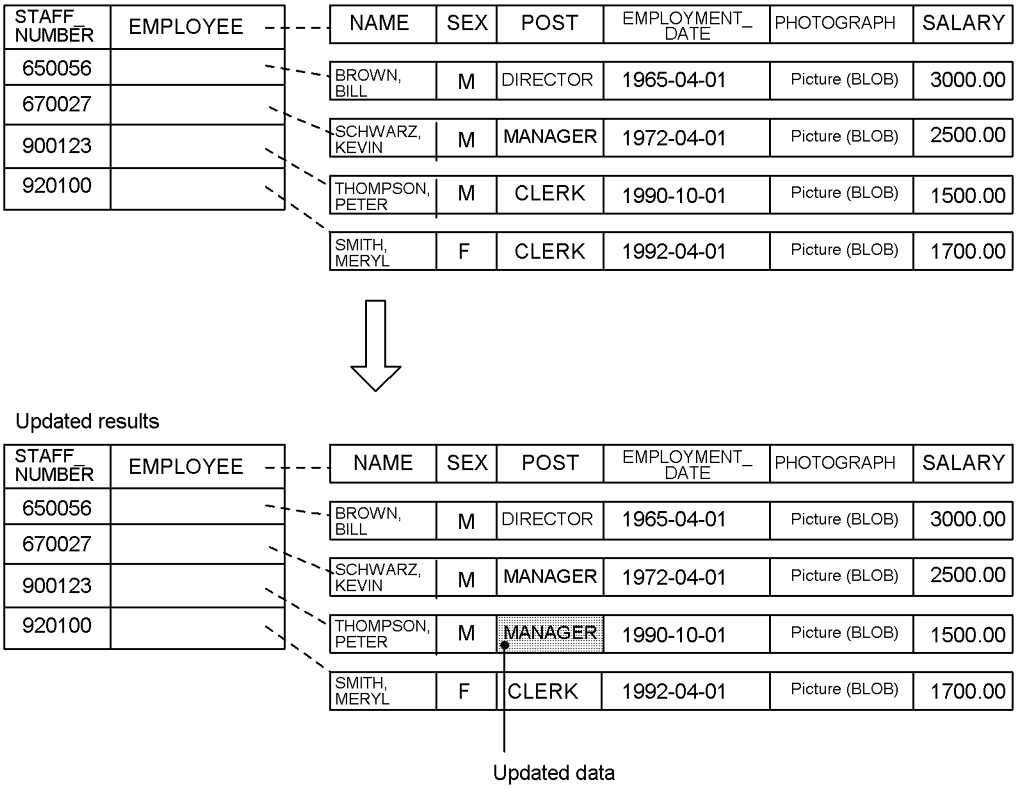
(2) Updating a table with abstract data types

Figure 2-57 shows an example of updating a table that has abstract data types. This example updates the post of the employee with staff number `900123 to MANAGER`. The SQL statement for updating the table can be specified as follows:

```
UPDATE STAFF_TABLE
    SET EMPLOYEE..POST='MANAGER'
    WHERE STAFF_NUMBER='900123'
```

In this example, the `POST` attribute in the `EMPLOYEE` column is updated `to MANAGER` for the employee whose staff number is `900123`. A component specification is used for specifying the attribute of the abstract data type. In this example, `EMPLOYEE..POST` is the component specification.

Figure 2-57: Example of updating a table with abstract data types



STAFF_ NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	

NAME	SEX	POST	EMPLOYMENT_ DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00

Updated results

STAFF_ NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	

NAME	SEX	POST	EMPLOYMENT_ DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	MANAGER	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00

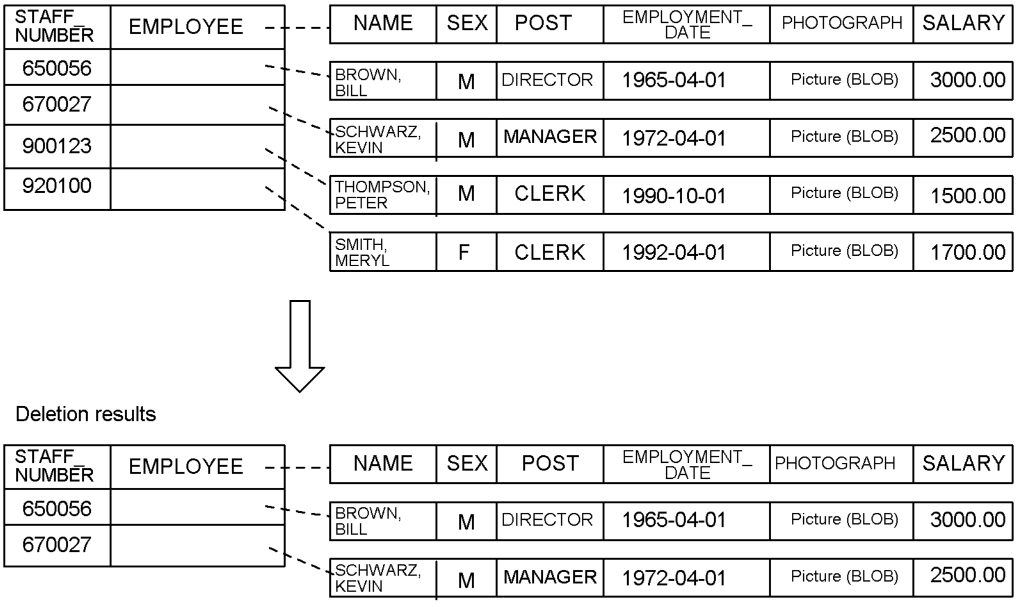
(3) Deleting rows from a table with abstract data types

Figure 2-58 shows an example of row deletion from a table that has abstract data types. The example deletes the rows for employees whose POST is CLERK. The SQL statement for deleting the rows can be specified as follows:

```
DELETE FROM STAFF_TABLE
   WHERE EMPLOYEE..POST='CLERK'
```

This example deletes the rows of employees whose POST attribute in the EMPLOYEE column is CLERK. A component specification is used to specify the abstract data type attribute. In this example, *the* component specification is EMPLOYEE..POST.

Figure 2-58: Example of deleting rows from a table with abstract data types



STAFF_NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	

NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00

Deletion results

STAFF_NUMBER	EMPLOYEE
650056	
670027	

NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00

(4) Inserting rows into a table with abstract data types

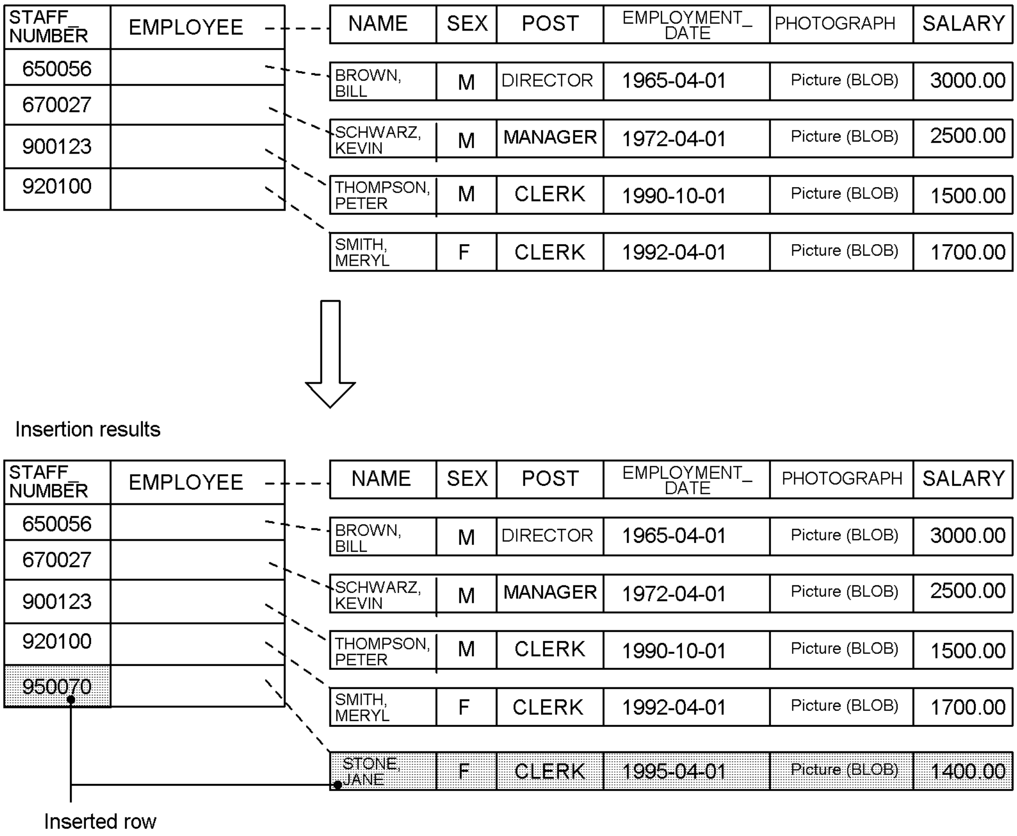
Figure 2-59 shows an example of row insertion into a table that has abstract data types. This example inserts a row into a staff table. The SQL statement for inserting the row can be specified as follows:

```
INSERT INTO STAFF_TABLE
    VALUES ('950070',t_EMPLOYEE('STONE, JANE,
                                'F'
                                'CLERK'
                                '1995-04-01'
                                :PHOTOGRAPH AS BLOB,
                                1400.00
                                )
            )
```

In this example, the `t_EMPLOYEE` constructor function, which was defined when the abstract data type was defined, is used to insert the row for staff number `950070` into the staff table.

`:PHOTOGRAPH` is a BLOB-type embedded variable in which a photographic image of the employee's face is set.

Figure 2-59: Example of inserting rows into a table with abstract data types



STAFF_NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	

NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00

Insertion results

STAFF_NUMBER	EMPLOYEE
650056	
670027	
900123	
920100	
950070	

NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00
STONE, JANE	F	CLERK	1995-04-01	Picture (BLOB)	1400.00

Chapter

3. UAP Design

This chapter explains basic issues that programmers must consider when designing UAPs.

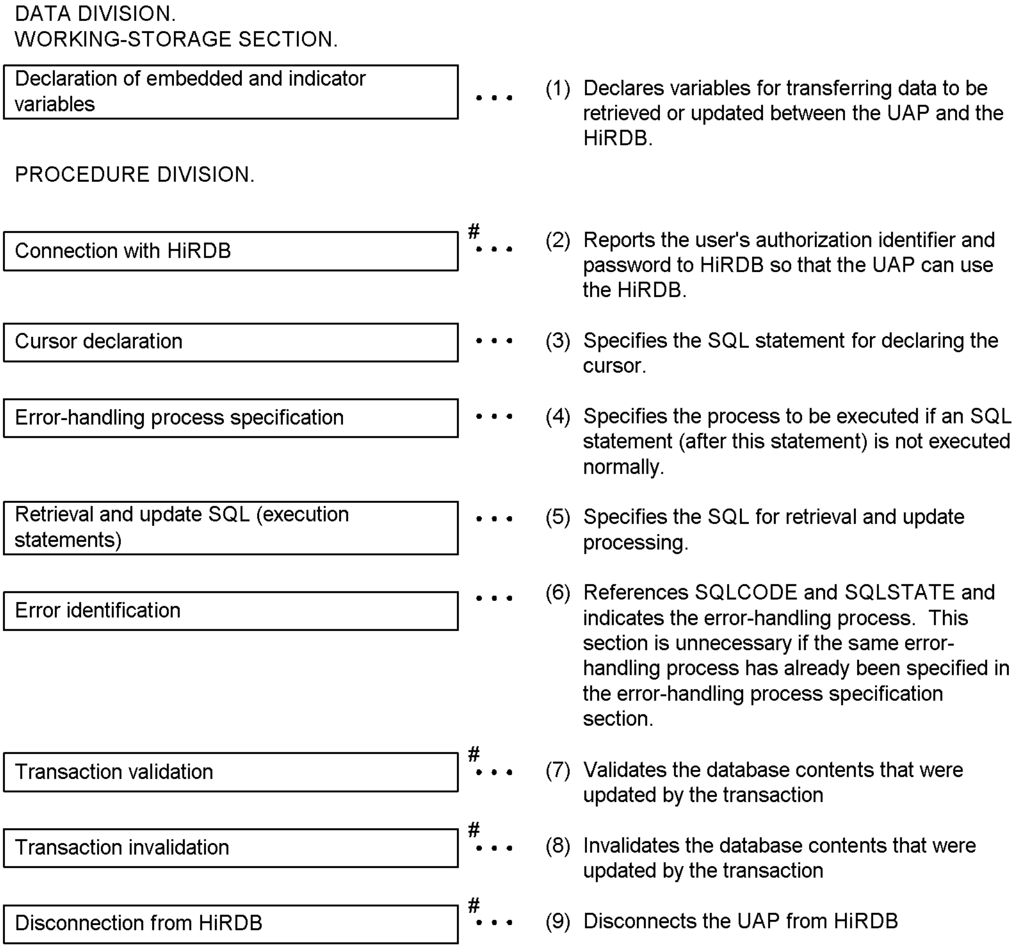
This chapter contains the following sections:

3.1 Basic SQL configuration in a UAP
3.2 Overview of UAPs
3.3 Transaction control
3.4 Locking
3.5 Use of a cursor
3.6 SQL error identification and corrective measures

3.1 Basic SQL configuration in a UAP

The figure below shows the basic SQL configuration in a UAP. This explanation assumes that the UAP is written in COBOL.

Figure 3-1: Basic SQL configuration in a UAP



DATA DIVISION.
WORKING-STORAGE SECTION.

Declaration of embedded and indicator variables • • • (1) Declares variables for transferring data to be retrieved or updated between the UAP and the HiRDB.

PROCEDURE DIVISION.

Connection with HiRDB #• • • (2) Reports the user's authorization identifier and password to HiRDB so that the UAP can use the HiRDB.

Cursor declaration • • • (3) Specifies the SQL statement for declaring the cursor.

Error-handling process specification • • • (4) Specifies the process to be executed if an SQL statement (after this statement) is not executed normally.

Retrieval and update SQL (execution statements) • • • (5) Specifies the SQL for retrieval and update processing.

Error identification • • • (6) References SQLCODE and SQLSTATE and indicates the error-handling process. This section is unnecessary if the same error-handling process has already been specified in the error-handling process specification section.

Transaction validation #• • • (7) Validates the database contents that were updated by the transaction

Transaction invalidation #• • • (8) Invalidates the database contents that were updated by the transaction

Disconnection from HiRDB #• • • (9) Disconnects the UAP from HiRDB

Note

The numbers enclosed in parenthesis correspond to the numbers of the explanation sections described as follows.

#: If necessary, specify an error handling process for this section in *the error handling* *process specification section* or the *error identification* section. However, make sure that the error handling process for transaction invalidation specified in the *error handling process specification* section does not form an endless loop.

(1) Declaration of embedded and indicator variables

The UAP must declare variables for transferring data between SQL and the UAP descriptive language so that the UAP can receive data retrieved by SQL statements and insert UAP data into SQL tables. Use embedded variables for this purpose. If a data item that includes a null value must be transferred, use an indicator variable along with the embedded variable for that item.

An example of declarations for embedded and indicator variables is shown as follows.

For details about how to specify embedded and indicator variables in SQL statements, see *(5) Retrieval and update SQL (execution statements)*.

```
EXEC SQL
     BEGIN DECLARE SECTION  ........................1
END-EXEC.
77 XUSERID  PIC X(7)  .............................2
77 XPSWD    PIC X(7)  .............................2
77 XPCODE   PIC X(4)  .............................2
77 XPNAME   PIC N(8)  .............................2
77 XSTOCK   PIC S9(9)COMP  ........................2
77 ISTOCK   PIC S9(4)COMP  ........................3
EXEC SQL
  END DECLARE SECTION  ............................4
END-EXEC.
```

Explanation:

1. Indicates the beginning of the embedded variable declaration section.
2. Declares an embedded variable; if data is to be transferred between SQL and the UAP, specify embedded variables according to the predetermined rules. For details about the SQL data types and data specifications, see F. *SQL Data Types and Data Descriptions*.
3. Declares an indicator variable for embedded variable (`:xstock`). The indicator variable declaration for a BLOB-type embedded variable is `PIC S9(9) COMP`.
4. Indicates the end of the embedded function declaration section.

If the default value setting facility for null values is used, an embedded variable can accept a default value (0 for numerical data and a space for character data) in place of a null value when the retrieval result is a null value. When this facility is used, indicator variables do not have to be used if the default values and the null value do not have to be discriminated. For details about the default value setting facility for null values, see the *HiRDB Version 9 SQL Reference* manual.

(2) Connection with HiRDB

This section reports the user's authorization identifier and password to HiRDB so that the UAP can use HiRDB. This is called *connection with HiRDB*. The SQL statements for connection with HiRDB are shown as follows:

```
EXEC SQL
     CONNECT  :XUSERID IDENTIFIED BY :XPSWD
END-EXEC.
```

Connects with HiRDB based on the authorization identifier stored in the embedded variable `(:XUSERID)` and the password stored in the embedded variable `(:XPSWD)`.

(3) Cursor declaration

This section uses the `DECLARE CURSOR` statement to declare the cursor that allows the UAP to extract multiple-row retrieval results one row at a time. Use the `DECLARE CURSOR` statement to retrieve, update, and delete data. To open the cursor, use the `OPEN` statement. To extract the retrieval results and move the cursor to the next line, use the `FETCH` statement. To close the cursor, use the `CLOSE` statement.

Embedded and indicator variables can be specified as retrieval condition values in the cursor declaration. If such variables are specified, the UAP passes the values in those variables to HiRDB when the `OPEN` statement for that cursor is executed.

For details about cursors, see *3.5 Use of a cursor*.

The SQL statements for cursor declaration are shown as follows:

```
EXEC SQL
 DECLARE CR1 CURSOR FOR SELECT PCODE, PNAME, STOCK FROM STOCK
END-EXEC.
```

Declares cursor `CR1` for extracting `PCODE`, `PNAME`, and `STOCK` one row at a time from the `STOCK` table.

(4) Error-handling process specification

If a `WHENEVER` statement is specified before an SQL statement, the UAP can automatically determine whether an error occurred.

(a) If an error occurs

```
EXEC SQL
  WHENEVER SQLERROR GO TO error-handling-process
END-EXEC.
```

`WHENEVER SQLERROR`

Declares the process to be executed if an error occurs.

`GO TO` *error-handling-process*

Switches the process to the specified clause or paragraph name (*error-handling-process*) if an error occurs. If an SQL Communications Area is referenced from within this process, return code information can be checked.

(b) If a row to be retrieved is not found

```
EXEC SQL
  WHENEVER NOT FOUND GO TO retrieval-end-process
END-EXEC.
```

`WHENEVER NOT FOUND`

Declares the process to be executed if the row to be retrieved is not found.

`GO TO` *retrieval-end-process*

Switches the process to the specified clause or paragraph name (*retrieval-end-process*), if the row to be retrieved is not found.

(c) Effective range of WHENEVER statement

A `WHENEVER` statement is effective for all SQL statements found between that `WHENEVER` statement and the next `WHENEVER` statement of the same type. For details about the effective range of the `WHENEVER` statement, see the *HiRDB Version 9 SQL Reference* manual.

(5) Retrieval and update SQL (execution statements)

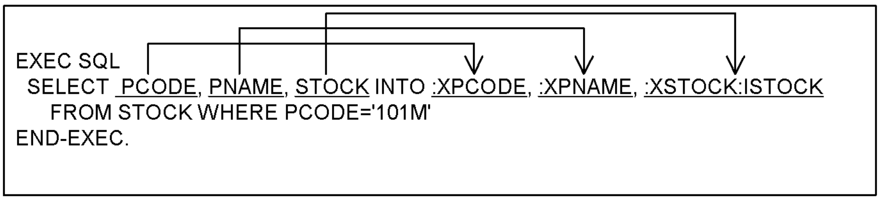
In this section, specify SQL statements for retrieving, inserting, or deleting data. For details about how to specify the individual SQL statements, see *2. Database Operations*.

This section explains how to use embedded and indicator variables.

(a) Specifying embedded and indicator variables in a 1-row SELECT or FETCH statement

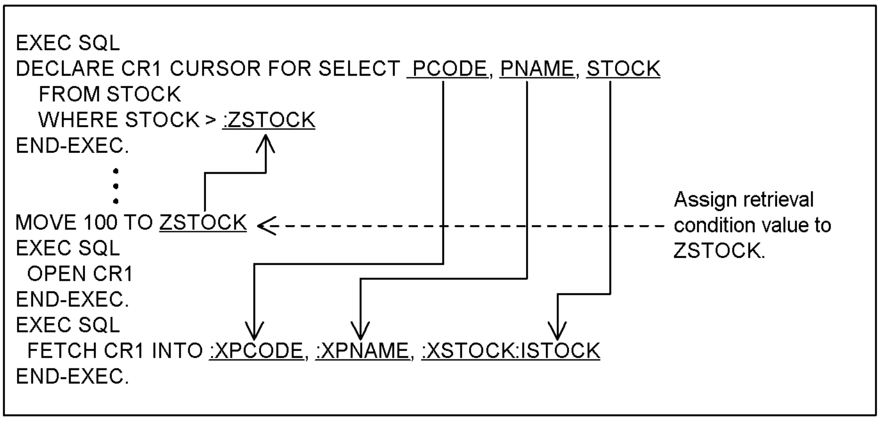
Specify the embedded and indicator variables in the `INTO` clause of a `1-row SELECT` or `FETCH` statement. Add a colon in front of each variable. Specify each indicator variable immediately after its corresponding embedded variable. An example is shown as follows:

<1-row SELECT statement>



```
EXEC SQL
 SELECT PCODE, PNAME, STOCK INTO :XPCODE, :XPNAME, :XSTOCK:ISTOCK
   FROM STOCK WHERE PCODE='101M'
END-EXEC.
```

<FETCH statement>



```
EXEC SQL
DECLARE CR1 CURSOR FOR SELECT PCODE, PNAME, STOCK
  FROM STOCK
  WHERE STOCK > :ZSTOCK
END-EXEC.
         •
         •
         •
MOVE 100 TO ZSTOCK
EXEC SQL
 OPEN CR1
END-EXEC.
EXEC SQL
 FETCH CR1 INTO :XPCODE, :XPNAME, :XSTOCK:ISTOCK
END-EXEC.
```

Assign retrieval condition value to ZSTOCK.

The embedded variables that were specified in the `INTO` clause correspond to the column name sequence specified in the column lineup of the `SELECT` statement. The retrieval results are stored to the embedded variables according to this sequence.

If a retrieval result includes a null value, a negative value is stored in the indicator variable. You can, therefore, check the indicator variable value to determine whether the result is a null value. In this case, the value of the embedded variable is undefined. If the value of an indicator variable is 0, a value other than a null value was received. If the value is positive, character string data other than a null value was received but the right end was truncated, because the area length of the embedded variable was too short.

If an embedded variable is specified in a retrieval condition value, the retrieval condition value can be assigned during SQL execution.

(b) Specifying an embedded or indicator variable in an UPDATE or INSERT statement

Specify the embedded and indicator variables in the `SET` clause of an `UPDATE` statement or the `VALUES` clause of an `INSERT` statement. Add a colon in front of each variable. Specify each indicator variable immediately after its corresponding embedded variable. An example is shown as follows:

`UPDATE` statement

```
    EXEC SQL
    UPDATE STOCK SET STOCK=:XSTOCK:ISTOCK WHERE PCODE=:XPCODE
    END-EXEC.
```

`INSERT` statement

```
    EXEC SQL
    INSERT INTO STOCK VALUES(:XPCODE,:XPNAME,:XCOLOR,:XPRICE,
    :XSTOCK:ISTOCK,:XSTOCK_CAPACITY,:XREQSTOCK)
    END-EXEC.
```

If the `UPDATE` or `INSERT` statement sets a null value in a table, specify a negative value in the indicator value before executing that SQL. No setting value is necessary for the embedded function. When passing a non-null value, set the indicator variable value to 0 or a positive value.

(6) Error identification

If an error occurs during SQL execution, the UAP checks `SQLCODE` and `SQLSTATE` to determine the return codes returned by HiRDB. The UAP uses the return codes to specify which process should then be executed. However, if an error-handling process has already been specified in Section *(4) Error-handling process specification*, the same process does not have to be specified in this section.

Do not execute error identification immediately after a declaration statement, such as `DECLARE CURSOR`. If error identification is executed, the UAP references an incorrect SQLCODE, and HiRDB malfunctions.

For details about error identification, see *3.6.1 Error identification.*

(7) Transaction validation

If update processing was executed in a transaction, this section validates the updated database contents and terminates the transaction normally.

The SQL statements for validating a transaction are shown as follows:

```
EXEC SQL
  COMMIT
END-EXEC.
```

Validates a transaction; to release the UAP from HiRDB after validating the transaction, specify `RELEASE` in the `COMMIT` statement, and execute the statement. If `RELEASE` is specified, the `DISCONNECT` statement does not have to be executed.

(8) Transaction invalidation

This section invalidates the database contents that were updated in a transaction and terminates the transaction. Specify this section to cancel a database update if the update processing in a translation is invalid.

The SQL statements for invalidating a transaction are shown as follows:

```
EXEC SQL
  ROLLBACK
END-EXEC.
```

Invalidates a transaction; to release the UAP from HiRDB after terminating the transaction, specify `RELEASE` in the `ROLLBACK` statement, and execute the statement. If `RELEASE` is specified, the `DISCONNECT` statement does not have to be executed.

(9) Disconnection from HiRDB

This section terminates a transaction normally and releases the UAP from HiRDB. The `DISCONNECT` statement executes the same processing executed by a `COMMIT` statement in which `RELEASE` is specified.

The SQL statements for terminating a transaction normally and releasing the UAP from HiRDB are shown as follows:

```
EXEC SQL
  DISCONNECT
END-EXEC.
```

Terminates a transaction normally and releases the UAP from HiRDB. To cancel a transaction and then release the UAP from HiRDB, execute a `ROLLBACK` statement where `RELEASE` is specified.

Note

> If you terminate a UAP without executing a `DISCONNECT`, `COMMIT` statement (with `RELEASE` specified), or `ROLLBACK` statement (with `RELEASE` specified), the system automatically executes a `ROLLBACK` statement (with `RELEASE` specified), and the transaction that was being executed becomes invalid.

3.2 Overview of UAPs

This section explains the basic issues to be taken into consideration in designing a UAP.

3.2.1 UAP descriptive languages

In this type of UAP, SQL statements are incorporated directly into a source program written in C language (based on ANSI-C) or in COBOL (based on COBOL85).

The following table lists the UAP descriptive languages that can be used in HiRDB.

Table 3-1: UAP descriptive languages

Operating environment	Descriptive languages
HP-UX	• C language (Microsoft Visual C++) • C++ language (Optimizing C++) • COBOL language (COBOL85 and COBOL2002) • OOCOBOL language (OOCOBOL)#
Solaris	• C language • COBOL language# COBOL85, COBOL2002, and COBOL language products of other companies (MicroFocusCOBOL and SUN Japanese COBOL)
AIX	• C language • C++ language • COBOL language (COBOL85 and COBOL2002)
Linux	• C language (gcc) • C++ language (GCC) • COBOL language (COBOL85 and COBOL2002) • OOCOBOL language (OOCOBOL)#
Windows	• C language (Microsoft Visual C++) • C++ language (Optimizing C++) • COBOL language (COBOL85 and COBOL2002) • OOCOBOL language (OOCOBOL)#

#: The multi-connection facility cannot be used.

An embedded-type UAP cannot be compiled or linked directly. Execute the SQL preprocessor and convert the UAP into a post-source program before compiling and linking the UAP. For details about how to preprocess, compile and link UAPs, see *8. Preparation for UAP Execution*.

3.2.2 Interface areas

Interface areas are used for exchanging information between HiRDB and a UAP. The following table lists and describes the types of interface areas and their usage.

Table 3-2: Interface area types and uses

Area type	Use	Language	
		C	COBOL
SQL Communications Areas	For obtaining detailed information on SQL execution results.	R[#1]	R[#1]
SQL Descriptor Areas	• For sending to the system information on input variables that are resolved dynamically during UAP execution. • For receiving information of item to be retrieved from SQL statements that are preprocessed for the dynamic UAP execution. • For specifying column name data areas.	O	O
Column name data areas	For receiving information of item to be retrieved from SQL statements that are preprocessed for the dynamic UAP execution.	O	O
Type name data areas	For receiving user-defined data type names.	O	O
Character set descriptor areas	• For sending to the system the character set names for input variables that are resolved dynamically during UAP execution. • For receiving character set names for items to be retrieved from SQL statements that are preprocessed for dynamic UAP execution.	O	O
Embedded variables	For transferring values (specified in SQL statements embedded in UAP).	O	O
Indicator variables	For transferring values (specified in SQL statements embedded in UAP).	O	O
? parameters	For transferring values from a UAP to the SQL statements that are preprocessed for the dynamic UAP execution.	O	O[#2]

R: Required

O: Optional

#1: These areas need not be declared, because they are expanded within the UAP when the SQL preprocessor is executed. For details about SQL preprocessor execution, see *8.2 Preprocessing*.

#2: An embedded variable and an indicator variable are used instead of a ? parameter.

For details about SQL Communications Areas and SQL Descriptor Areas, see *A. SQL Communications Area* and *B. SQL Descriptor Area*. For details about embedded variables, indicator areas, and ? parameters, see the *HiRDB Version 9 SQL Reference* manual.

3.2.3 Integrity constraints

HiRDB uses the following two integrity constraints to ensure the validity of a database:

- NOT NULL constraint
- Uniqueness constraint

(1) NOT NULL constraint

The NOT NULL constraint prohibits the null value from being set in a specified column.

The NOT NULL operand of the CREATE TABLE statement is used to implement the NOT NULL constraint. Because there must always be a value in every row of a column for which the NOT NULL constraint is specified, a constraint error occurs if an attempt is made to assign the null value in the column. When a constraint error occurs, the database cannot be updated; the null value must never be set in a column for which the NOT NULL constraint is specified.

(2) Uniqueness constraint

When the uniqueness constraint is specified for a column, the value in every row of the column must be unique (no value can be duplicated in the column).

The uniqueness constraint can be specified for the following types of columns:

(a) Column defined as a cluster key

Specified with the UNIQUE operand of the CREATE TABLE statement.

For details about the cluster key specifications, see the *HiRDB Version 9 Installation and Design Guide*.

(b) Column for which an index is defined

Specified with the UNIQUE operand of the CREATE INDEX statement.

For details about the CREATE TABLE and CREATE INDEX specifications, see the manual *HiRDB Version 9 SQL Reference*.

3.2.4 Retrieval methods using SQL statements

An SQL statement used to retrieve a table can be executed either statically or dynamically. The following table shows the classification of UAP retrieval methods using SQL statements.

Table 3-3: Classification of UAP retrieval methods using SQL statements

Retrieval method		SQL statement for specifying query
Embedded UAP	Static SQL	Single-row `SELECT` statement
		Cursor declaration
	Dynamic SQL	Single-row `SELECT` statement
		Dynamic `SELECT` statement

(1) Single-row SELECT statement

The single-row `SELECT` statement extracts only a single-row of retrieval results from a table.

Because a cursor need not be used when the single-row `SELECT` statement is used, you can retrieve the table with only one SQL statement.

The single-row `SELECT` statement is effective when used in the cases listed below. You can also dynamically execute a single-row `SELECT` statement that is constructed during UAP execution.

- You know that the retrieval results will be contained within a single-row
- You use a set function without grouping (using a `GROUP BY` clause)

Even when a single-row is retrieved, using a cursor results in better processing efficiency for updating or deleting the retrieved row. You should consider whether the single-row `SELECT` statement or the cursor will be used.

(2) Cursor declaration

If retrieval results include multiple rows, the UAP cannot receive them all at once. A cursor is used to extract one row at a time. The flow from cursor declaration to retrieval completion is described as follows.

1. Execute `DECLARE CURSOR` to declare a cursor.
2. Execute the `OPEN` statement to open and use the declared cursor.
3. Execute the `FETCH` statement to position the cursor at the first row of the retrieval results. Embedded variables specified by the `INTO` clause of the `FETCH` statement are used to extract the retrieval results.
4. Execute the `FETCH` statement to advance the cursor to the next row (the retrieval results are extracted one row at a time in this manner).
5. Repeat the operation in step 4 until there are no more rows to be retrieved.
6. When the retrieval is completed, execute the `CLOSE` statement to close the cursor.

(3) Dynamic SELECT statement

Use the dynamic `SELECT` statement to extract multiple retrieval results through dynamic SQL execution. To extract retrieval results with the dynamic `SELECT` statement, you must either declare a cursor in advance or allocate a cursor by using the `ALLOCATE CURSOR` statement. Once you declare or allocate a cursor, use the `PREPARE` statement to preprocess the SQL statements that are constructed during UAP execution. You can then perform the same operations as in normal retrieval using a cursor.

3.2.5 Static and dynamic SQLs

SQL statements written directly into the user application program when it is created are called *static SQL statements*; SQL statements that are constructed during UAP execution instead of being written into UAP is called *dynamic SQL statements*.

Because the execution characteristics of static and dynamic SQLs are different, evaluate them carefully before you create a UAP.

(1) Differences during execution

The following table shows the execution characteristics of static and dynamic SQLs.

Table 3-4: Execution characteristics of static and dynamic SQLs

Type	Advantage	Disadvantage
Static SQL	If the UAP is to be executed repeatedly, an executed SQL statement is converted to execute form and can be used again in the shared memory, thus improving processing efficiency.	Because the SQL statements are embedded in the UAP, the ability to change search conditions is limited.
Dynamic SQL	Because SQL statements are constructed during execution, it is easy to change search conditions.	The SQL statements must be analyzed and converted to execute form each time they are executed, resulting in poor processing efficiency.#

#: Processing efficiency improves when an SQL having the same character string is executed several times.

(2) Values provided at time of execution

When static SQL statements are executed, values to be inserted, new values to be set, and search conditions can be modified. When dynamic SQL statements are executed, any part of the SQL statements, such as the table name, column names, and conditional expressions, can be changed, in addition to those values that can be changed during execution of static SQL statements.

The following examples show values that can be changed during execution of static and dynamic SQL statements. Bold letters indicate the areas where values can be changed.

Figure 3-2: Example of values provided at the time of SQL execution

Example of values provided at time of execution of static SQL statements

UPDATE table-name
SET column-name = **new-value**
WHERE column-name = **conditional-value**

Example of values provided at time of execution of dynamic SQL statements

UPDATE **table-name**
SET **column-name =** **new-value**
WHERE **column-name =** **conditional-value**

(3) Notes on executing dynamic SQL statements

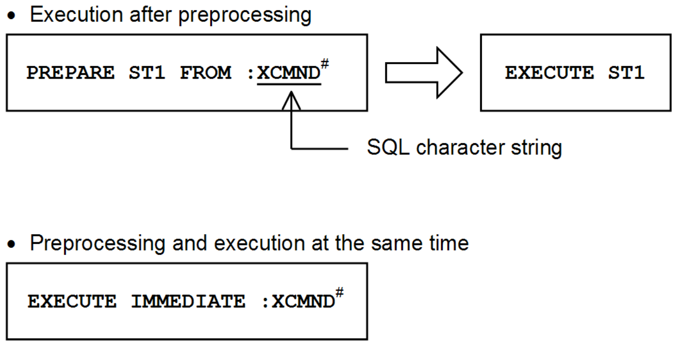
A dynamic SQL provides more flexibility in changing search conditions than a static SQL. However, dynamic SQL statements must be executed each time a condition is changed. For this reason, execution efficiency (processing efficiency) must be considered when deciding whether or not to use a dynamic SQL.

(a) Preprocessing and executing dynamic SQL statements

Dynamic SQL statements need to be processed first by the PREPARE statement and then executed during UAP execution. How a dynamic SQL statement is executed depends on whether the SQL statement to be preprocessed is the dynamic SELECT statement or another statement. If the SQL statement to be preprocessed is the SELECT statement, it is executed with the OPEN, FETCH, and CLOSE statements. If the SQL statement to be preprocessed is a statement other than the dynamic SELECT statement, it is executed with the EXECUTE statement. The EXECUTE IMMEDIATE statement can also be used to both preprocess and execute an SQL statement in a single operation. When the same SQL statement is to be executed dynamically by changing values, ? parameters should be used so that the SQL statement is preprocessed only once, rather than having to preprocess the SQL statement several times; the SQL statement can then be executed repeatedly by changing the values that are assigned to the ? parameters. This results in improved performance (processing efficiency). For details about ? parameters, see the *HiRDB Version 9 SQL Reference* manual.

Figure 3-3 shows the dynamic SQL execution mode, and *Table 3-5* lists the SQLs that can be preprocessed by the PREPARE statement and the SQL statements that can be preprocessed and executed by the EXECUTE IMMEDIATE statement.

Figure 3-3: Dynamic SQL execution mode


Execution after preprocessing
PREPARE ST1 FROM :XCMND#
EXECUTE ST1
SQL character string
Preprocessing and execution at the same time
EXECUTE IMMEDIATE :XCMND#

#: XCMND declares any embedded variables in the embedded variable SQL declaration section. For details about embedded variables, see the HiRDB Version 9 SQL Reference manual.

Table 3-5: SQL statements preprocessed by the PREPARE statement, and SQL statements preprocessed and executed by the EXECUTE IMMEDIATE statement

Type	SQL statement	PREPARE	EXECUTE IMMEDIATE
Data Manipulation SQL	`ASSIGN LIST` statement	U[#3]	U
	`CALL`	U[#3]	U
	`DELETE`[#1]	U[#3]	U
	Preparable dynamic `DELETE` statement: locating	U	U
	`DROP LIST` statement	U[#3]	U
	`INSERT`	U[#3]	U
	`PURGE TABLE`	U[#3]	--
	Single-row `SELECT`[#2]	U[#3]	U
	Dynamic `SELECT`	U[#4]	--
	`UPDATE`[#1]	U[#3]	U
	Preparable dynamic `UPDATE` statement: locating	U	U
	Assignment statement	U[#3]	--

Type	SQL statement	PREPARE	EXECUTE IMMEDIATE
Control SQL	CALL COMMAND	U	U
	COMMIT	--	--
	CONNECT	--	--
	DISCONNECT	--	--
	LOCK TABLE	U#3	U
	ROLLBACK	--	--
	SET SESSION AUTHORIZATION statement	--	--
Definition SQL	ALLOCATE MEMORY TABLE	U#3	U
	ALTER INDEX	U#3	U
	ALTER PROCEDURE	U#3	U
	ALTER ROUTINE	U#3	U
	ALTER TABLE	U#3	U
	ALTER TRIGGER	U#3	U
	COMMENT	U#3	U
	CREATE AUDIT	U#3	U
	CREATE CONNECTION SECURITY	U#3	U
	CREATE FUNCTION	U#3	U
	CREATE INDEX	U#3	U
	CREATE PROCEDURE	U#3	U
	CREATE SCHEMA	U#3	U
	CREATE SEQUENCE	U#3	U
	CREATE TABLE	U#3	U
	CREATE TRIGGER	U#3	U
	CREATE TYPE	U#3	U

Type	SQL statement	PREPARE	EXECUTE IMMEDIATE
	`CREATE VIEW`	U[#3]	U
	`DEALLOCATE MEMORY TABLE`	U[#3]	U
	`DROP AUDIT`	U	U
	`DROP CONNECTION SECURITY`	U[#3]	U
	`DROP DATA TYPE`	U[#3]	U
	`DROP FUNCTION`	U[#3]	U
	`DROP INDEX`	U[#3]	U
	`DROP PROCEDURE`	U[#3]	U
	`DROP SCHEMA`	U[#3]	U
	`DROP SEQUENCE`	U[#3]	U
	`DROP TABLE`	U[#3]	U
	`DROP TRIGGER`	U[#3]	U
	`DROP VIEW`	U[#3]	U
	`GRANT`	U[#3]	U
	`REVOKE`	U[#3]	U

U: Can be used.

--: Cannot be used.

Note

An SQL statement that contains embedded variables cannot be executed dynamically; in this case, `?` parameters must be used instead of embedded variables. For details about `?` parameters, see the *HiRDB Version 9 SQL Reference* manual.

#1: Operations requiring the use of a cursor cannot be performed.

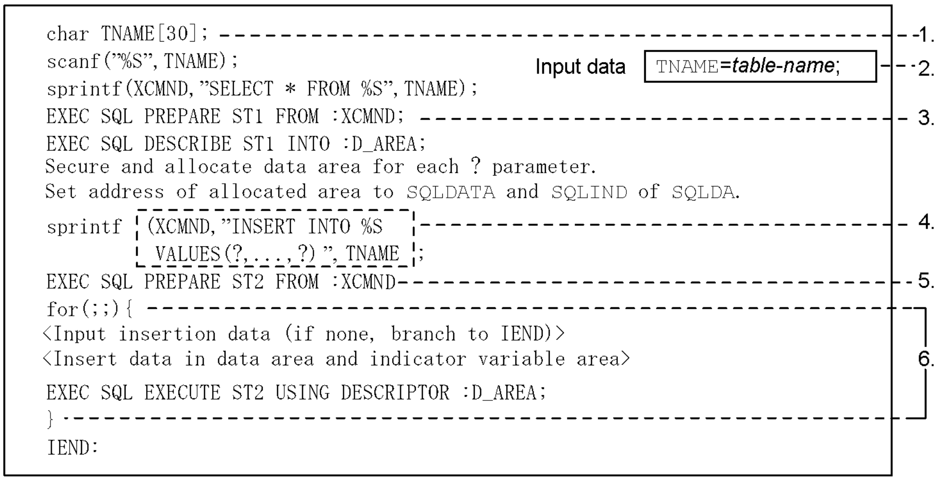
#2: The SQL must not contain an `INTO` clause.

#3: Executed by the `EXECUTE` statement.

#4: Executed by the `OPEN`, `FETCH`, or `CLOSE` statement.

An example of inserting data into a dynamically-specified table is shown as follows:

Figure 3-4: Example of inserting data into a dynamically specified table



```
char TNAME[30]; ------------------------------------------------ 1.
scanf("%S", TNAME);                 Input data  TNAME=table-name;  -- 2.
sprintf(XCMND,"SELECT * FROM %S",TNAME);
EXEC SQL PREPARE ST1 FROM :XCMND; ------------------------------ 3.
EXEC SQL DESCRIBE ST1 INTO :D_AREA;
Secure and allocate data area for each ? parameter.
Set address of allocated area to SQLDATA and SQLIND of SQLDA.
sprintf (XCMND,"INSERT INTO %S
        VALUES(?,...,?)",TNAME); ------------------------------- 4.
EXEC SQL PREPARE ST2 FROM :XCMND-------------------------------- 5.
for(;;){ -------------------------------------------------------
<Input insertion data (if none, branch to IEND)>
<Insert data in data area and indicator variable area>           6.
EXEC SQL EXECUTE ST2 USING DESCRIPTOR :D_AREA;
}---------------------------------------------------------------
IEND:
```

Explanation:

1. Declares a variable (TNAME) that stores the table name.
2. Reads the variable (TNAME) from the input data to the table name.
3. Uses a DESCRIBE statement to set the number of columns in the table specified in 2, and the data type, data length, and maximum number of elements of each column in the SQL descriptor area (D_AREA). These data items are set as the number of ? parameters, and the data type, data length, and maximum number of elements for the data correspond to each ? parameter.
4. Generates an INSERT statement for inserting data into the specified table.
5. Preprocesses the INSERT statement in XCMND and adds an SQL statement identifier (ST2).
6. Repeats input of insertion data for an individual row, setting of data in the data area, and execution according to the EXECUTE statement, as long as there is data to be inserted.

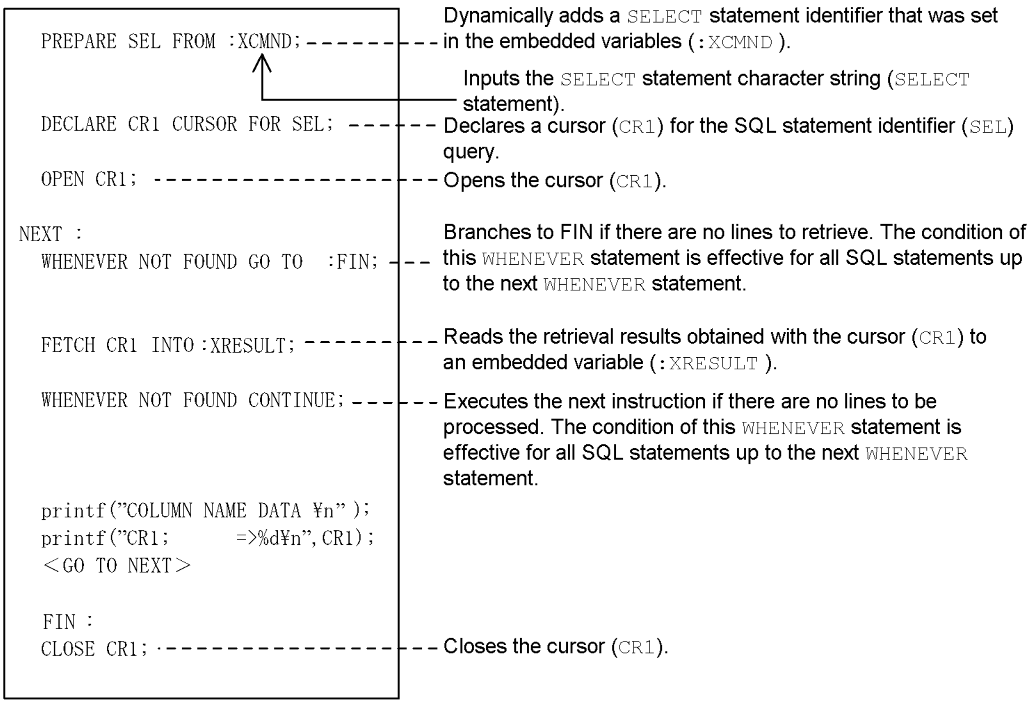
(b) Using the EXECUTE statement and the EXECUTE IMMEDIATE statement

The EXECUTE IMMEDIATE statement is functionally equivalent to executing the PREPARE and EXECUTE statements in succession. When SQL statements are to be executed repeatedly, it is more efficient to execute it iteratively using the EXECUTE statement after first preprocessing it with the PREPARE statement than to execute it several times with the EXECUTE IMMEDIATE statement.

(c) Executing dynamic SQL statements with preprocessing a dynamic SELECT statement

This execution mode varies depending on whether the SQL statement to be preprocessed is a dynamic SELECT statement or a statement other than the dynamic SELECT statement. If the SQL statement to be preprocessed is a dynamic SELECT statement, the SQL statements after preprocessing should be executed using the OPEN, FETCH, or CLOSE statement; if it is not a dynamic SELECT statement, an EXECUTE statement should be used. An example of executing SQL statements with processing a dynamic SELECT statement is shown as follows:

Figure 3-5: Example of dynamic processing when the preprocessed SQL is a dynamic SELECT statement



```
  PREPARE SEL FROM :XCMND;

  DECLARE CR1 CURSOR FOR SEL;

  OPEN CR1;

NEXT :
  WHENEVER NOT FOUND GO TO  :FIN;

  FETCH CR1 INTO :XRESULT;

  WHENEVER NOT FOUND CONTINUE;

  printf("COLUMN NAME DATA ¥n" );
  printf("CR1;      =>%d¥n",CR1);
  <GO TO NEXT>

  FIN :
  CLOSE CR1;
```

Dynamically adds a SELECT statement identifier that was set in the embedded variables (:XCMND).

Inputs the SELECT statement character string (SELECT statement).

Declares a cursor (CR1) for the SQL statement identifier (SEL) query.

Opens the cursor (CR1).

Branches to FIN if there are no lines to retrieve. The condition of this WHENEVER statement is effective for all SQL statements up to the next WHENEVER statement.

Reads the retrieval results obtained with the cursor (CR1) to an embedded variable (:XRESULT).

Executes the next instruction if there are no lines to be processed. The condition of this WHENEVER statement is effective for all SQL statements up to the next WHENEVER statement.

Closes the cursor (CR1).

Note: In this example, the SQL prefix and terminator have been omitted.

(d) Dynamic execution of an SQL statement that uses a cursor for a dynamic SELECT statement

When a dynamic SELECT statement is preprocessed and an SQL statement that uses a cursor is executed dynamically for that dynamic SELECT statement, a cursor declared in a cursor declaration is not used. In this case, a cursor allocated with the ALLOCATE CURSOR statement is used for the preprocessed dynamic SELECT statement. An example of dynamic execution of an SQL statement that uses a cursor for a dynamic SELECT statement is shown below.

```
PREPARE GLOBAL :SEL FROM :XCMND;
//Adds an extended statement name (:SEL='SEL1') to the dynamic SELECT statement that was set to an embedded variable
(:XCMND).
ALLOCATE GLOBAL :CR CURSOR FOR GLOBAL :SEL;
//Allocates a cursor (:CR='CR1') to the query identified by the extended statement name (:SEL='SEL1').
PREPARE UPD1 FROM
  'UPDATE SET C1=? WHERE CURRENT OF GLOBAL CR1';
//Preprocesses the UPDATE statement that uses the cursor (CR1) and attaches an SQL statement identifier (UPD1).
OPEN GLOBAL :CR;
//Adds a cursor (:CR='CR1').
FETCH GLOBAL :CR INTO :XKEKKA;
//Reads the search results obtained using the cursor (:CR='CR1') into an embedded variable (:XKEKKA).
EXECUTE UPD1 USING :XDATA;
//Executes the UPDATE statement for the preprocessed SQL statement identifier (UPD1). At this time, the embedded variable
(:XDATA) corresponding to the ? parameter is specified.
CLOSE GLOBAL :CR;
//Closes the cursor (:CR='CR1').
```

(e) Receiving information determined during dynamic SQL execution

When a UAP dynamically executes SQL statements, it uses an SQL Descriptor Area as the area for notifying HiRDB about information determined during the execution (including the number, attributes, and addresses of data transfer areas). To realize dynamic execution, the UAP receives search item information for SQL statements preprocessed with the `PREPARE` statement in the SQL Descriptor Area by using one of the following methods:

- Executing the `DESCRIBE` statement
- Specifying `OUTPUT` and `INPUT` when executing the `PREPARE` statement. (When this method is used, the number of communications might be reduced, because information can be received during execution of the `PREPARE` statement.)

For details about the `DESCRIBE` statement, see the manual *HiRDB Version 9 SQL Reference*. For an example of the use of SQL Descriptor Areas, see *B. SQL Descriptor Area*.

3.3 Transaction control

This section explains when a UAP starts and terminates a transaction in a HiRDB system, setting synchronization points, handling transactions, and rollbacks.

3.3.1 Connection to and disconnection from a HiRDB system

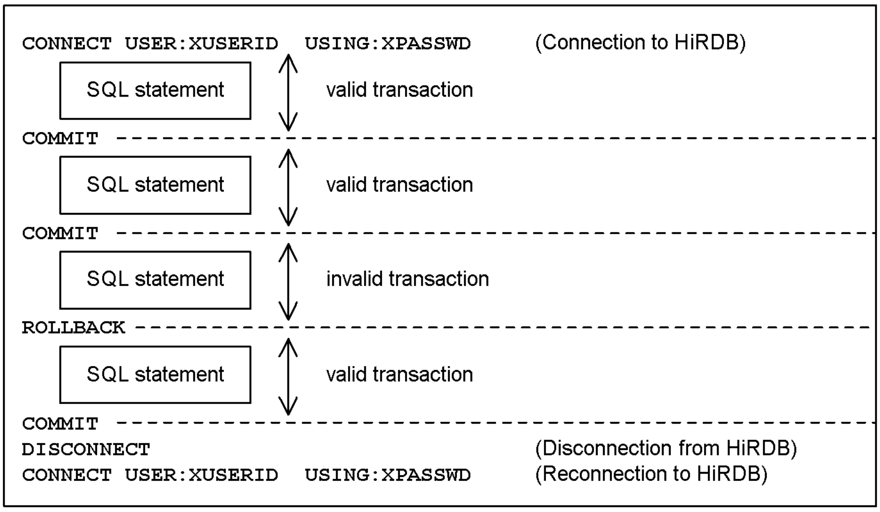
Executing the `CONNECT` statement connects a UAP to a HiRDB system, and executing the `DISCONNECT` statement disconnects them.

3.3.2 Transaction startup and termination

A transaction is started when an SQL statement of the UAP is executed and is terminated when a `COMMIT` or `ROLLBACK` statement is executed. Any number of transactions can be started and terminated while the UAP is connected to the HiRDB system.

The following figure shows examples of transaction startup and termination.

Figure 3-6: Examples of transaction startup and termination



In a HiRDB/Parallel Server, processing of SQL statements is branched to multiple servers; but a process is managed as one transaction, and you do not need to consider the internal branches.

3.3.3 Synchronization point setting and rollback

The following table explains the setting of synchronization points and the handling of transactions.

Table 3-6: Synchronization points and transactions

Synchronization point	Set by:	Handling transactions
Set points in UAP by executing SQL statements	Executing `COMMIT` statement	Validated[#1]
	Executing `ROLLBACK` statement	Invalidated[#1, #2]
Set points in HiRDB by executing SQL statements	Executing definition SQL statements	Validated[#1]
	Executing `PURGE TABLE` statement	Validated[#1]
	Processing cannot be continued while executing SQL statements	Invalidated[#3]
Set points in HiRDB by terminating UAP	UAP normal termination	Validated
	UAP abnormal termination	Invalidated[#2]

#1: Cannot be executed in the OLTP environment. For details about synchronization point setting and rollback in the OLTP environment, see *3.3.4 UAP transaction management in an OLTP environment*.

#2: Results in implicit rollback; the following are major causes of implicit rollback:

#3: When a transaction is invalidated, all transactions since the most recent synchronization point are invalidated.

- Deadlock
- RDAREA page shortage
- Detection of RDAREA error or shutdown

3.3.4 UAP transaction management in an OLTP environment

In OLTP, you cannot code the `COMMIT` and `ROLLBACK` statements. When synchronization point setting or transaction rollback occurs in a UAP executing in this environment, you must use an application program interface (API) that conforms to X/Open.

An example using OpenTP1 is explained here. For details about how to create a program by using OpenTP1, see the manual *OpenTP1 Version 5 Program Reference C Language* and the manual *OpenTP1 Version 5 Program Reference COBOL Language*.

A remote procedure call (RPC) can be used to implement one transaction among multiple OLTP user server processes. Each process is called a *transaction branch*, and the totality of these processes is called an *OLTP global transaction*.

When HiRDB is accessed from an OLTP global transaction, HiRDB cannot be accessed by the multiple transaction branches that make up the global transaction branch.

Sometimes when a resource is to be accessed, a timeout occurs because a lock on the resource was issued by a preceding transaction branch in the global transaction branch, thus causing a succeeding transaction branch to wait until the resource becomes available. Similarly, deadlock can occur between transaction branches.

The chain RPC function can be used in such a situation so that multiple RPCs can be treated as belonging to the same transaction branch.

(1) C

(a) Transaction startup

Code the `tx_begin` function in the UAP.

(b) Setting synchronization points

Code the `tx_commit` function in the UAP.

(c) Setting rollbacks

Code the `tx_rollback` function in the UAP.

(2) COBOL85

(a) Transaction startup

```
DATA DIVISION.

*include TX definitions.

  01 TX-RETURN_STATUS
     COPY TXSTATUS.

PROCEDURE DIVISION.
CALL "TXBEGIN" USING TX-RETURN_STATUS.
```

(b) Setting synchronization points

```
DATA DIVISION.

*include TX definitions.

  01 TX-RETURN_STATUS
     COPY TXSTATUS.

PROCEDURE DIVISION.
CALL "TXCOMMIT" USING TX-RETURN_STATUS.
```

(c) Setting rollbacks

```
DATA DIVISION.
```

```
*include TX definitions.

  01 TX-RETURN_STATUS
     COPY TXSTATUS.

PROCEDURE DIVISION.
CALL "TXROLLBACK" USING TX-RETURN_STATUS.
```

3.3.5 Moving a transaction

When a UAP commits a transaction in a process different from the process in which the UAP accessed HiRDB, the commitment processing is called *moving the transaction*.

The UAP referenced is a UAP that connects itself to HiRDB via the HiRDB XA library.

When the transaction-move function is used, `1` must be specified in the `PDXAMODE` operand of the client environment definition. For details about the `PDXAMODE` operand, see *6.6.4 Environment definition information*.

(1) Scope of LOCK TABLE UNTIL DISCONNECT when the PDXAMODE operand is specified

The specification of the `PDXAMODE` operand affects the scope of the `LOCK TABLE UNTIL DISCONNECT` specification, as explained as follows:

(a) PDXAMODE=0

1. Resource Manager opened by means of AP coding

   The `LOCK TABLE UNTIL DISCONNECT` specification remains in effect until the Resource Manager is closed.

2. Resource Manager opened separately for each transaction

   The `LOCK TABLE UNTIL DISCONNECT` specification remains in effect throughout the global transaction.

(b) PDXAMODE=1

1. Resource Manager opened by means of AP coding

   - Transaction is not moved

     The `LOCK TABLE UNTIL DISCONNECT` specification remains in effect until the Resource Manager is closed.

   - Transaction is moved

     The `LOCK TABLE UNTIL DISCONNECT` specification remains in effect throughout the global transaction.

2. Resource Manager opened separately for each transaction

   The `LOCK TABLE UNTIL DISCONNECT` specification remains in effect throughout the global transaction.

The following table shows the scope of the `LOCK TABLE UNTIL DISCONNECT` specification when OpenTP1 is used.

Table 3-7: Scope of the LOCK TABLE UNTIL DISCONNECT specification when OpenTP1 is used

<table>
<tr><th>PDXAMODE specification</th><th colspan="4">OpenTP1 specification</th><th>Scope of LOCK TABLE UNTIL DISCONNECT</th></tr>
<tr><td rowspan="2">0</td><td colspan="4">trn_rm_open_close_scope=process</td><td>Effective until Resource Manager is closed.</td></tr>
<tr><td colspan="4">trn_rm_open_close_scope=transaction</td><td>Effective within a global transaction.</td></tr>
<tr><td rowspan="5">1</td><td rowspan="4">trn_rm_open_close_scope=process</td><td colspan="3">-d option specified in trnstring operand</td><td rowspan="3">Effective until the Resource Manager is closed.</td></tr>
<tr><td rowspan="3">-d option not specified in trnstring operand</td><td colspan="2">A single AP comprises a global transaction in the OpenTP1 system.</td></tr>
<tr><td rowspan="2">Multiple APs comprise a global transaction in the OpenTP1 system.</td><td>A single AP links to the HiRDB XA library.</td></tr>
<tr><td>Multiple APs link to the HiRDB XA library.</td><td rowspan="2">Effective within the global transaction.</td></tr>
<tr><td colspan="4">trn_rm_open_close_scope=transaction</td></tr>
</table>

Note

The `-d` option can be specified when the TP1/Server Base version is 03-03 or later and the HiRDB version is for UNIX systems.

3.4 Locking

The HiRDB system automatically locks tables to prevent data inconsistencies, because data inconsistencies are apt to occur when several users manipulate a table simultaneously. This section explains the structure of locking and what aspects of locking the user can change.

3.4.1 Units of locking

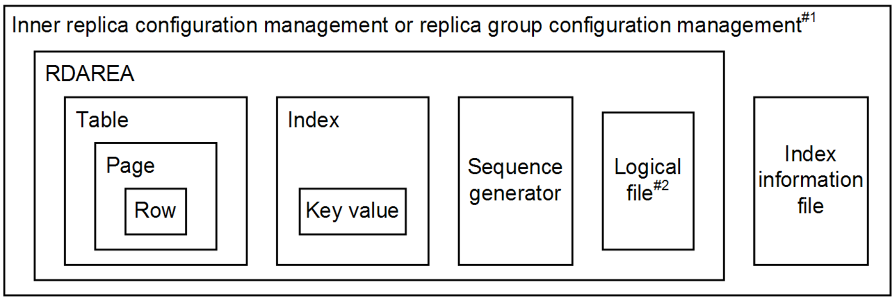
(1) Locked resources and inclusive relationships

HiRDB locks a resource to prevent unauthorized referencing or updating.

HiRDB performs locking to maintain database integrity. In a HiRDB/Parallel Server, *closed locking* is performed for each server, because resources are not shared among servers.

When a higher-level resource is locked, the resources under that resource need not be locked, because locked resources maintain inclusive relationships. The following figure shows the resources that can be locked and their inclusive relationships.

Figure 3-7: Locked resources and inclusive relationships



#1: When the inner replica facility is used, the highest locked resource is the inner replica configuration management information or the replica group configuration management information.

If the inner replica configuration management information cannot be locked, the replica group configuration management information is locked. When an RDAREA is accessed, it is locked even if no replica RDAREA has been defined for it. This prevents a replica RDAREA from being defined for a normal RDAREA or the configuration of an inner replica group from being changed during job execution.

#2: This file is used by plug-ins.

(2) Setting the minimum unit of resource locking

For purposes of lock control that the HiRDB system implements automatically, the minimum unit of resource locking (the row or the page) can be specified for each table.

You can also disable lock control with the index key value for an index. This setting is called *non-locking of index key values*.

(a) When the row is specified as the minimum unit of resource locking

Because the row is a smaller unit of resource locking than the page, the efficiency of concurrent execution improves, but processing time and memory requirements associated with locking increase.

To specify the row as the minimum unit of resource locking, use the `CREATE TABLE`, `ALTER TABLE`, or `LOCK` statement. For details, see the *HiRDB Version 9 SQL Reference* manual.

(b) When the page is specified as the minimum unit of resource locking

Compared with row-level locking, the processing time and memory requirements associated with locking decrease, but the efficiency of concurrent execution is reduced.

To specify the row as the minimum unit of resource locking, use the `CREATE TABLE`, `ALTER TABLE`, or `LOCK` statement. For details, see the *HiRDB Version 9 SQL Reference* manual.

(c) When non-locking of index key values is specified

Locking is applied only to the table and not to the index key value. This setting allows you to avoid the following problems:

- Deadlock between data update and index retrieval
- Unnecessary wait when data that has the same key is accessed
- Unnecessary wait when data that has a different key is accessed

For details about non-locking of index key values, see *3.4.6 Non-locking of index key values*.

These three settings each have different tradeoffs. These tradeoffs must be considered when the minimum unit of resource locking is specified.

3.4.2 Lock modes

(1) Mode types

You can apply five lock modes to a resource, as explained as follows:

1. Protected retrieve (PR) mode

   In the PR mode, only the first transaction that uses the resource occupies it and can reference, add to, update, and delete data in the locked resource. Other transactions can only reference the locked resource.

2. Exclusive (EX) mode

   In the EX mode, only the transaction that uses the resource occupies it and can reference, add to, update, and delete data in the locked resource. Other transactions cannot reference, add to, update, or delete the locked resource.

3. Shared retrieve (SR) mode

   In the SR mode, if a lock is applied in PR mode to a certain resource, the lock is applied to the resource that is located above that resource. Other transactions can also reference, add to, update, and delete the locked resource.

4. Shared update (SU) mode

   In the SU mode, if a lock is applied in EX mode to a certain resource, the lock is applied to the resource that is located above that resource. Other transactions can also reference, add to, update, and delete the locked resource.

5. Protected update (PU) mode

   In the PU mode, the first transaction that uses the resource can reference, add to, update, and delete data in it; other transactions can only reference the locked resource.

   Unlike the first four modes, the PU mode occurs as a result of locking mode transition.

Locking is applied first to the highest-level resource and then to lower-level resources. If a transaction cannot be executed simultaneously with other transactions that have locking in effect for the same resource, that transaction goes onto wait status. When the PR or EX mode is encountered while locking is being applied from a higher-level resource to a lower-level resource, a resource that is located below the resource to which the mode has been applied is not locked.

When two users attempt to perform identical processing on the same resource, the difference in the combination of the lock modes may prevent simultaneous execution. The following table shows when two users can perform execution simultaneously based on lock modes.

Table 3-8: Simultaneous execution by two users based on lock modes

Mode	SR	PR	SU	PU	EX
SR	A	A	A	A	NA
PR	A	A	NA	NA	NA
SU	A	NA	A	NA	NA
PU	A	NA	NA	NA	NA
EX	NA	NA	NA	NA	NA

A: Simultaneous execution allowed

NA: Simultaneous execution not allowed

When two users cannot perform execution simultaneously, the system usually waits for the other transaction to be committed (updated at the synchronization point). `WITH ROLLBACK, or NOWAIT` can be specified in the SQL statement to cause an error return without waiting for the other transaction to be committed.

(2) Mode transition

If the user applies different locking modes repeatedly to the same resource, the mode shifts to the stronger one.

After locking has been applied using a strong mode, applying a weaker mode does not cause the mode to shift to the weaker one. For example, if EX is used for locking during row updating, the lock mode of the row remains as EX, even if PR is applied subsequently for referencing the updated row.

The following table shows the lock mode transition rules when a lock is applied to the current lock.

Table 3-9: Lock mode transition rules

Mode applied subsequently	**Current mode**				
	SR	**PR**	**SU**	**PU**	**EX**
SR	--	--	--	--	--
PR	PR	--	PU	--	--
SU	SU	PU	--	--	--
EX	EX	EX	EX	EX	--

--: Mode transition does not occur.

(code): Mode after transition.

(3) Mode combinations

Different lock modes can be combined, depending on the SQL statement and the execution environment.

Tables 3-10 and *3-11* show typical lock mode combinations based on the SQL statement and execution environment for row-level locking. *Tables 3-12* and *3-13* show similar combinations for page-level locking, and *Tables 3-14* and *3-15* show them for non-locking of index key values. *Tables 3-16* and *3-17* show typical lock mode combinations for cases when a table is set to check pending status.

Table 3-10: Typical lock mode combinations (row locking) (1/2)

SQL statement and execution environment		Resource							
		Higher level -------------------- Lower level							
		Inner replica config #6	Replica group config #8	RDAREA				Table	Table (NO WAIT search)
				For tables	For IX	For sequence generator	Last HiRDB file#5		
Retrieval	NOWAIT specified	SR	SR	SR		--	--	--	SR
	WITH SHARE specified	SR	SR	SR		--	--	SR	--
	WITH EXCLUSIVE specified#1	SR	SR	SU	SR	--	--	SU	--
	FOR UPDATE clause specified#1	SR	SR	SU	SR	--	--	SU	--
	None of the above	SR	SR	SR		--	--	SR	--
Updating #1, #12	NEXT VALUE clause specified	SR		SU		SU	EX	SU	--
	None of the above	SR		SU		--	EX	SU	--
Addition #1	NEXT VALUE clause specified	SR		SU		SU	EX	SU	--
	None of the above	SR		SU		--	EX	SU	--
Deletion#1		SR	SR	SU		--	--	SU	--

SQL statement and execution environment			Resource							
			Higher level --------------------- Lower level							
			Inner replica config #6	Replica group config #8	RDAREA				Table	Table (NO WAIT search)
					For tables	For IX	For sequence generator	Last HiRDB file#5		
`LOCK` statement	`SHARE` specified#11		SR	SR	SR	--	--	--	PR	--
	`EXCLUSIVE` specified	Unshared table	SR	SR	SU	--	--	--	EX	--
		Shared table#11	SR	SR	EX	--	--	--	EX	--
Table deletion#2, #13			--	--	SU		--	--	EX	
Index	Definition#13		--	--	SU		--	--	EX	--
	Deletion#3, #13		--	--	SR#10	SU	--	--	EX#4	EX
Deletion of all rows#2, #13, #14			SR	SR	SU		--	--	EX	
Table definition change#13			SR#9	SR#9	SU#7		--	--	EX	
Definition of sequence generator			--	--	--	--	SU	--	--	--
Deletion of sequence generator			--	--	--	--	SU	--	--	--

--: Locking is not applied.

(code): Lock mode.

IX: indexes

#1: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during

UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#2: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#3: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#4: Plug-in indexes are locked in the EX mode, but B-Tree indexes are not locked.

#5: If automatic extension of the RDAREA is applied, the last HiRDB file that makes up the RDAREA is locked from start to end of the automatic extension processing.

#6: If the inner replica facility is used, the server containing the RDAREA to be processed is locked. However, if `Y` is specified in the `pd_inner_replica_lock_shift` operand, the corresponding server is not locked.

#7: If an RDAREA is added or is altered with the free space reusage facility, the RDAREA is locked.

#8: If a change related to the inner replica configuration is made, such as changing the current database, defining or deleting a replica, or performing updatable online reorganization, the replica group containing the RDAREA to be processed is locked. A lock is always applied when `Y` is specified in the `pd_inner_replica_lock_shift` operand.

#9: If an RDAREA to be processed is accessed, the RDAREA is locked.

#10: If the inner replica facility is applied, the resource is locked.

#11: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#12: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#13: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index. When HiRDB/Single Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied when the operation is executed for a shared table or a shared index.

#14: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-11: Typical lock mode combinations (row locking) (2/2)

SQL statement and execution environment			Resource						
			Higher level --------------------- Lower level						
			Index	Index information file[#4]	Sequence generator	Page	Row	Key value	Logical file
Retrieval	`NOWAIT` specified		--	--	--	--	--	--	PR
	`WITH SHARE` specified		--	--	--	--	PR	PR	PR
	`WITH EXCLUSIVE` specified[#1]		--	--	--	--	EX	PR	EX
	`FOR UPDATE` clause specified[#1]		--	--	--	--	EX	PR	EX
	None of the above		--	--	--	--	PR	PR	PR
Updating[#1, #6]	`NEXT VALUE` clause specified		--	EX	SU	--	EX	EX	EX
	None of the above		--	EX	--	--	EX	EX	EX
Addition[#1]	`NEXT VALUE` clause specified		--	EX	SU	--	EX	EX	EX
	None of the above		--	EX	--	--	EX	EX	EX
Deletion[#1]			--	--	--	--	EX[#8]	EX	EX
`LOCK` statement	`SHARE` specified[#5]		--	--	--	--	--	--	--
	`EXCLUSIVE` specified	Unshared table	--	--	--	--	--	--	--
		Shared table[#5]	--	--	--	--	--	--	--
Table deletion[#2, #7]			--	--	--	--	--	--	--
Index	Definition[#7]		--	--	--	--	--	--	--
	Deletion[#3, #7]		EX	--	--	--	--	--	--
Deletion of all rows[#2, #7, #9]			--	--	--	--	--	--	--

SQL statement and execution environment	Resource						
	Higher level -------------------- Lower level						
	Index	Index information file[#4]	Sequence generator	Page	Row	Key value	Logical file
Table definition change[#7]	--	--	--	--	--	--	--
Definition of sequence generator	--	--	EX	--	--	--	--
Deletion of sequence generator	--	--	EX	--	--	--	--

--: Locking is not applied.

(code): Lock mode.

#1: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#2: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#3: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#4: The delayed batch creation facility for plug-in indexes is used to lock a plug-in index when it is updated. The lock is held until a `commit` statement is executed.

#5: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#6: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#7: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index.

#8: The row to be deleted is locked in the EX mode until the transaction is committed or rolled back. However, if another transaction executes retrieval processing while the row is being deleted, the retrieval processing does not wait for lock-release because it cannot apply a lock on the row to be deleted.]

#9: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-12: Typical lock mode combinations (page locking) (1/2)

SQL statement and execution environment		Resource							
		Higher level -------------------- Lower level							
		Inner replica config #6	Replica group config #8	RDAREA				Table	Table (NOWAIT search)
				For tables	For IX	For sequence generator	Last HiRDB file#5		
Retrieval	`NOWAIT` specified	SR	SR	SR		--	--	--	SR
	`WITH SHARE` specified	SR	SR	SR		--	--	SR	--
	`WITH EXCLUSIVE` specified#1	SR	SR	SU	SR	--	--	SU	--
	`FOR UPDATE` clause specified#1	SR	SR	SU	SR	--	--	SU	--
	None of the above	SR	SR	SR		--	--	SR	--
Updating #1, #12	`NEXT VALUE` clause specified	SR	SR	SU		SU	EX	SU	--
	None of the above	SR	SR	SU		--	EX	SU	--
Addition #1	`NEXT VALUE` clause specified	SR	SR	SU		SU	EX	SU	--
	None of the above	SR	SR	SU		--	EX	SU	--
Deletion#1		SR	SR	SU		--	--	SU	--

SQL statement and execution environment			Resource							
			Higher level -------------------- Lower level							
			Inner replica config #6	Replica group config #8	RDAREA				Table	Table (NOWAIT search)
					For tables	For IX	For sequence generator	Last HiRDB file#5		
LOCK statement	SHARE specified#11		SR	SR	SR	--	--	--	PR	--
	EXCLUSIVE specified	Unshared table	SR	SR	SU	--	--	--	EX	--
		Shared table#11	SR	SR	EX	--	--	--	EX	--
Table deletion#2, #13			--	--	SU		--	--	EX	EX
Index	Definition#13		--	--	SU		--	--	EX	--
	Deletion#3, #13		--	--	SR#10	SU	--	--	EX#4	EX
Deletion of all rows#2, #13, #14			SR	SR	SU		--	--	EX	EX
Table definition change#13			SR#9	SR#9	SU#7		--	--	EX	EX
Definition of sequence generator			--	--	--		SU	--	--	--
Deletion of sequence generator			--	--	--		SU	--	--	--

--: Locking is not applied.

(code): Lock mode.

IX: indexes

#1: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during

UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#2: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#3: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#4: Plug-in indexes are locked in the EX mode, but B-Tree indexes are not locked.

#5: If automatic extension of the RDAREA is applied, the last HiRDB file that makes up the RDAREA is locked from start to end of the automatic extension processing.

#6: If the inner replica facility is used, the server containing the RDAREA to be processed is locked. However, if `Y` is specified in the `pd_inner_replica_lock_shift` operand, the server is not locked.

#7: If an RDAREA is added or is altered with the free space reusage facility, the RDAREA is locked.

#8: If a change related to the inner replica configuration is made, such as changing the current database, defining or deleting a replica, or performing updatable online reorganization, the replica group containing the RDAREA to be processed is locked. A lock is always applied when `Y` is specified in the `pd_inner_replica_lock_shift` operand.

#9: If an RDAREA to be processed is accessed, the RDAREA is locked.

#10: If the inner replica facility is applied, the resource is locked.

#11: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#12: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#13: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index. When HiRDB/Single Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied when the operation is executed for a shared table or a shared index.

#14: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-13: Typical lock mode combinations (page locking) (2/2)

SQL statement and execution environment			**Resource**						
			Higher level -------------------- Lower level						
			Index	**Index information file[#4]**	**Sequence generator**	**Page**	**Row**	**Key value**	**Logical file**
Retrieval	`NOWAIT` specified		--	--	--	--	--	--	PR
	`WITH SHARE` specified		--	--	--	PR	--	PR	PR
	`WITH EXCLUSIVE` specified[#1]		--	--	--	EX	--	PR	EX
	`FOR UPDATE` clause specified[#1]		--	--	--	EX	--	PR	EX
	None of the above		--	--	--	PR	--	PR	PR
Updating[#1, #6]	`NEXT VALUE` clause specified		--	EX	SU	EX	--	EX	EX
	None of the above		--	EX	--	EX	--	EX	EX
Addition[#1]	`NEXT VALUE` clause specified		--	EX	SU	EX	--	EX	EX
	None of the above		--	EX	--	EX	--	EX	EX
Deletion[#1]			--	--	--	EX	--	EX	EX
`LOCK` statement	`SHARE` specified[#5]		--	--	--	--	--	--	--
	`EXCLUSIVE` specified	Unshared table	--	--	--	--	--	--	--
		Shared table[#5]	--	--	--	--	--	--	--
Table deletion[#2, #7]			--	--	--	--	--	--	--
Index	Definition[#7]		--	--	--	--	--	--	--
	Deletion[#3, #7]		EX	--	--	--	--	--	--
Deletion of all rows[#2, #7, #8]			--	--	--	--	--	--	--

SQL statement and execution environment	Resource						
	Higher level -------------------- Lower level						
	Index	Index information file[#4]	Sequence generator	Page	Row	Key value	Logical file
Table definition change[#7]	--	--	--	--	--	--	--
Definition of sequence generator	--	--	EX	--	--	--	--
Deletion of sequence generator	--	--	EX	--	--	--	--

--: Locking is not applied.

(code): Lock mode.

#1: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#2: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#3: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#4: The delayed batch creation facility for plug-in indexes is used to lock a plug-in index when it is updated. The lock is held until a `commit` statement is executed.

#5: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#6: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#7: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index.

#8: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-14: Typical lock mode combinations (non-locking of index key values) (1/2)

SQL statement and execution environment		Resource							
		Higher level -------------------- Lower level							
		Inner replica config #5	Replica group config #7	RDAREA				Table	Table (NOWAIT search)
				For tables	For IX	For sequence generator	Last HiRDB file#4		
Retrieval	`NOWAIT` specified	SR	SR	SR		--	--	--	SR
	`WITH SHARE` specified	SR	SR	SR		--	--	SR	--
	`WITH EXCLUSIVE` specified#10	SR	SR	SU	SR	--	--	SU	SU
	`FOR UPDATE` clause specified#10	SR	SR	SU	SR	--	--	SU	SU
	None of the above	SR	SR	SR		--	--	SR	--
Updating #10, #12	`NEXT VALUE` clause specified	SR	SR	SU		SU	EX	SU	--
	None of the above	SR	SR	SU		--	EX	SU	--
Addition #10	`NEXT VALUE` clause specified	SR	SR	SU		SU	EX	SU	--
	None of the above	SR	SR	SU		--	EX	SU	--
Deletion#10		SR	SR	SU		--	--	SU	--

SQL statement and execution environment			Resource							
			Higher level -------------------- Lower level							
			Inner replica config #5	Replica group config #7	RDAREA				Table	Table (NOWAIT search)
					For tables	For IX	For sequence generator	Last HiRDB file#4		
LOCK statement	SHARE specified#11		SR	SR	SR	--	--	--	PR	--
	EXCLUSIVE specified	Unshared table	SR	SR	SU	--	--	--	EX	--
		Shared table #11	SR	SR	EX	EX	--	--	EX	--
Table deletion#1, #13			--	--	SU		--	--	EX	EX
Index	Definition#13		--	--	SU		--	--	EX	--
	Deletion#2, #13		--	--	SR#9	SU	--	--	EX#3	EX
Deletion of all rows#1, #13, #14			SR	SR	SU		--	--	EX	EX
Table definition change#13			SR#8	SR#8	SU#6		--	--	EX	EX
Definition of sequence generator			--	--	--		SU	--	--	--
Deletion of sequence generator			--	--	--		SU	--	--	--

--: Locking is not applied.

(code): Lock mode.

IX: indexes

#1: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#2: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#3: Plug-in indexes are locked in the EX mode, but B-Tree indexes are not locked.

#4: If automatic extension of the RDAREA is applied, the last HiRDB file that makes up the RDAREA is locked from start to end of the automatic extension processing.

#5: If the inner replica facility is used, the server containing the RDAREA to be processed is locked. However, if `Y` is specified in the `pd_inner_replica_lock_shift` operand, the server is not locked.

#6: If an RDAREA is added or is altered with the free space reusage facility, the RDAREA is locked.

#7: If a change related to the inner replica configuration is made, such as changing the current database, defining or deleting a replica, or performing updatable online reorganization, the replica group containing the RDAREA to be processed is locked. A lock is always applied when `Y` is specified in the `pd_inner_replica_lock_shift` operand.

#8: If an RDAREA to be processed is accessed, the RDAREA is locked.

#9: If the inner replica facility is applied, the resource is locked.

#10: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#11: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#12: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#13: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index. When HiRDB/Single Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied when the operation is executed for a shared table or a shared index.

#14: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-15: Typical lock mode combinations (non-locking of index key values) (2/2)

SQL statement and execution environment		Resource						
		Higher level -------------------- Lower level						
		Index	Index information file#3	Sequence generator	Page	Row	Key value	Logical file
Retrieval	`NOWAIT` specified	--	--	--	--	--	--	PR
	`WITH SHARE` specified	--	--	--	--, PR#4	PR, --# 4	--	PR
	`WITH EXCLUSIVE` specified#6	--	--	--	--, EX#4	EX, --# 4	--	EX
	`FOR UPDATE` clause specified#6	--	--	--	--, EX#4	EX, --# 4	--	EX
	None of the above	--	--	--	-- PR#4	PR, --# 4	--	PR
Updating #6, 8	`NEXT VALUE` clause specified	--	EX	SU	--, EX#4, #5	EX, --#4, #5	--	EX
	None of the above	--	EX	--	--, EX#4, #5	EX, --#4, #5	--	EX
Addition #6	`NEXT VALUE` clause specified	--	EX	SU	--, EX#4, #5	EX, --# 4, #5	--	EX
	None of the above	--	EX	--	--, EX#4, #5	EX, --#4, #5	--	EX
Deletion#6		--	--	--	-- EX#4, #5	EX, --#4, #5	--	EX

SQL statement and execution environment			Resource						
			Higher level -------------------- Lower level						
			Index	Index information file#3	Sequence generator	Page	Row	Key value	Logical file
LOCK statement	SHARE specified#7		--	--	--	--	--	--	--
	EXCLUSIVE specified	Unshared table	--	--	--	--	--	--	--
		Shared table #7	--	--	--	--	--	--	--
Table deletion#1, #9			--	--	--	--	--	--	--
Index	Definition#9		--	--	--	--	--	--	--
	Deletion#2, #9		EX	--	--	--	--	--	--
Deletion of all rows#1, #9, #10			--	--	--	--	--	--	--
Table definition change#9			--	--	--	--	--	--	--
Definition of sequence generator			--	--	EX	--	--	--	--
Deletion of sequence generator			--	--	EX	--	--	--	--

--: Locking is not applied.

(code): Lock mode.

#1: All segments being used for the table and associated indexes are locked in the EX mode and kept locked until the transaction is committed.

#2: All segments being used for the index are locked in the EX mode and kept locked until the transaction is committed.

#3: The delayed batch creation facility for plug-in indexes is used to lock a plug-in index when it is updated. The lock is held until a `commit` statement is executed.

#4: In row locking, resource rows are locked and resource pages are not locked. In page locking, resource rows are not locked, and resource pages are locked.

#5: If a unique index is defined, resource rows are locked, even in page locking.

#6: If the database update log is not being collected during UAP execution, a table is locked in the EX mode and kept locked until it is committed; rows and keys are not locked. For details about operations when a database update log is not collected during UAP execution, see the *HiRDB Version 9 System Operation Guide*.

#7: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when a shared table is accessed.

#8: When HiRDB/Parallel Server is used, locking is applied to all back-end servers when an `UPDATE` statement that does not update the index is executed for a shared table.

#9: When HiRDB/Parallel Server is used, locking equivalent to an `EXCLUSIVE`-specified `LOCK` statement is applied to all back-end servers when the operation is executed for a shared table or a shared index.

#10: If `USE` or nothing is specified in the `pd_check_pending` operand in the system definition, the data dictionary table (resource type: `3005`, type name: `DICT`) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: `0001`, type name: `RDAR`) is locked in the SU mode until the transaction terminates.

Table 3-16: Typical lock mode combinations (when check pending status is set) (1/2)

SQL statement and utility	**Resource#1**					
	Higher level -------------------- Lower level					
	RDAREA				**Table**	**Table (NO WAIT search)**
	For tables#2	**For indexes**	**For LOB**	**Last HiRDB file**		
Deletion of all rows#4	SU	--	--	--	EX	EX
Changing a table's definition (changing the storage partitioning conditions)	SU	--	--	--	EX	EX
Database load utility (`pdload`)#3	SU	--	--	--	EX	EX
Database reorganization utility (`pdrorg`)#3	SU	--	--	--	EX	EX

<table>
<tr><th rowspan="4">SQL statement and utility</th><th colspan="6">Resource#1</th></tr>
<tr><th colspan="6">Higher level -------------------- Lower level</th></tr>
<tr><th colspan="4">RDAREA</th><th rowspan="2">Table</th><th rowspan="2">Table (NO WAIT search)</th></tr>
<tr><th>For tables#2</th><th>For indexes</th><th>For LOB</th><th>Last HiRDB file</th></tr>
<tr><td>Database structure modification utility (pdmod)</td><td>SU</td><td>--</td><td>--</td><td>--</td><td>EX</td><td>EX</td></tr>
<tr><td>Integrity check utility (pdconstck)#3</td><td>SU</td><td>--</td><td>--</td><td>--</td><td>EX</td><td>EX</td></tr>
<tr><td>Reflection command for online reorganization (pdorend)#3</td><td>SU</td><td>--</td><td>--</td><td>--</td><td>EX</td><td>EX</td></tr>
</table>

--: Locking is not applied.

(code): Lock mode.

#1

This table shows resources for tables in which a referential constraint or a check constraint is defined.

#2

Locking is applied to the RDAREAs in which the check pending status is to be set.

#3

When a HiRDB/Parallel Server is used, locking equivalent to a LOCK statement with EXCLUSIVE specified is applied to all back-end servers when the utility is executed for a shared table. For details about the lock mode applied during execution of the LOCK statement on a shared table, see the EXCLUSIVE *specified* rows under LOCK *statement* in *Tables 3-10* to *3-15*.

#4

If USE or nothing is specified in the pd_check_pending operand in the system definition, the data dictionary table (resource type: 3005, type name: DICT) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: 0001, type name: RDAR) is locked in the SU mode until the transaction terminates.

Table 3-17: Typical lock mode combinations (when check pending status is set) (2/2)

SQL statement and utility	Resource[#1]					
	Higher level -------------------- Lower level					
	Index	Index information file	Page	Row	Key value	Logical file
Deletion of all rows[#3]	--	--	--	--	--	--
Changing a table's definition (changing the storage partitioning conditions)	--	--	--	--	--	--
Database load utility (`pdload`)[#2]	--	--	--	--	--	--
Database reorganization utility (`pdrorg`)[#2]	--	--	--	--	--	--
Database structure modification utility (`pdmod`)	--	--	--	--	--	--
Integrity check utility (`pdconstck`)[#2]	--	--	--	--	--	--
Reflection command for online reorganization (`pdorend`)[#2]	--	--	--	--	--	--

--: Locking is not applied.

#1

This table shows resources for tables in which a referential constraint or a check constraint is defined.

#2

When a HiRDB/Parallel Server is used, locking equivalent to a `LOCK` statement with EXCLUSIVE specified is applied to all back-end servers when the utility is executed for a shared table. For details about the lock mode applied during execution of the LOCK statement on a shared table, see the EXCLUSIVE *specified* rows under LOCK *statement* in *Tables 3-10* to *3-15*.

#3

If USE or nothing is specified in the pd_check_pending operand in the system definition, the data dictionary table (resource type: 3005, type name: DICT) is locked temporarily in the EX mode. The data dictionary RDAREA (resource type: 0001, type name: RDAR) is locked in the SU mode until the transaction terminates.

(4) Lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE

Tables 3-18 to *3-21* show the lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE.

Table 3-18: Lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE (when an index is not defined) (1/2)

SQL statement and execution environment		**Resource**						
		Higher level --------------------- Lower level						
		Inner replica config#	**Replica group config#**	**RDAREA**			**Table**	**Table (NO WAIT search)**
				For tables	**For indexes**	**Last HiRDB file**		
Retrieval	NOWAIT specified	B	B	B	--	--	--	B
	WITH SHARE specified	B	B	B	--	--	B	--
	WITH EXCLUSIVE specified#	B	B	B	--	--	B	--
	FOR UPDATE clause specified#	B	B	B	--	--	B	--
	None of the above	B	B	B	--	--	B	--
Updating		B	B	B	--	--	B	--

SQL statement and execution environment		Resource						
		Higher level -------------------- Lower level						
		Inner replica config#	Replica group config#	RDAREA			Table	Table (NO WAIT search)
				For tables	For indexes	Last HiRDB file		
Addition		B	B	B	--	--	B	--
Deletion		B	B	B	--	--	B	--
`LOCK` statement	`SHARE` specified	B	B	B	--	--	B	--
	`EXCLUSIVE` specified	B	B	B	--	--	B	--
Table deletion		--	--	B	--	--	B	B
Index	Definition	--	--	B	--	--	B	B
	Deletion	--	--	B	--	--	B	B
Deletion of all rows		B	B	B	--	--	B	B
Table definition change		--	--	--	--	--	B	B

--: Locking is not applied or is not applicable (page locking cannot be specified).

B: Lock is not released when the SQL statement is executed.

#: If the inner replica facility is being used, the inner replica configuration management is locked. If the updatable online reorganization is being used, the inner replica configuration management or the replica group configuration management is locked.

Table 3-19: Lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE (when an index is not defined) (2/2)

SQL statement and execution environment		Resource					
		Higher level -------------------- Lower level					
		Index	Index information file	Page	Row	Key value	Logical file
Retrieval	NOWAIT specified	--	--	--	--	--	--
	WITH SHARE specified	--	--	--	B	--	--
	WITH EXCLUSIVE specified	--	--	--	B	--	--
	FOR UPDATE clause specified	--	--	--	B	--	--
	None of the above	--	--	--	B	--	--
Updating		--	--	--	R	--	--
Addition		--	--	--	R	--	--
Deletion		--	--	--	R	--	--
LOCK statement	SHARE specified	--	--	--	--	--	--
	EXCLUSIVE specified	--	--	--	--	--	--
Table deletion		--	--	--	--	--	--
Index	Definition	--	--	--	--	--	--
	Deletion	--	--	--	--	--	--
Deletion of all rows		--	--	--	--	--	--
Table definition change		--	--	--	--	--	--

--: Locking is not applied or is not applicable (page locking cannot be specified).

R: Lock is released when the SQL statement is executed.

B: Lock is not released when the SQL statement is executed.

Table 3-20: Lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE (when an index is defined) (1/2)

SQL statement		**Resource**						
		Higher level -------------------- Lower level						
		Inner replica config#	**Replica group config#**	**RDAREA**			**Table**	**Table (NO WAIT search)**
				For tables	**For indexes**	**Last HiRDB file**		
Retrieval	`NOWAIT` specified	B	B	B	B	--	--	B
	`WITH SHARE` specified	B	B	B	B	--	B	--
	`WITH EXCLUSIVE` specified	B	B	B	B	--	B	--
	`FOR UPDATE` clause specified	B	B	B	B	--	B	--
	None of the above	B	B	B	B	--	B	--
Updating		B	B	B	B	--	B	--
Addition		B	B	B	B	--	B	--
Deletion		B	B	B	B	--	B	--
`LOCK` statement	`SHARE` specified	B	B	B	B	--	B	--
	`EXCLUSIVE` specified	B	B	B	B	--	B	--
Table deletion		--	--	B	B	--	B	B
Index	Definition	--	--	B	B	--	B	B
	Deletion	--	--	B	B	--	B	B
Deletion of all rows		B	B	B	B	--	B	B
Table definition change		--	--	--	B	--	B	B

Legend:

--: Locking is not applied or is not applicable (page locking cannot be specified).

B: Lock is not released when the SQL statement is executed.

#: If the inner replica facility is being used, the inner replica configuration management is locked. If the updatable online reorganization is being used, the inner replica configuration management or the replica group configuration management is locked.

Table 3-21: Lock release timings in tables for which the WITHOUT ROLLBACK option is specified in CREATE TABLE (when an index is defined) (2/2)

SQL statement		**Resource**					
		Higher level -------------------- Lower level					
		Index	**Index information file**	**Page**	**Row**	**Key value**	**Logical file**
Retrieval	`NOWAIT` specified	--	--	--	--	--	--
	`WITH SHARE` specified	--	--	--	B	--[#1]	--
	`WITH EXCLUSIVE` specified	--	--	--	B	--[#1]	--
	`FOR UPDATE` clause specified	--	--	--	B	--[#1]	--
	None of the above	--	--	--	B	--[#1]	--
Updating		--	--	--	R	--[#2]	--
Addition		--	--	--	R	--[#3]	--
Deletion		--	--	--	R	--[#3]	--
`LOCK` statement	`SHARE` specified	--	--	--	--	--	--
	`EXCLUSIVE` specified	--	--	--	--	--	--
Table deletion		--	--	--	--	--	--
Index	Definition	--	--	--	--	--	--
	Deletion	--	--	--	--	--	--
Deletion of all rows		--	--	--	--	--	--

SQL statement	Resource					
	Higher level -------------------- Lower level					
	Index	Index information file	Page	Row	Key value	Logical file
Table definition change	--	--	--	--	--	--

Legend:

--: Locking is not applied or is not applicable (index definition and page locking cannot be specified).

R: Lock is released when the SQL statement is executed.

B: Lock is not released when the SQL statement is executed.

#1: If the `pd_indexlock_mode` operand of the system definition is `KEY` (index locking is applied), the lock is released when the key value of the processed target is changed to another key value.

#2: The lock is released if the resource is a unique key index.

#3: The lock is not released if the `pd_indexlock_mode` operand of the system definition is `KEY` (index locking is applied).

3.4.3 Lock period

(1) Starting and releasing a lock

When a transaction locks a resource, that resource is usually occupied until `COMMIT` or `ROLLBACK` occurs. For example, because the EX mode is in effect while a locked resource (row or page) is being updated, all other transactions for the resource being updated must wait until `COMMIT` or `ROLLBACK` is executed. However, if the `UNTIL DISCONNECT` option is specified in the `LOCK` statement, the lock on the resource is retained until either the resource is disconnected or the transaction is committed after the table is deleted.

Normally, when row deletion is executed, the lock is maintained until the transaction is completed. However, because the row ends up being deleted from the database, retrieval processing in other transactions does not wait for lock-release in the row being deleted. For details about locking other transactions' retrieval processing, see *3.4.7 Locking uncommitted data to be deleted.*

(2) Referencing during a lock

Once a resource is locked, that resource is usually not released until `COMMIT` or `ROLLBACK` occurs. When an SQL statement with `WITHOUT LOCK` specified is used for retrieval, however, the lock is released from a locked resource (row or page) as soon as that resource has been referenced. When an SQL statement with WITHOUT LOCK NOWAIT specified is used for retrieval, even tables and rows that have been locked in EX mode by another transaction can be referenced as if they were not locked, except when a logical file is referenced. However, a table cannot be referenced if it is being accessed by the pdload or pdrorg command. For details about the pdload and pdrorg commands, see the *HiRDB Version 9 Command Reference* manual.

In retrieval using an SQL statement with WITHOUT LOCK NOWAIT specified, referencing is allowed even during updating. Therefore, care must be taken, because the result of the referencing might not be the same as the result after the updating.

3.4.4 Deadlocks and corrective measures

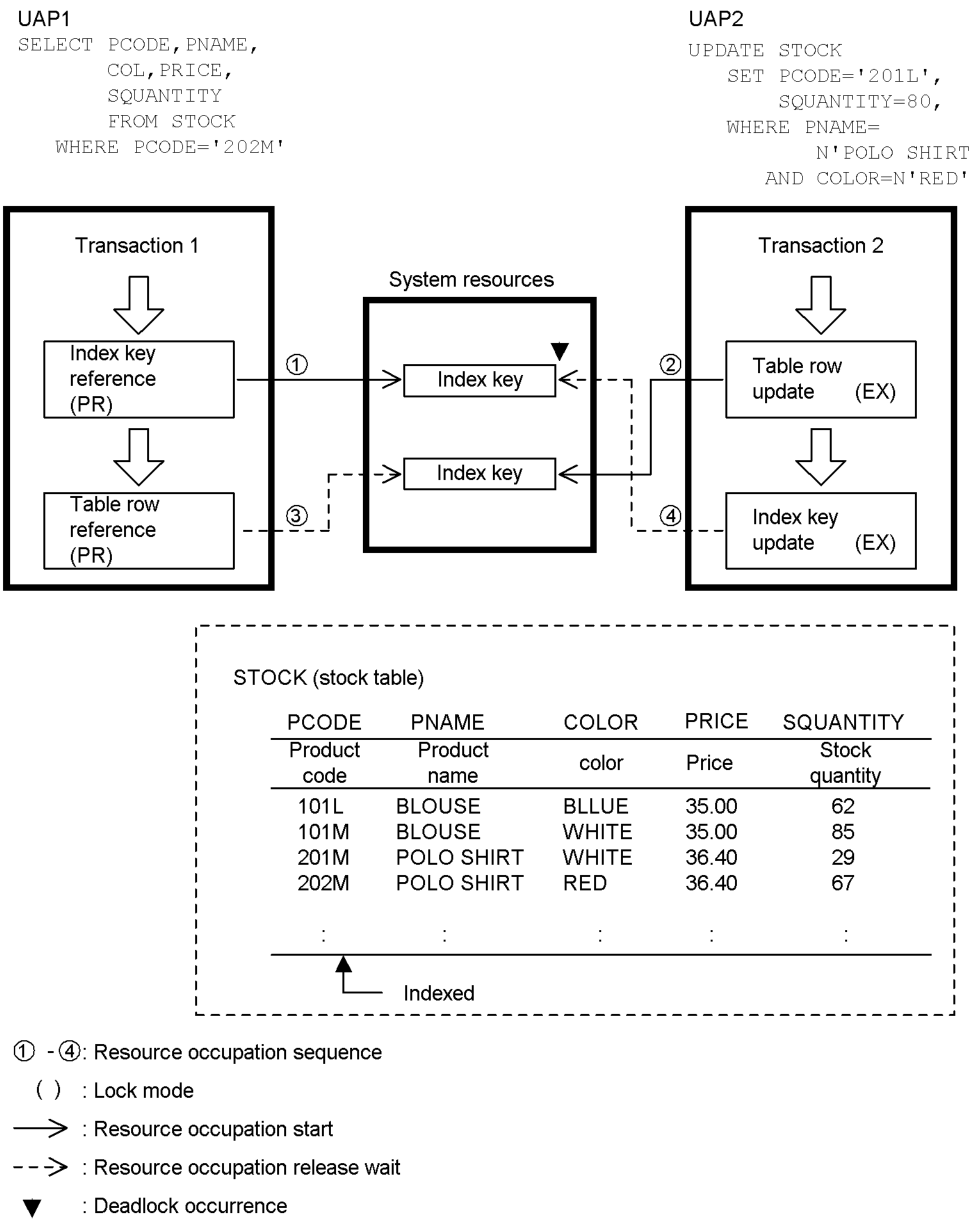
(1) Causes of deadlock

When two transactions attempt to access multiple resources but are waiting for the other to initiate a move, processing can become stuck; this is called *deadlock*.

Deadlock occurs most often between a referencing transaction and an updating (including deleting) transaction. It is, therefore, possible to reduce the frequency of deadlocks by changing the UAP access sequence.

The figure below shows an example of deadlock, in which two transactions simultaneously access a row with the same key. The figure also shows the relationship between the order in which locking is applied and deadlock.

Figure 3-8: Example of deadlock



UAP1

```
SELECT PCODE,PNAME,
       COL,PRICE,
       SQUANTITY
       FROM STOCK
   WHERE PCODE='202M'
```

UAP2

```
UPDATE STOCK
   SET PCODE='201L',
       SQUANTITY=80,
   WHERE PNAME=
          N'POLO SHIRT'
      AND COLOR=N'RED'
```

STOCK (stock table)

PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	color	Price	Stock quantity
101L	BLOUSE	BLLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
:	:	:	:	:

① - ④: Resource occupation sequence

( ) : Lock mode

⟶ : Resource occupation start

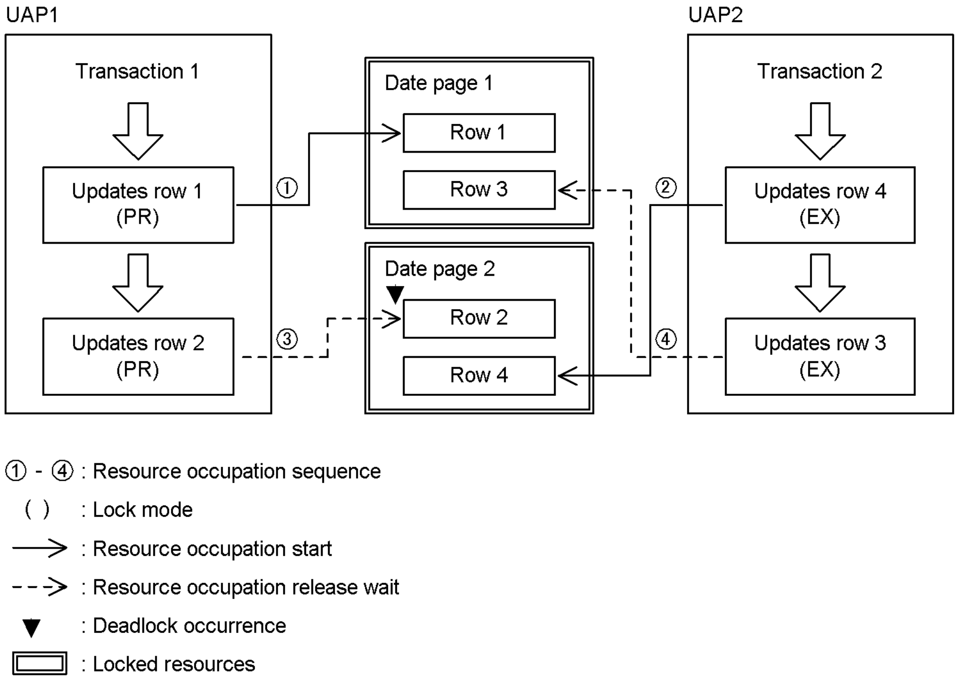
- - -> : Resource occupation release wait

▼ : Deadlock occurrence

With page-locking, there are situations in which deadlock cannot be prevented even though UAP access procedures are standardized.

The following figure shows an example of deadlock occurring in page-locking.

Figure 3-9: Example of deadlock in page-locking



In the example shown in *Figure 3-9*, the order in which rows are stored in a page cannot be standardized unless a cluster key is specified. Therefore, the sequence of UAP accesses cannot be standardized at the page level. In this case, `ALTER TABLE` must be used to change page locking to row locking in order to prevent deadlock from occurring.

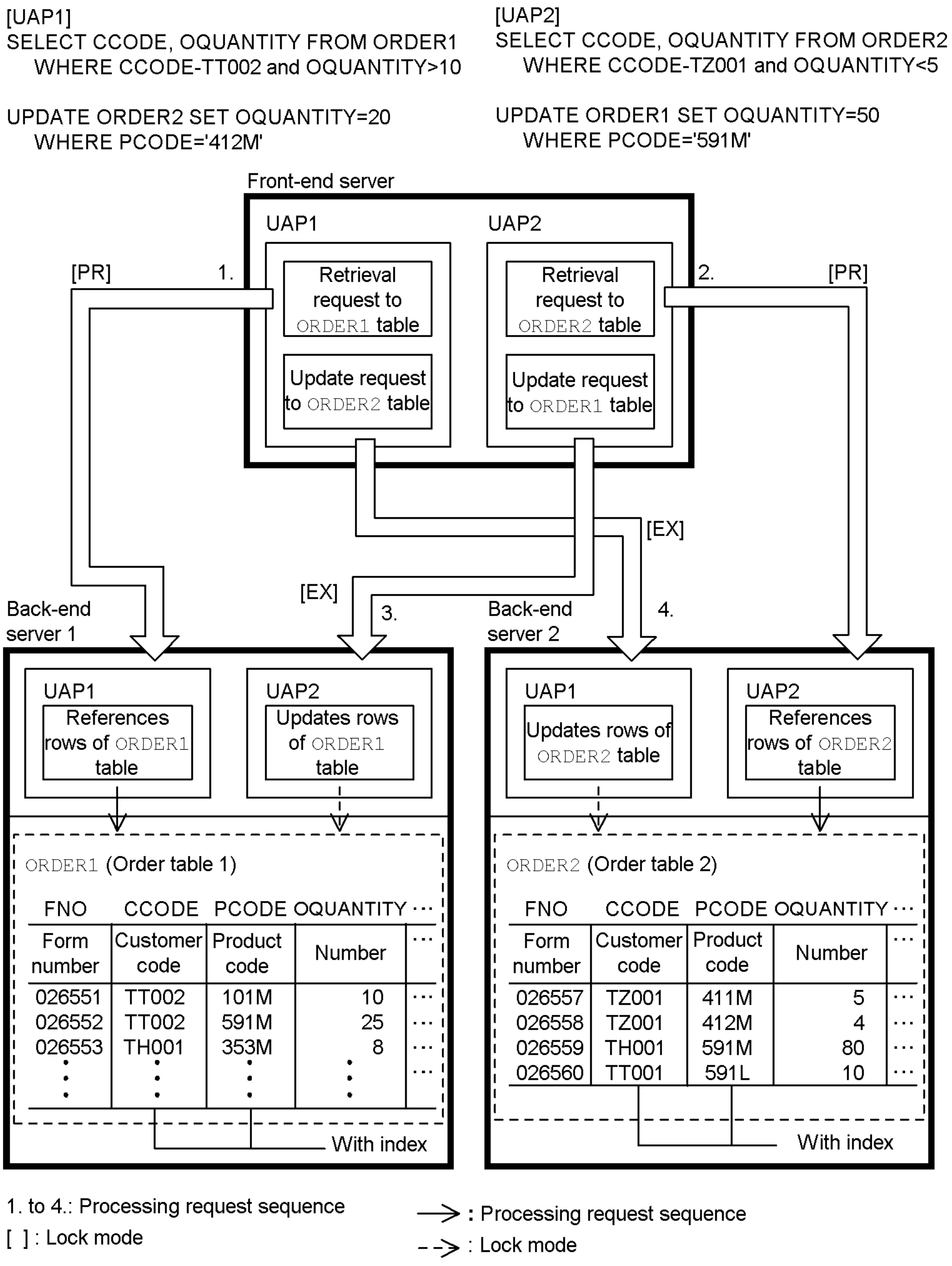
(2) Deadlock between servers

Deadlock can occur between servers as well as within a single server. In a HiRDB/ Parallel Server, deadlock between servers is called *global deadlock*.

Global deadlock occurs between a referencing transaction and an updating transaction in the same way that deadlock occurs within a single server, as shown in *Figure 3-9*. The frequency of these deadlocks can be reduced by changing the UAP access sequence.

The figure below shows an example of global deadlock, in which two transactions attempt to execute retrieval and updating in reverse order on tables stored on two separate servers. The figure also shows the relationship between the order in which locking is applied and deadlock.

Figure 3-10: Example of global deadlock



```
[UAP1]
SELECT CCODE, OQUANTITY FROM ORDER1
   WHERE CCODE-TT002 and OQUANTITY>10

UPDATE ORDER2 SET OQUANTITY=20
   WHERE PCODE='412M'
```

```
[UAP2]
SELECT CCODE, OQUANTITY FROM ORDER2
   WHERE CCODE-TZ001 and OQUANTITY<5

UPDATE ORDER1 SET OQUANTITY=50
   WHERE PCODE='591M'
```

Front-end server
UAP1
UAP2
Retrieval request to ORDER1 table
Update request to ORDER2 table
Retrieval request to ORDER2 table
Update request to ORDER1 table
[PR]
1.
2.
[PR]
[EX]
[EX]
3.
4.
Back-end server 1
Back-end server 2
UAP1
References rows of ORDER1 table
UAP2
Updates rows of ORDER1 table
UAP1
Updates rows of ORDER2 table
UAP2
References rows of ORDER2 table
With index
With index

ORDER1 (Order table 1)

FNO	CCODE	PCODE	OQUANTITY	···
Form number	Customer code	Product code	Number	···
026551	TT002	101M	10	···
026552	TT002	591M	25	···
026553	TH001	353M	8	···
:	:	:	:	···

ORDER2 (Order table 2)

FNO	CCODE	PCODE	OQUANTITY	···
Form number	Customer code	Product code	Number	···
026557	TZ001	411M	5	···
026558	TZ001	412M	4	···
026559	TH001	591M	80	···
026560	TT001	591L	10	···

1. to 4.: Processing request sequence
[ ] : Lock mode

→ : Processing request sequence
- -> : Lock mode

In this example, locking occurs between UAP1 and UAP2 within each back-end server; however, deadlock occurs at the front-end server because of lock-release wait.

(3) Detection of deadlocks

Deadlock in a unit is detected by the unit's locking mechanism. In HiRDB, for a resource locked by the same server within the unit, deadlocks among multiple transactions on that server are detected. However, deadlocks on resources among multiple servers in the same unit cannot be detected other than by HiRDB based on occurrence of a lock timeout. Deadlocks on resources among multiple units are detected based on a timeout in the same manner as with deadlocks among multiple servers in the same unit.

(a) Deadlock detection method and timing

The deadlock detection method and timing depend on whether lock processing is distributed. For details about distribution of lock processing, see the *HiRDB Version 9 System Operation Guide*. The following table describes the deadlock detection method and timing.

Table 3-22: Deadlock detection method and timing

pd_lck_deadlock_ check value	pd_lck_pool_pa rtition value#	Deadlock detection method	Deadlock detection timing
Y	2 or greater	A deadlock monitoring process is applied periodically to detect deadlock (interval monitoring method).	Deadlocks are detected at the interval specified in the pd_lck_deadlock_check_interval operand. In such a case, there is a delay between occurrence of a deadlock and its detection.
	1	Deadlock is detected when a server process is placed in lock-release wait status (immediate detection method).	Deadlocks are detected as soon as they occur.
N	--	Deadlock is not detected. A UAP is placed in lock-release wait status for the amount of time specified in the pd_lck_wait_timeout operand and then a lock timeout error is issued to the UAP.	--

Legend:

--: Not applicable

#

pd_fes_lck_pool_partition operand value for a front-end server.

(b) Disabling deadlock detection

You can disable deadlock detection by specifying N in the pd_lck_deadlock_check operand.

If you increase the number of pool partitions for locking when you are using the interval monitoring method for deadlock detection, locking performance will likely be affected adversely each time a deadlock is detected. Therefore, in an application system in which deadlocks do not occur, you might be able to improve SQL execution performance by disabling deadlock detection. If you are able to configure a deadlock-free application system, you can disable deadlock detection.

For an application system in which deadlocks might occur, you should not disable deadlock detection. If a deadlock occurs after deadlock detection has been disabled, SQL statements will not terminate with an error until the time specified in the pd_lck_wait_timeout operand elapses and a timeout occurs. Additionally, HiRDB will not output deadlock information, which will make it difficult to determine the cause of a deadlock.

(4) Deadlock countermeasures

The two major causes of deadlock are:

- UAP access sequence (order in which lock is applied)
- Retrieving and updating in reverse order

There are other types of deadlock in addition to those shown in *Figures 3-8*, *3-9*, and *3-10*. The following table shows the major types of deadlocks and the countermeasures for each type.

Table 3-23: Deadlocks and their countermeasures

Deadlocked resources	Cause	Countermeasures
Row and row	UAP access sequence (shown in *Figure 3-10*)	• Standardize UAP access sequence. • Use LOCK TABLE to lock the table. • Re-execute UAP after deadlock occurs.
Row and index key	Retrieval and updating in reverse order (shown in *Figure 3-8*)	• Do not update the retrieved row. • Do not update the values in a column to the same value. • Minimize index definition. • Use LOCK TABLE to lock the table. • Perform NOWAIT retrieval. • Re-execute UAP after deadlock occurs. • Apply non-locking of index key values

Deadlocked resources	Cause	Countermeasures
Index key and index key	UAP access sequence	• Standardize UAP access sequence. • Minimize index definition. • Use LOCK TABLE to lock the table. • Perform NOWAIT retrieval. • Re-execute UAP after deadlock occurs. • Apply non-locking of index key values
Page and page	Rows are stored in a page in an unpredictable sequence (shown in *Figure 3-9*)	• Use ALTER TABLE to change page locking to row locking.

(5) Locking based on deadlock priority values

Deadlock priority values can be used to control the transaction that is to be terminated with an error when a deadlock occurs. When deadlock priority control is specified in the pd_deadlock_priority_use operand in the system common definition, and deadlock priority values are specified in the PDDLKPRIO operand in the client environment definition, HiRDB determines the deadlock priority order of the transactions based on these specified values. Specification of a low value means a higher-processing priority; specification of a high value means that an error and rollback are more likely to occur. If two transactions have the same deadlock priority value, the error occurs for the transaction that was started later (this transaction is rolled back). If specification of the PDDLKPRIO operand is omitted, HiRDB triggers the error for the transaction that caused the deadlock (and rolls back this transaction), based on the type of UAP, the utility, and the operation command. For the default that is assumed when specification of the PDDLKPRIO operand is omitted, see *6.6.4 Environment definition information*. Unless the transaction is terminated by a ROLLBACK or DISCONNECT statement, a UAP that is rolled back implicitly because of deadlock results in an error, even if an SQL statement is issued. When an X/Open-compliant UAP is used as a client in an OLTP environment, a transaction must be terminated, even if deadlock occurs in the UAP.

To enable output of deadlock information if a deadlock occurs, specify Y in the pd_lck_deadlock_info operand of the system definitions. For details about the pd_lck_deadlock_info operand, see the *HiRDB Version 9 System Definition* manual.

(6) Preventing deadlocks

Although the frequency of deadlock occurrences can be reduced by enlarging the lock range, the concurrent execution capability of transactions declines. Conversely, while narrowing the lock range improves the concurrent execution capability, incorrect referencing and updating occurs, resulting in an increase in the deadlock frequency. To avoid deadlocks while maintaining concurrent execution capability, consider the measures listed as follows:

- Do not assign indexes to columns that are updated frequently
- Do not update retrieval conditions columns
- Specify the `FOR UPDATE OF` clause in the cursor definition only for columns to be updated
- Do not update values in a column (a unique index column in particular) to the same values (use the `SET` clause to specify only the columns that it is certain are updated to a different value)
- Specify the `FOR UPDATE` clause in the cursor specification to update or delete a row retrieved by the cursor
- Specify `WITH EXCLUSIVE LOCK` the columns that are updated after retrieval
- When assigning conditions to multiple columns, consider the use of a multicolumn index (to avoid expanding the retrieval range of single-column indexes)
- Consider the use of retrievals that use `WITHOUT LOCK NOWAIT`
- When accessing multiple tables, standardize the access sequence (if A is accessed before B, do not access A again; instead, save the value of A)
- Specify `LOCK TABLE`
- If a row must be updated immediately after being inserted with the `INSERT` statement, try to perform both steps in the same transaction
- To allow multiple UAPs to use multiple indexes with AND and update the same table simultaneously, specify `1` in the `pd_work_table_option` operand of the system definitions. For details about how to use multiple indexes with `AND`, see the explanation for `PDSQLOPTLVL` in *6.6.4 Environment definition information*. For details about the `pd_work_table_option` operand, see the *HiRDB Version 9 System Definition* manual.
- Apply non-locking of index key values.
- If you employ the snapshot method, create a UAP that can retry operations in the event of deadlock, because unexpected deadlocks might occur.

As explained above, both the lock range and the lock sequence must be evaluated to avoid deadlock. The lock sequence depends on the SQL statement and index types. For details, see Section *3.4.9 Lock sequence based on SQL statement and index types*.

(7) Avoiding deadlock in logical files used by plug-ins

If a plug-in uses logical files, use the EX mode to lock the files in logical file units for update manipulation, and use the PR mode for retrieval manipulation.

A logical file becomes locked while it is manipulated, regardless of the data value. Consequently, if an update transaction accesses a column that has a plug-in definition for using a logical file, logical file contention can occur between that transaction, and all other transactions that manipulate that column. To prevent contention, avoid executing any other programs while a program that updates columns with plug-in definitions for using logical files is being executed.

- ■ Deadlock avoidance measure 1

  Specify `LOCK TABLE.`

- ■ Deadlock avoidance measure 2

  If a logical file becomes a deadlocked resource, check whether the logical file is used for a data-type plug-in or for an index-type plug-in, and see *Table 3-10* through *Table 3-15*. For details about the deadlock information that is output if a deadlock occurs, see the *HiRDB Version 9 System Operation Guide*.

  Lock information:

  ```
  Type 000e -> logical file
  ```

  First four bytes of lock information -> RDAREA number

Investigate the RDAREA name from the RDAREA number.

If the RDAREA is used for storing abstract data LOB attributes, the data is treated as a row.

If the RDAREA is used for a plug-in index, the data is treated as an index key.

Notes

- For a plug-in index retrieval, the logical file is locked in PR mode even if `NOWAIT` is specified.
- Even if a `LOCK TABLE lock` is applied, the logical file is locked in EX or PR mode during data manipulation.
- If multiple columns with plug-in definitions that use logical files have been defined, *deadlock avoidance measure 2* cannot prevent deadlocks. Instead, use *deadlock avoidance measure 1*.

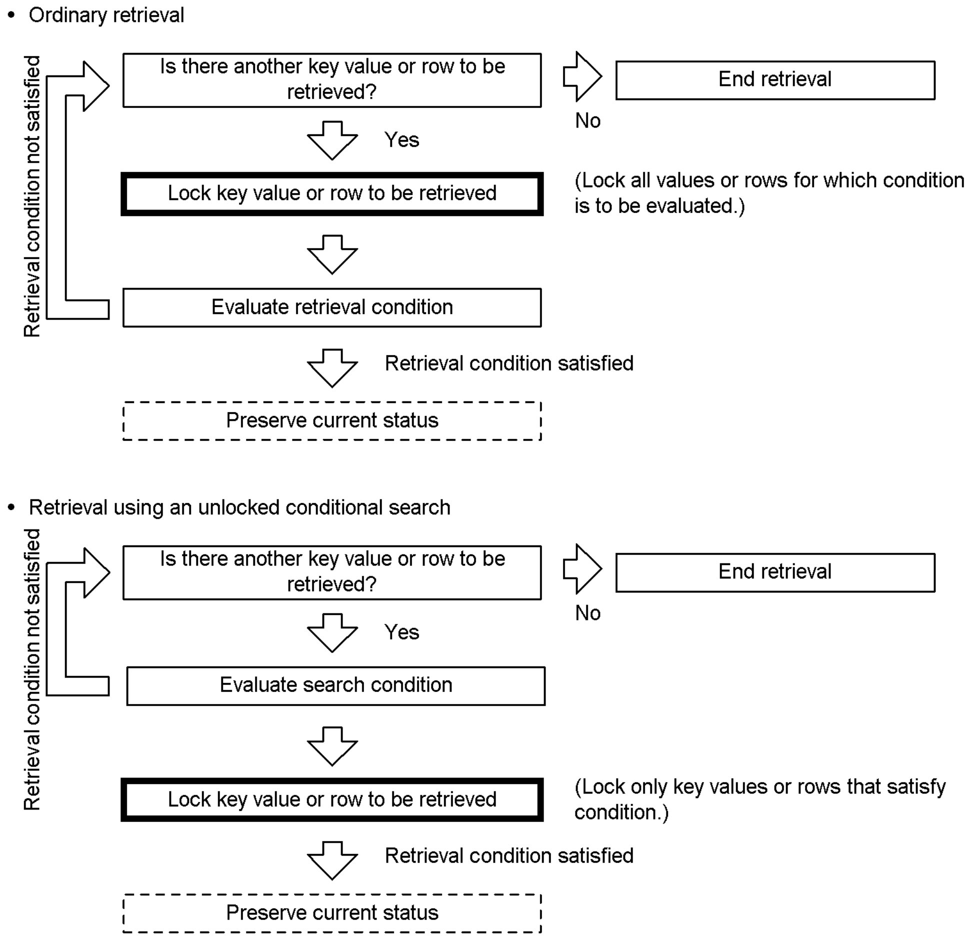
3.4.5 Unlocked conditional search

Unlocked conditional search does not lock items whenever retrieval processing is underway, but instead locks only those rows and key values that satisfy the retrieval condition. Unlocked conditional search can reduce the retrieval time compared to ordinary retrieval processing, because rows and key values that do not satisfy the retrieval condition are not locked.

Moreover, when updating and retrieval are executed simultaneously, it is unnecessary to wait for lock-release if another user is updating or adding rows that do not satisfy the condition. Consequently, the incidence of deadlocks and of lock-release timeouts is reduced.

The following figure shows the processing flows of an ordinary search and of a search using unlocked conditional search.

Figure 3-11: Processing flows of an ordinary retrieval and of a retrieval using an unlocked conditional search


• Ordinary retrieval
Retrieval condition not satisfied
Is there another key value or row to be retrieved?
No
End retrieval
Yes
Lock key value or row to be retrieved
(Lock all values or rows for which condition is to be evaluated.)
Evaluate retrieval condition
Retrieval condition satisfied
Preserve current status
• Retrieval using an unlocked conditional search
Retrieval condition not satisfied
Is there another key value or row to be retrieved?
No
End retrieval
Yes
Evaluate search condition
Lock key value or row to be retrieved
(Lock only key values or rows that satisfy condition.)
Retrieval condition satisfied
Preserve current status

An unlocked conditional search is used by specifying `YES` in the `PDLOCKSKIP` operand in the client environment definition.

An unlocked conditional search is effective under the following conditions:

- When the number of items that satisfy the condition is small compared to the number of items to which the conditional search is to be applied.

When a condition is selected for retrieval after the search range has been narrowed to some extent by means of an index key, only those items that satisfy the condition are locked. Consequently, if the number of items that satisfy the condition is small compared to the number of items within the range of the search, the number of lock processes is reduced (by number-of-items-that-satisfy-condition/number-of-items-in-search-range) compared to an ordinary retrieval.

- When retrieval does not use an index

  In the case of an ordinary retrieval that does not use an index, all rows are locked temporarily.

  If unlocked conditional search is used for a retrieval that does not use an index, only those items that satisfy the condition are locked, and therefore the number of lock processes is reduced (by number-of-items-that-satisfy-condition/total-number-of-rows-in-target-table).

- When retrieval is executed simultaneously with an updating process that does not satisfy the condition.

  Even if updating has already been performed within the retrieval range by another updating transaction, no lock-release waiting occurs if the updated results do not satisfy the condition.

Unlocked conditional search is not applied in the following cases even if it is specified:

- Retrieval that does not apply locking, such as `WITHOUT LOCK NOWAIT` (`0` is specified in `PDISLLVL` in the client environment definition and lock option is omitted)
- Retrieval that uses an index when non-locking of index keys is applied
- Retrieval under any of the following conditions and without using an index:
  - A system-defined scalar function is specified
  - A user-defined function is specified
  - A `VARCHAR`-type column with a defined length of 256 bytes or more is specified
  - A `MVARCHAR`-type column with a defined length of 256 bytes or more is specified
  - A `NVARCHAR`-type column with a defined length of 128 characters or more is specified
  - A `BINARY`-type column with a defined length of 256 bytes or more is included

- A BLOB-type column is specified
- A repetition column is specified
- A component specification is used
- A subquery containing an external reference is specified
- A qualified predicate or IN predicate is specified
- A search spanning two or more tables, except a condition that is evaluated as an inner table for nest-loop-join
- An index for use of AND multiple indexes (AND PLURAL INDEXES SCAN) is not used for evaluation
- An index for use of OR multiple indexes (OR PLURAL INDEXES SCAN) is not used for evaluation

Because unlocked conditional search does not involve locking, uncommitted data might be used for the conditional search. For example, if a conditional search is performed simultaneously with an updating transaction, the result of the conditional search might differ from the result of the update transaction processing.

3.4.6 Non-locking of index key values

Non-locking of index key values is when index key values are not locked. In this case, only the table data is locked.

When non-locking of index key values is applied, index key values cannot be locked during retrieval processing that uses an index. Also, in table update processing (row insertion, row deletion, or column value updating), index key values for the index defined in the update-target column cannot be locked.

(1) Application criteria

Non-locking of index key values should normally be applied.

However, the uniqueness constraint assurance operations, remaining entries for unique indexes, and the system log size that is output during table data update must be considered when deciding whether to apply non-locking of index key values. For details about the uniqueness constraint assurance operations and remaining entries for unique indexes, see *(4)(b) Remaining entries for unique indexes*. For details about the system log size that is output during table update, see the *HiRDB Version 9 Installation and Design Guide*.

(2) Specifying non-locking of index key values

To apply non-locking of index key values, specify NONE in the system definition for the pd_indexlock_mode operand. For details about the pd_indexlock_mode operand, see the *HiRDB Version 9 System Definition* manual.

If the value specified for the `pd_inner_replica_control` operand in the system definition is greater than `1`, `NONE` is assumed for the system definition's `pd_indexlock_mode` operand regardless of the actual specification of the `pd_indexlock_mode` operand.

(3) Example of deadlock avoidance

Deadlocks like the one shown in *Figure 3-8* can be avoided by applying non-locking of index key values. The following figure shows an example of deadlock avoidance by applying non-locking of index key values.

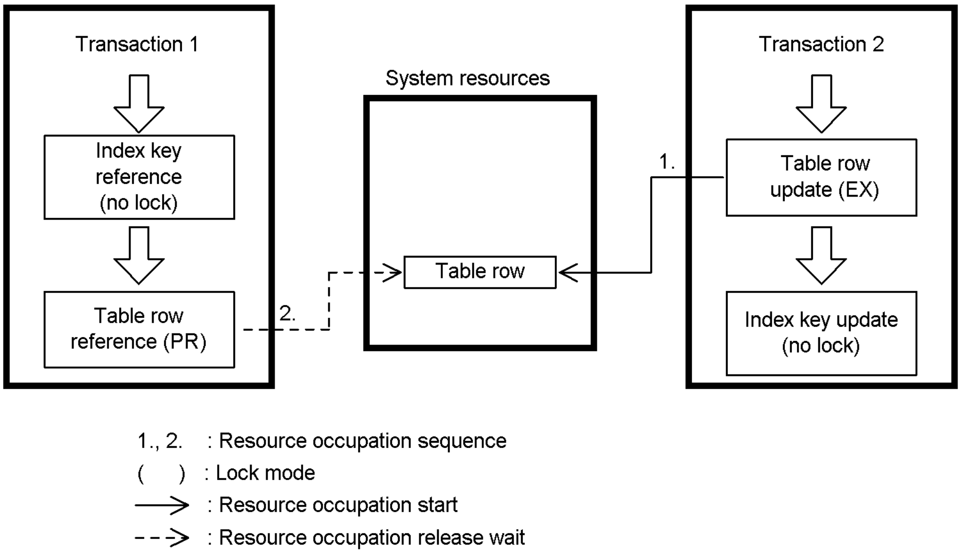
Figure 3-12: Example of deadlock avoidance by applying non-locking of index key values

<UAP1>

```
SELECT PCODE,PNAME,
       COL,PRICE,SQUANTITY
           FROM STOCK
   WHERE PCODE='202M'
```

<UAP2>

```
UPDATE STOCK
   SET PCODE='201L',
       SQUANTITY=80
WHERE PNAME=
     N'POLO SHIRT'
     AND COLOR=N'RED'
```


Transaction 1
Index key reference (no lock)
Table row reference (PR)
2.
System resources
Table row
1.
Transaction 2
Table row update (EX)
Index key update (no lock)
1., 2. : Resource occupation sequence
( ) : Lock mode
: Resource occupation start
: Resource occupation release wait

(4) Notes

(a) Uniqueness constraint assurance operations for unique indexes

When non-locking of index key values is applied, the uniqueness constraint assurance operations in row addition and update are different from the operations for the index key value method (a method that does not use non-locking of index key values) in tables for which the uniqueness constraint is specified. These operational differences must be considered when non-locking of index key values is applied.

Uniqueness constraint assurance processing checks whether the data for keys to be added by using an index (unique index) is already in the table. This processing also guarantees the uniqueness of added keys.

In uniqueness constraint assurance processing, *if index key entries that have the same key are found, a uniqueness error occurs immediately*. Even if the other transaction operating that index key has the uncomplete status for transaction determination and rollback is possible, the HiRDB system indicates a uniqueness error immediately, without executing a lock check.

To continue processing instead of waiting for transaction settlement during insertion or update processing of table data for which the uniqueness constraint was specified, apply non-locking of index key values. To give priority to attempting insertion or update processing even if waiting is involved, also apply non-locking of index key values or lock the uncommitted data to be deleted by specifying `WAIT` in the `pd_lock_uncommited_delete_data` operand. For details about locking uncommitted data to be deleted, see *3.4.7 Locking uncommitted data to be deleted*.

(b) Remaining entries for unique indexes

Lock-release wait or deadlock may occur in a unique index when non-locking of index key values is applied.

When non-locking of index key values is applied to a unique index, the index key before `DELETE` or `UPDATE` statement execution is kept instead of being deleted so that the uniqueness constraint is assured. This remaining index key is called a *remaining entry*. Although this remaining entry is deleted at the appropriate timing after transaction determination, if an `INSERT` or `UPDATE` statement is executed for the same key as the remaining entry, an unexpectedly long wait period or deadlock may occur, depending on when the statement is executed.

To prevent these conditions, *create UAPs so that they do not update columns that have the uniqueness constraint*.

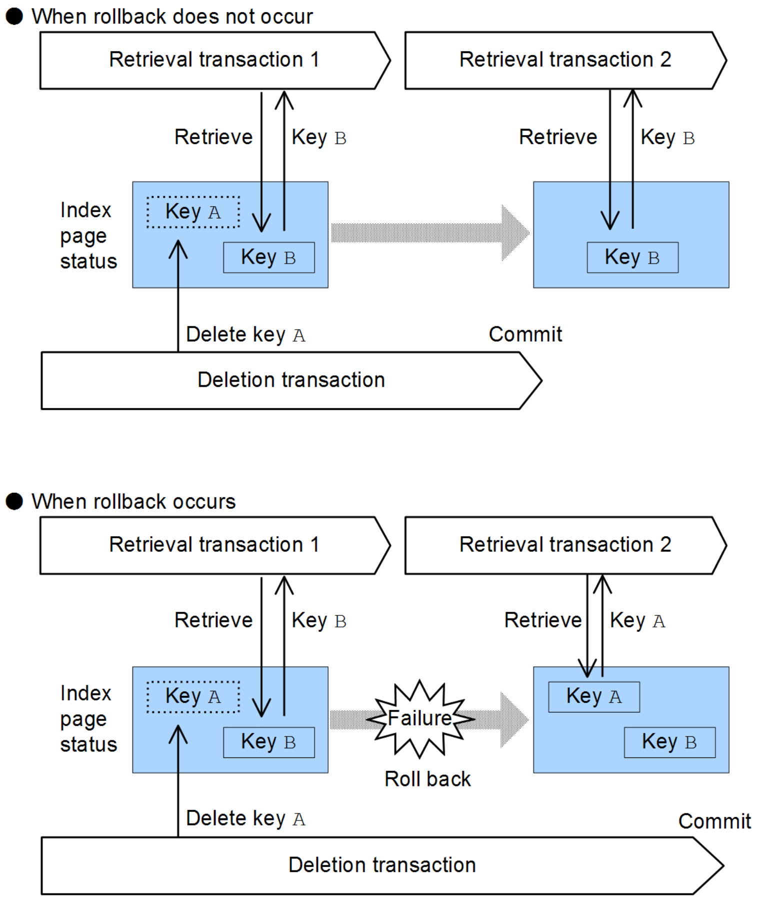
(c) Deadlocks that cannot be avoided even when non-locking of index key values is applied

Depending on the access sequence of the UAP, a deadlock may occur between index keys. To prevent this condition, create UAPs so that they do not update columns that have the uniqueness constraint.

3.4.7 Locking uncommitted data to be deleted

Normally, when an index key is deleted, it is removed from the database before the deletion transaction is committed. If the deletion transaction rolls back, the contents of the index key are recovered but the index key is excluded from searches by other retrieval transactions that are executing concurrently. The following figure shows an example of a deleted index key that is excluded from searches.

Figure 3-13: Deleted index key that is excluded from searches



Explanation:

Key `A` is removed from the database before the deletion transaction is committed. As a result, retrieval transaction 1 skips key `A` and uses key `B` for retrieval. After that, if rollback does not occur, retrieval transaction 2 also uses key `B` for retrieval in the same manner as retrieval transaction 1.

On the other hand, if rollback occurs and key `A` is recovered, retrieval transaction 2 uses key `A` for retrieval. Because retrieval transactions 1 and 2 use different index keys due to rollback, their retrieval results also differ.

The same applies to row deletion. Row data is not removed from the database until it is committed, but such row data is excluded as a retrieval target by other retrieval transactions that are executing concurrently.

By locking uncommitted data until the deletion transaction is committed, you can prevent uncommitted data from being excluded from being a retrieval target by other retrieval transactions that are executing concurrently.

(1) Criteria

For the following types of jobs, it is recommended that you lock uncommitted data that is to be deleted:

- A job in which the nature of one transaction's processing depends on the results of a preceding transaction's processing
- A job that will not re-execute in the event of rollback

(2) Specification method

You lock uncommitted data to be deleted by specifying `WAIT` in the `pd_lock_uncommited_delete_data` operand. For details about the `pd_lock_uncommited_delete_data` operand, see the manual *HiRDB Version 9 System Definition*.

(3) Effects of locking uncommitted data to be deleted

Effects of locking uncommitted data to be deleted are as follows:

- If a retrieval transaction detects uncommitted data to be deleted during retrieval processing, it waits until the deletion transaction is committed or rolled back. As a result, the uncommitted data to be deleted is not skipped by the retrieval transaction even when the deletion transaction rolls back.
- The uniqueness constraint is guaranteed in the same manner as with locking of index key values. This prevents detection of a uniqueness error in the event the deletion transaction rolls back.

(4) Processing of retrieval transactions when uncommitted data to be deleted is locked

The following table describes the processing of retrieval transactions for each SQL execution condition when uncommitted data to be deleted is locked, and when it is not locked.

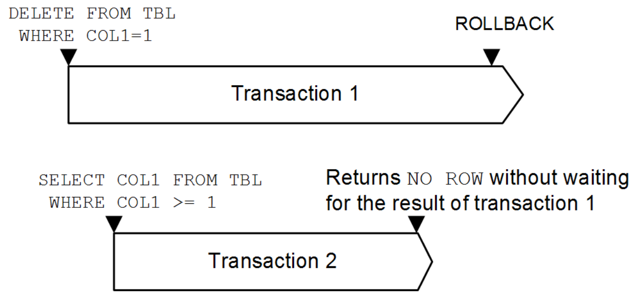
Table 3-24: Processing of retrieval transactions for each SQL execution condition

No.	SQL execution condition			Processing of retrieval transaction when rollback occurs on the UPDATE or DELETE statement	
	SQL statements executed concurrently	Nature of processing by UPDATE or DELETE statement	Nature of processing by SELECT statement	When data to be deleted is not locked	When data to be deleted is locked
1	`DELETE` and `SELECT` statements are executed concurrently.	`DELETE` statement for deleting rows	The keys to be deleted by the `DELETE` statement are included in the range of search conditions, or deleted rows are referenced.	Skips the deleted rows and keys without waiting for settlement of the transaction that executed the `DELETE` statement	Waits for settlement of the transaction that executed the `DELETE` statement and references the deleted rows and keys
2	`UPDATE` and `SELECT` statements are executed concurrently on a table for which indexes are defined.	`UPDATE` statement for updating indexes	The index keys to be updated by the `UPDATE` statement are included in the range of search conditions.	Skips non-updated keys without waiting for settlement of the transaction that executed the `UPDATE` statement	Waits for settlement of the transaction that executed the `UPDATE` statement and references the non-updated keys
3	`UPDATE` and `SELECT` statements are executed concurrently on a table for which multicolumn indexes are defined.	`UPDATE` statement that specifies some of the columns composing the multicolumn index in the search condition, and that updates other columns composing the index	The multicolumn index keys to be updated by the `UPDATE` statement are included in the range of search conditions.	Skips non-updated keys without waiting for settlement of the transaction that executed the `UPDATE` statement	Waits for settlement of the transaction that executed the `UPDATE` statement and references the non-updated keys

Example of No. 1:

This example executes `DELETE` and `SELECT` statements concurrently on the data `COL1=1` in the table `TBL` for which an index is defined for column `COL1`.

● When the data to be deleted is not locked

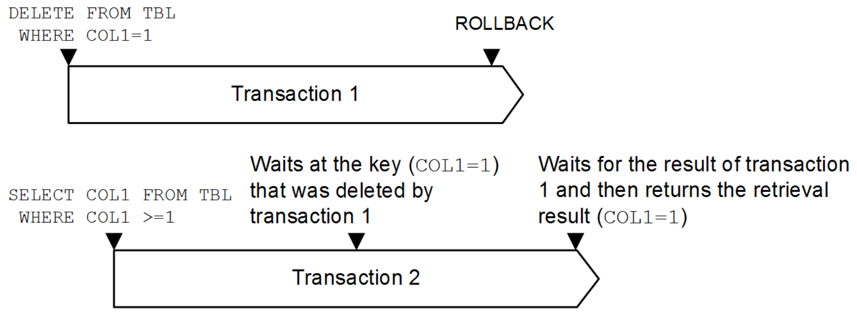


```
DELETE FROM TBL
 WHERE COL1=1
```

```
SELECT COL1 FROM TBL
 WHERE COL1 >= 1
```

Returns NO ROW without waiting for the result of transaction 1

● When the data to be deleted is locked



```
DELETE FROM TBL
 WHERE COL1=1
```

```
SELECT COL1 FROM TBL
 WHERE COL1 >=1
```

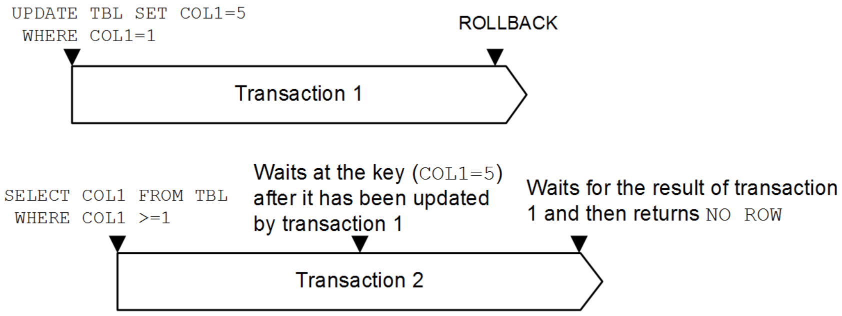
Waits at the key (COL1=1) that was deleted by transaction 1

Waits for the result of transaction 1 and then returns the retrieval result (COL1=1)

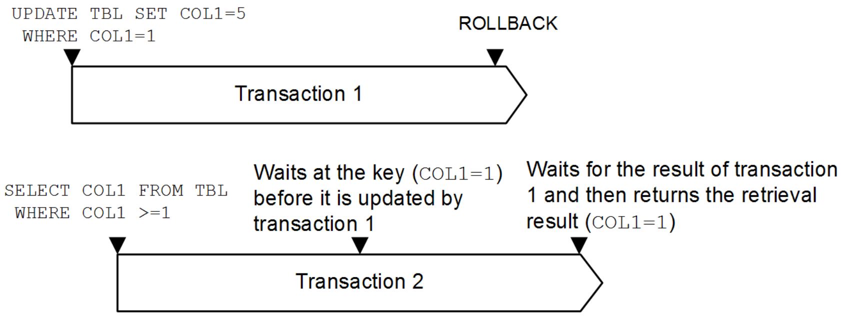
Example of No. 2:

This example executes UPDATE and SELECT statements concurrently on the data COL1=1 in the table TBL for which an index is defined for column COL1.

● When the data to be deleted is not locked


UPDATE TBL SET COL1=5
WHERE COL1=1
ROLLBACK
Transaction 1
SELECT COL1 FROM TBL
WHERE COL1 >=1
Waits at the key (COL1=5) after it has been updated by transaction 1
Waits for the result of transaction 1 and then returns NO ROW
Transaction 2

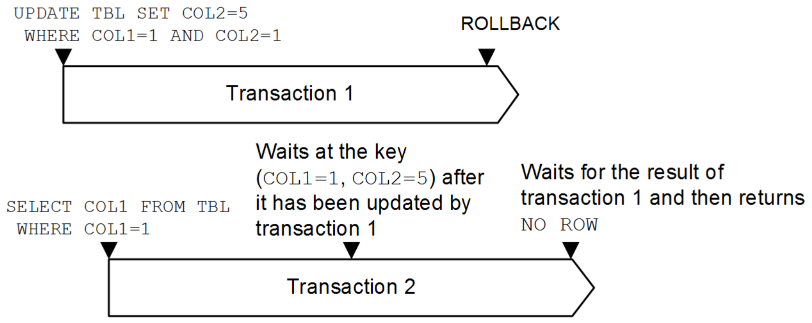
● When the data to be deleted is locked


UPDATE TBL SET COL1=5
WHERE COL1=1
ROLLBACK
Transaction 1
SELECT COL1 FROM TBL
WHERE COL1 >=1
Waits at the key (COL1=1) before it is updated by transaction 1
Waits for the result of transaction 1 and then returns the retrieval result (COL1=1)
Transaction 2

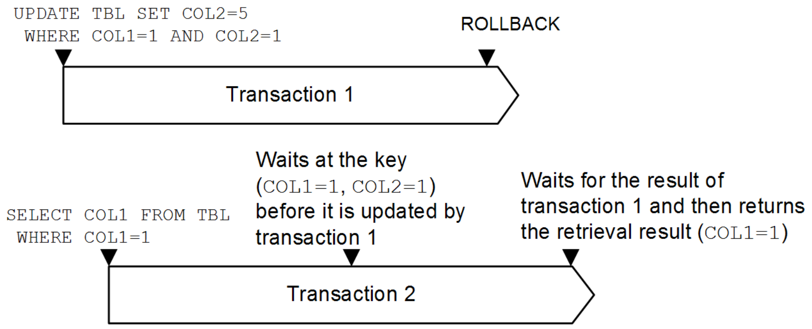
Example of No. 3:

This example executes UPDATE and SELECT statements concurrently on the data COL1=1 and COL2=1 in the table TBL for which a multicolumn index is defined for columns COL1 and COL2.

● When the data to be deleted is not locked


UPDATE TBL SET COL2=5
WHERE COL1=1 AND COL2=1
ROLLBACK
Transaction 1
SELECT COL1 FROM TBL
WHERE COL1=1
Waits at the key (COL1=1, COL2=5) after it has been updated by transaction 1
Waits for the result of transaction 1 and then returns NO ROW
Transaction 2

● When the data to be deleted is locked


UPDATE TBL SET COL2=5
WHERE COL1=1 AND COL2=1
ROLLBACK
Transaction 1
SELECT COL1 FROM TBL
WHERE COL1=1
Waits at the key (COL1=1, COL2=1) before it is updated by transaction 1
Waits for the result of transaction 1 and then returns the retrieval result (COL1=1)
Transaction 2

(5) Notes

(a) Remaining entries for indexes

If uncommitted data to be deleted is locked, index key entries existing before deletion or update processing remain in the index. As a result, null-value unique index keys and entries of a non-unique index remain in indexes. If lock-release wait status occurs for such remaining entries, deadlock might result. If a large number of entries remains in indexes, retrieval performance might be affected adversely.

Basically, the number of remaining entries increases each time an index key is updated or deleted. To avoid lock-release waits or deadlocks for remaining entries, you should delete remaining entries with the database reorganization utility (`pdrorg`) or the free page release utility (`pdreclaim`) whenever a certain amount of update or deletion processing has occurred. You can use the database condition analysis utility (`pddbst`) to determine the total number of remaining entries.

Remaining entries are deleted in the following cases:

- When an attempt is made to add indexes to a page containing remaining entries and the index page is split due to a shortage of free space in the page

- After a deletion transaction has been settled, the `pdreclaim` command with the `-x` option specified is executed for the corresponding index.
- After a deletion transaction has been settled, `pdrorg` is executed on the corresponding index or the table corresponding to the index.
- `PURGE TABLE` is executed on the table corresponding to a deleted index.
- An RDAREA containing a deleted index or a table corresponding to such an index is re-initialized.

Reducing the frequency of index update processing also helps avoid remaining entries.

(b) Remaining entries for tables

For areas storing table row data, entries also remain after rows are deleted. If uncommitted data to be deleted is locked, these remaining entries are also checked. This results in some overhead for skipping invalid data and locking.

Although unexpected locking does not occur on remaining entries for tables, unlike remaining entries for indexes, you should delete remaining entries by executing `pdrorg` or `pdreclaim` whenever a certain amount of update or deletion processing has occurred. You can use the database condition analysis utility (`pddbst`) to determine the total number of remaining entries.

Reducing the frequency of row deletion processing also helps avoid remaining entries.

3.4.8 Locking by UAPs

Although locking is controlled automatically by the HiRDB system, using the UAP to change the unit of locking sometimes reduces the locking overhead, resulting in better processing efficiency. Consider the items listed below when you design UAP:

(1) Search

1. If retrieval results will be referenced only once and the data need not be locked until `COMMIT` occurs, specify `WITHOUT LOCK` in the `SELECT` statement.

   When `WITHOUT LOCK` is specified, lock is released without waiting for transaction termination, thus resulting in better concurrent execution capability of transactions.

   Even when `WITHOUT LOCK NOWAIT` is specified, it is not possible to search a table while it is undergoing data processing by the database load utility (when `nowait=no` is specified in the `option` statement) or by the database reorganization utility (except when `-k unld` is specified).

   If `WITHOUT LOCK NOWAIT` is specified, a deadlock with `pdload` (when `nowait=yes` is specified in the `option` statement) might occur.

2. In cases other than the one above, lock the target table in the PR mode with the LOCK statement with SHARE specified.

   When a table is locked in advance with the LOCK statement, the overhead is reduced significantly because locking on a row or table basis does not occur. A lock buffer shortage can also be prevented.

3. When you search a shared table, we recommend that you specify WITHOUT LOCK or WITH ROLLBACK.

(2) Update

1. Before updating, lock the target table in the EX mode with the LOCK statement with EXCLUSIVE specified.

   When a table is locked in advance with the LOCK statement, the overhead is reduced significantly because locking on a row basis does not occur. A lock buffer shortage can also be prevented.

2. When updating a shared table (including addition and deletion) where the key value of an index is changed, or when updating a large section of a shared table, always lock the shared table with an EXCLUSIVE-specified LOCK statement. Note that when a shared table is locked with an EXCLUSIVE-specified LOCK statement, the RDAREA for indexes (shared RDAREA), which stores indexes defined for the shared table, is also locked.

(3) Deletion

1. When dropping a table or an index or when deleting all rows, lock in the EX mode all segments being used for the target table.

   If many segments are being used for the table, all of those segments must be locked in the EX mode. To do this, have the transaction occupy all of the segments until the COMMIT statement is executed. Note that in this case, a large table for managing locked resources is required in the lock buffer. Care must be taken especially when one of the following statements is used to delete a table, its schema, its indexes, or all its rows:

   - DROP TABLE statement
   - DROP SCHEMA statement
   - DROP INDEX statement
   - PURGE TABLE statement

2. To delete a schema that applies to multiple tables, the individual tables should be deleted before the schema is deleted.

   This method uses less memory.

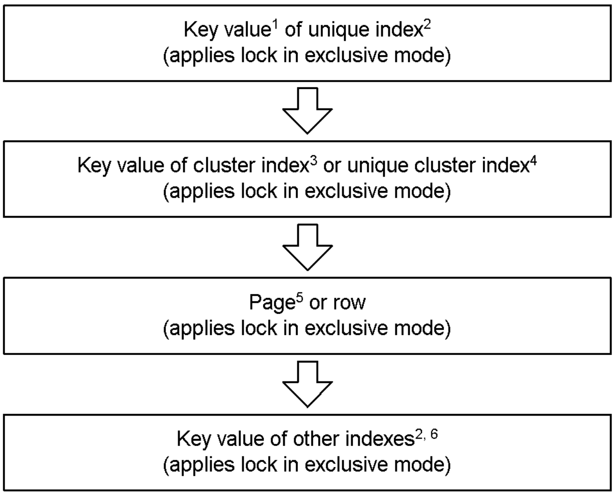
(4) Notes

1. When you lock a table with a LOCK statement, avoid simultaneous execution of other online transactions because those transactions remain in the wait status for a long time. However, no wait is involved for NOWAIT searches of unshared tables.
2. When a NOWAIT search is performed on a shared table, the shared table cannot be accessed if another user has executed an EXCLUSIVE-specified LOCK statement on that table.

3.4.9 Lock sequence based on SQL statement and index types

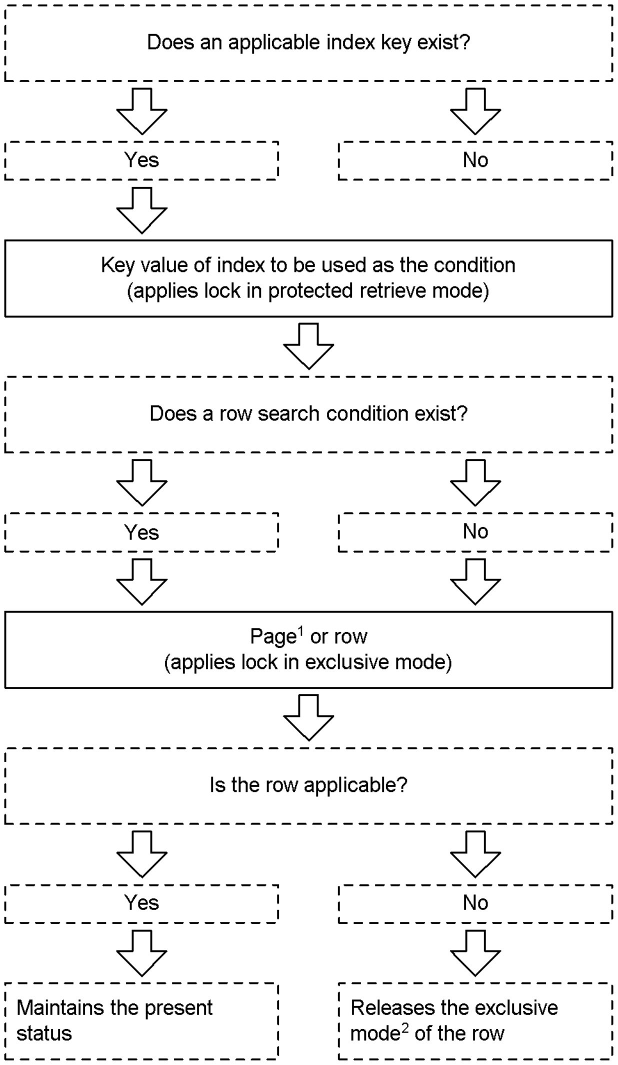
(1) Lock sequence of data manipulation SQL statements for index key values and data page rows

(a) INSERT statement


Key value[1] of unique index[2]
(applies lock in exclusive mode)
Key value of cluster index[3] or unique cluster index[4]
(applies lock in exclusive mode)
Page[5] or row
(applies lock in exclusive mode)
Key value of other indexes[2, 6]
(applies lock in exclusive mode)

[1] If multiple key values are defined, they are processed in the order opposite from the definition order.
[2] Index created for a column for which the uniqueness constraint (UNIQUE) is specified.
[3] Index created for a column for which a cluster key is specified.
[4] Index created for a cluster key column for which the uniqueness constraint (UNIQUE) is specified.
[5] Equivalent to page-locked table.
[6] Index other than a unique index, unique cluster index, or cluster index.

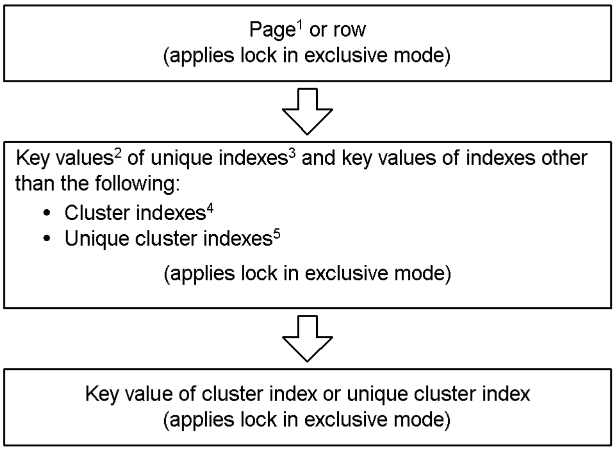
(b) DELETE statement that does not use a cursor or UPDATE statement to search for data matching a condition


Does an applicable index key exist?
Yes
No
Key value of index to be used as the condition
(applies lock in protected retrieve mode)
Does a row search condition exist?
Yes
No
Page[1] or row
(applies lock in exclusive mode)
Is the row applicable?
Yes
No
Maintains the present status
Releases the exclusive mode[2] of the row

[1] Equivalent to page-locked table.

[2] When the exclusive mode lock is applied to a row that has already been locked in the protected retrieve mode, the exclusive mode is not released, even when the processing in the exclusive mode terminates first. In this case, the exclusive mode is released only after both the processing in the protected retrieve mode and the processing in the exclusive mode have terminated.

(c) DELETE statement that uses a cursor



Page[1] or row
(applies lock in exclusive mode)

Key values[2] of unique indexes[3] and key values of indexes other than the following:

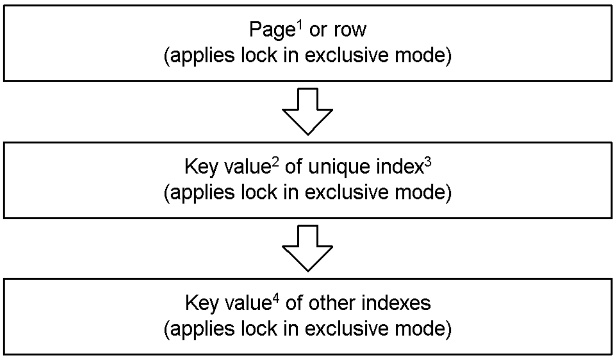
- Cluster indexes[4]
- Unique cluster indexes[5]

(applies lock in exclusive mode)

Key value of cluster index or unique cluster index
(applies lock in exclusive mode)

[1] Equivalent to page-locked table.
[2] If multiple key values are defined, they will be processed in the order opposite from the definition order.
[3] Index created for a column for which the uniqueness constraint (`UNIQUE`) is specified.
[4] Index created for a column for which a cluster key is specified.
[5] Index created for a cluster key column for which the uniqueness constraint (`UNIQUE`) is specified.

(d) UPDATE statement that uses a cursor



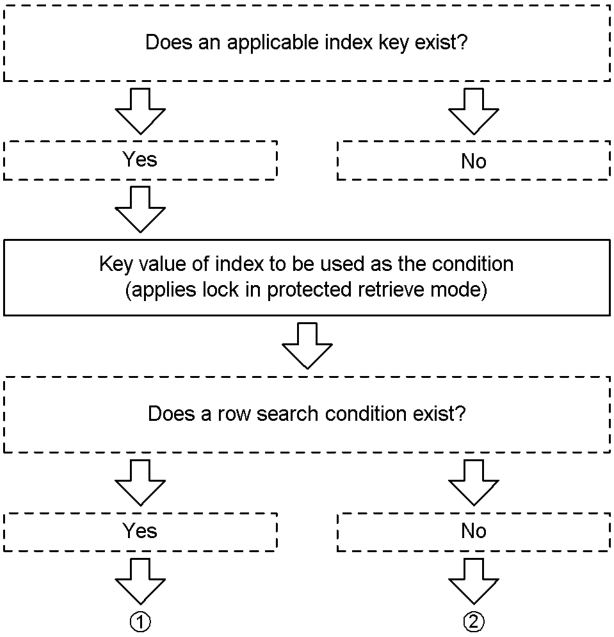
Page[1] or row
(applies lock in exclusive mode)

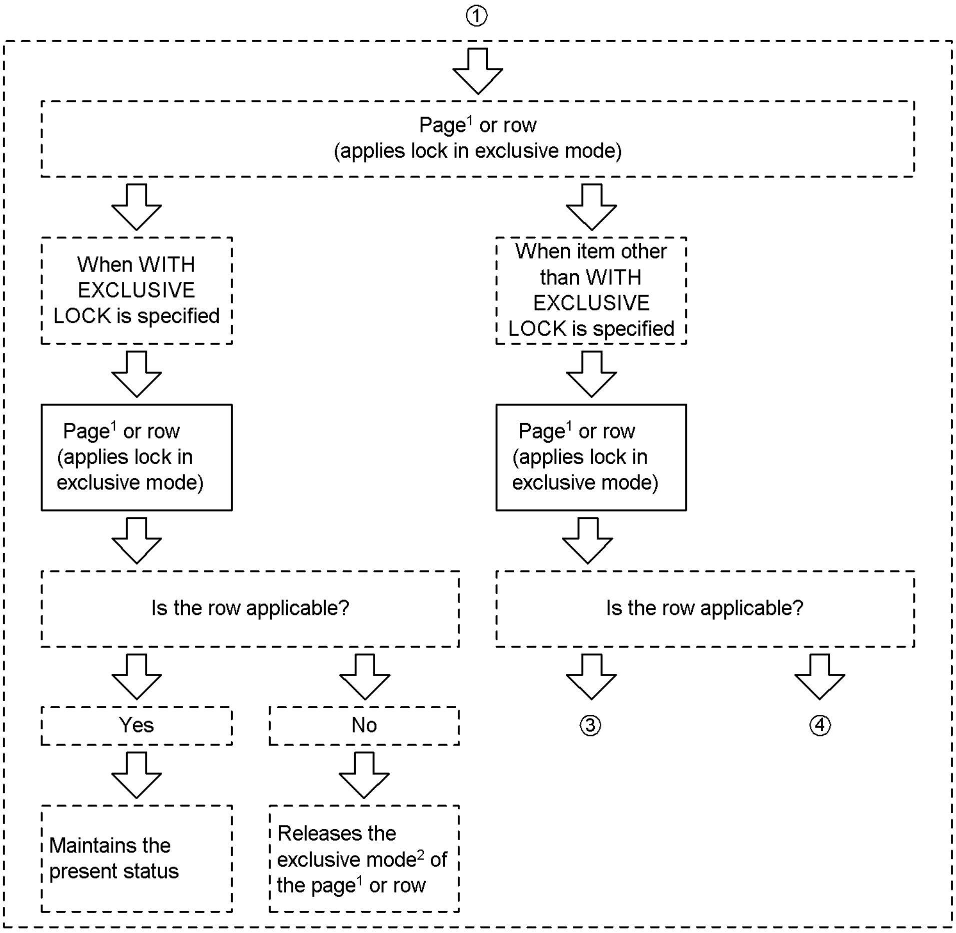
Key value[2] of unique index[3]
(applies lock in exclusive mode)

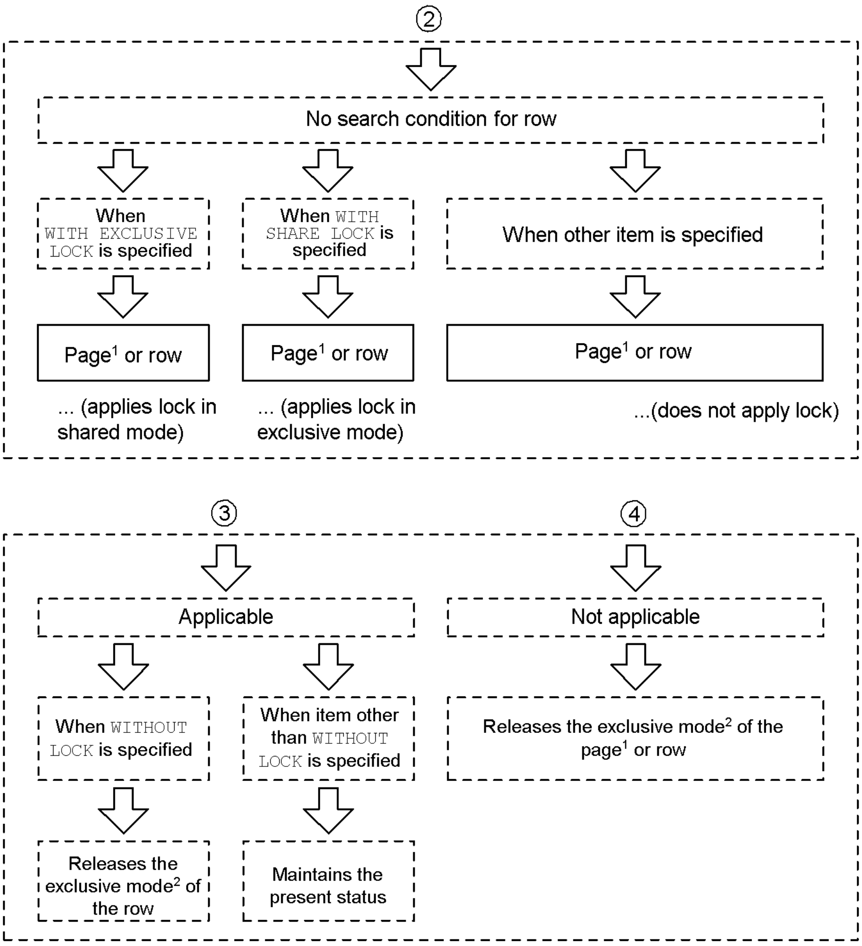
Key value[4] of other indexes
(applies lock in exclusive mode)

[1] Equivalent to page-locked table.
[2] If multiple key values are defined, they will be processed in the order opposite from the definition order.
[3] Index created for a column for which the uniqueness constraint (`UNIQUE`) is specified.
[4] Index other than a unique index, cluster index[5], or unique cluster index[6].
[5] Index created for a column for which a cluster key is specified.
[6] Index created for a cluster key column for which the uniqueness constraint (`UNIQUE`) is specified.

(e) SELECT or FETCH statement


Does an applicable index key exist?
Yes
No
Key value of index to be used as the condition
(applies lock in protected retrieve mode)
Does a row search condition exist?
Yes
No
①
②


①
Page1 or row
(applies lock in exclusive mode)
When WITH EXCLUSIVE LOCK is specified
When item other than WITH EXCLUSIVE LOCK is specified
Page1 or row (applies lock in exclusive mode)
Page1 or row (applies lock in exclusive mode)
Is the row applicable?
Is the row applicable?
Yes
No
③
④
Maintains the present status
Releases the exclusive mode2 of the page1 or row

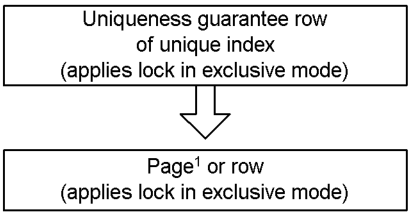

②
No search condition for row
When WITH EXCLUSIVE LOCK is specified
When WITH SHARE LOCK is specified
When other item is specified
Page1 or row
Page1 or row
Page1 or row
... (applies lock in shared mode)
... (applies lock in exclusive mode)
...(does not apply lock)
③
④
Applicable
Not applicable
When WITHOUT LOCK is specified
When item other than WITHOUT LOCK is specified
Releases the exclusive mode2 of the page1 or row
Releases the exclusive mode2 of the row
Maintains the present status

[1] Equivalent to page-locked table.

[2] When the exclusive mode lock is applied to a row that has already been locked in shared mode, the exclusive mode is not released, even when the exclusive mode terminates first. In this case, the exclusive mode is released only after both the processing in the shared mode and the processing in the exclusive mode have terminated.

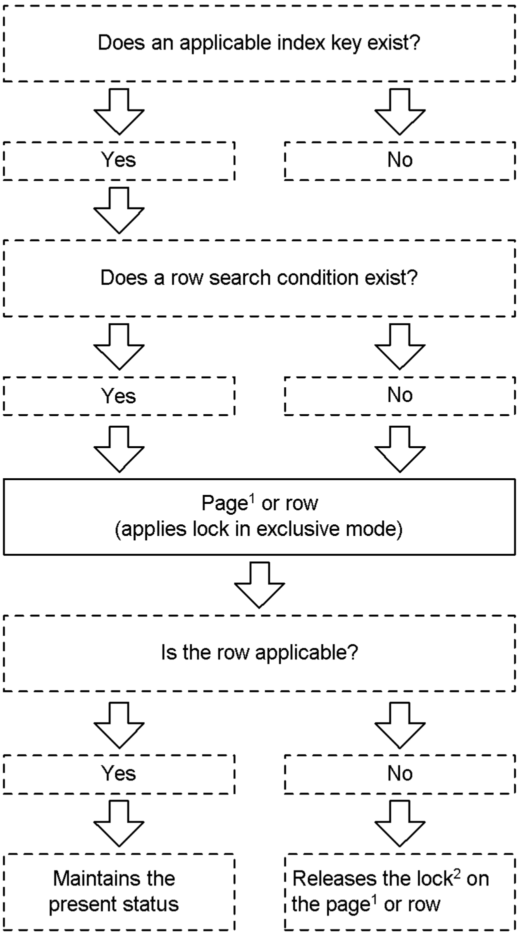
(2) Lock sequence of data manipulation SQL statements when non-locking of index key values is used

(a) INSERT statement


Uniqueness guarantee row
of unique index
(applies lock in exclusive mode)
Page[1] or row
(applies lock in exclusive mode)

[1] Equivalent to page-locked table.

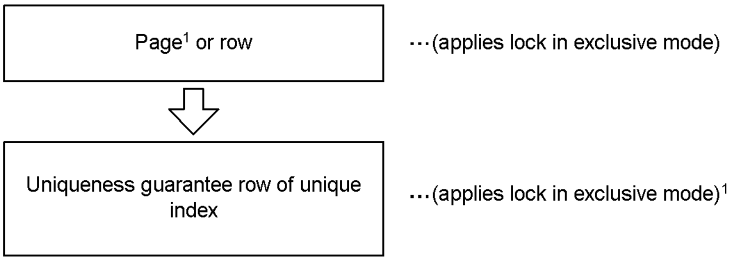
(b) DELETE statement that does not use a cursor or UPDATE statement to search for data matching a condition


Does an applicable index key exist?
Yes
No
Does a row search condition exist?
Yes
No
Page[1] or row
(applies lock in exclusive mode)
Is the row applicable?
Yes
No
Maintains the present status
Releases the lock[2] on the page[1] or row

[1] Equivalent to page-locked table.

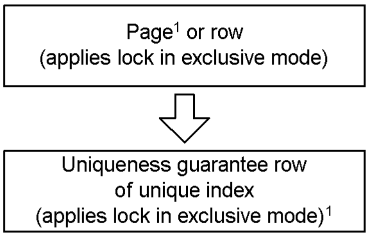
[2] When the exclusive mode lock is applied to a row that has already been locked in the protected retrieve mode, the exclusive mode lock is not released, even when the processing in the exclusive mode terminates first. In this case, the lock is released only after both the processing in the protected retrieve mode and the processing in the exclusive mode have terminated.

(c) DELETE statement that uses a cursor


Page[1] or row
…(applies lock in exclusive mode)
Uniqueness guarantee row of unique index
…(applies lock in exclusive mode)[1]

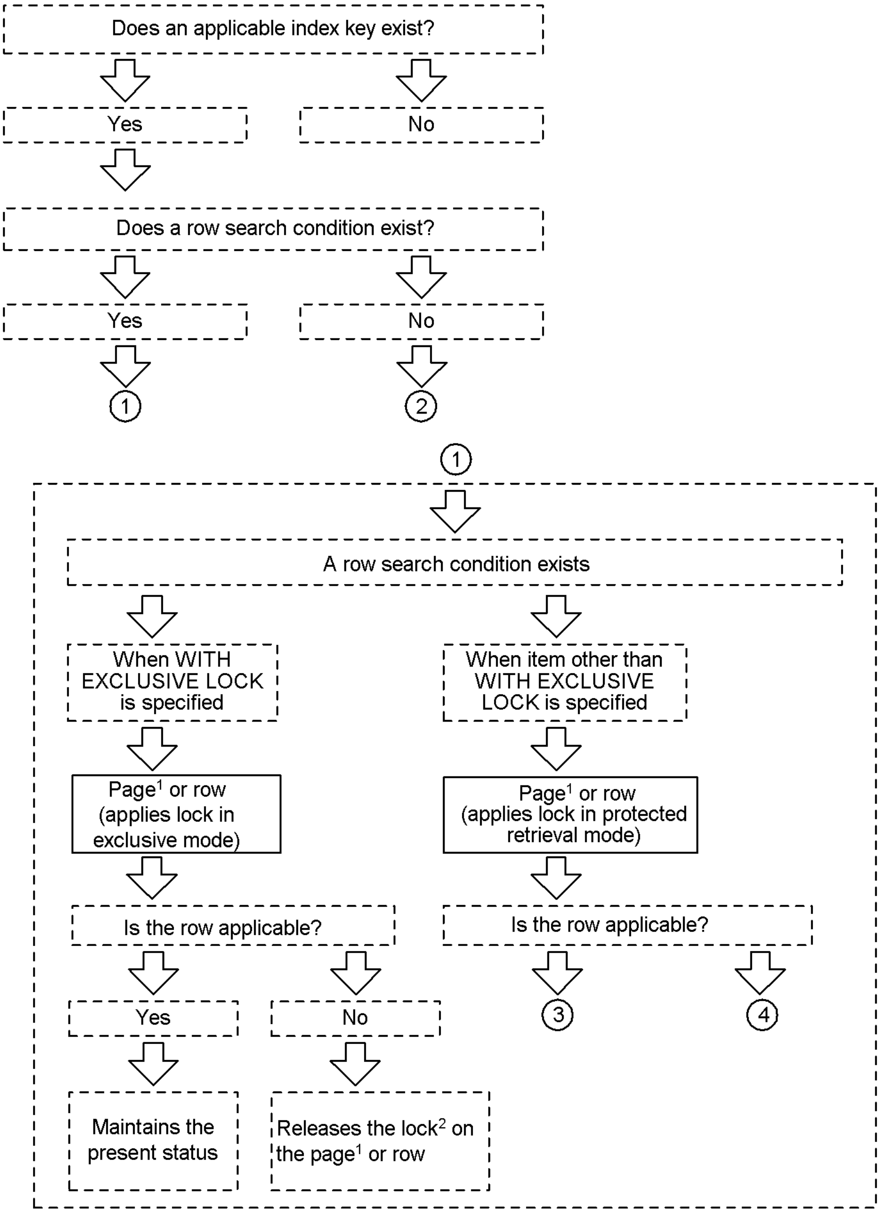
[1] Equivalent to page-locked table.

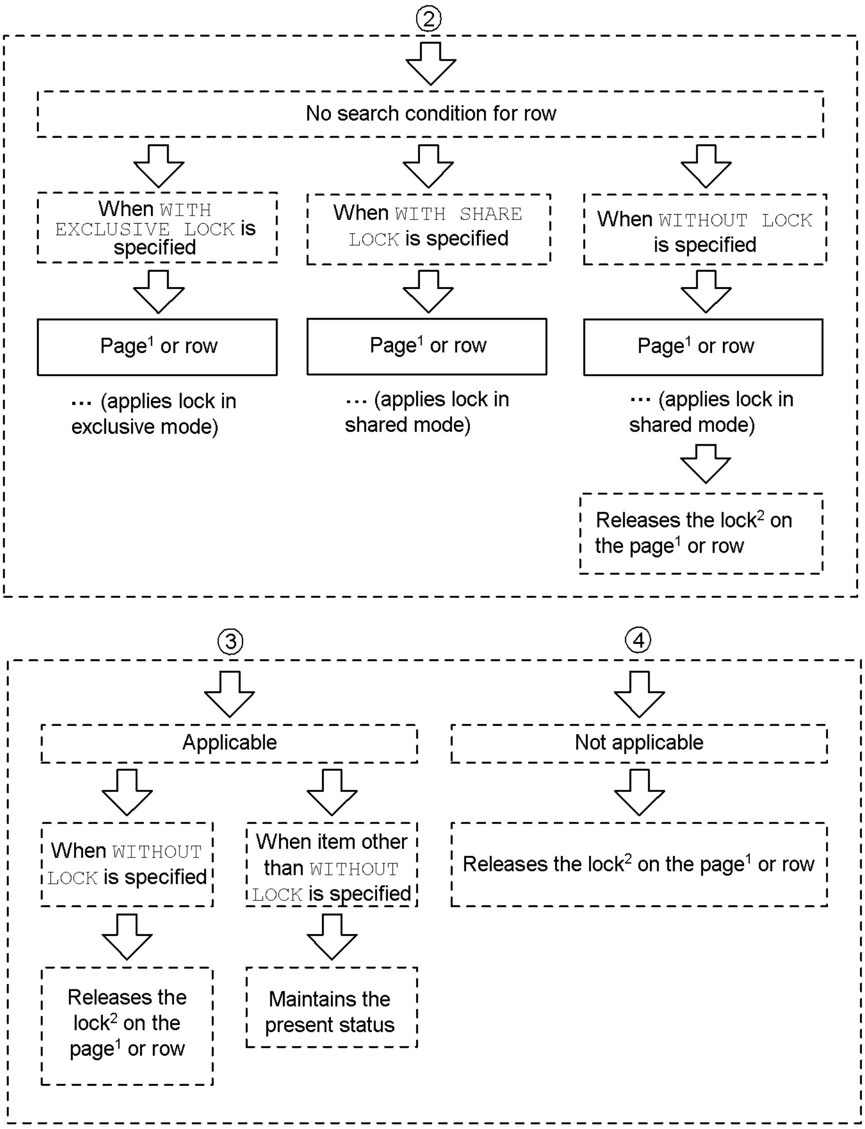
(d) UPDATE statement that uses a cursor


Page[1] or row
(applies lock in exclusive mode)
Uniqueness guarantee row
of unique index
(applies lock in exclusive mode)[1]

[1] Equivalent to page-locked table

(e) SELECT or FETCH statement


Does an applicable index key exist?
Yes
No
Does a row search condition exist?
Yes
No
1
2
1
A row search condition exists
When WITH EXCLUSIVE LOCK is specified
When item other than WITH EXCLUSIVE LOCK is specified
Page1 or row (applies lock in exclusive mode)
Page1 or row (applies lock in protected retrieval mode)
Is the row applicable?
Is the row applicable?
Yes
No
3
4
Maintains the present status
Releases the lock2 on the page1 or row

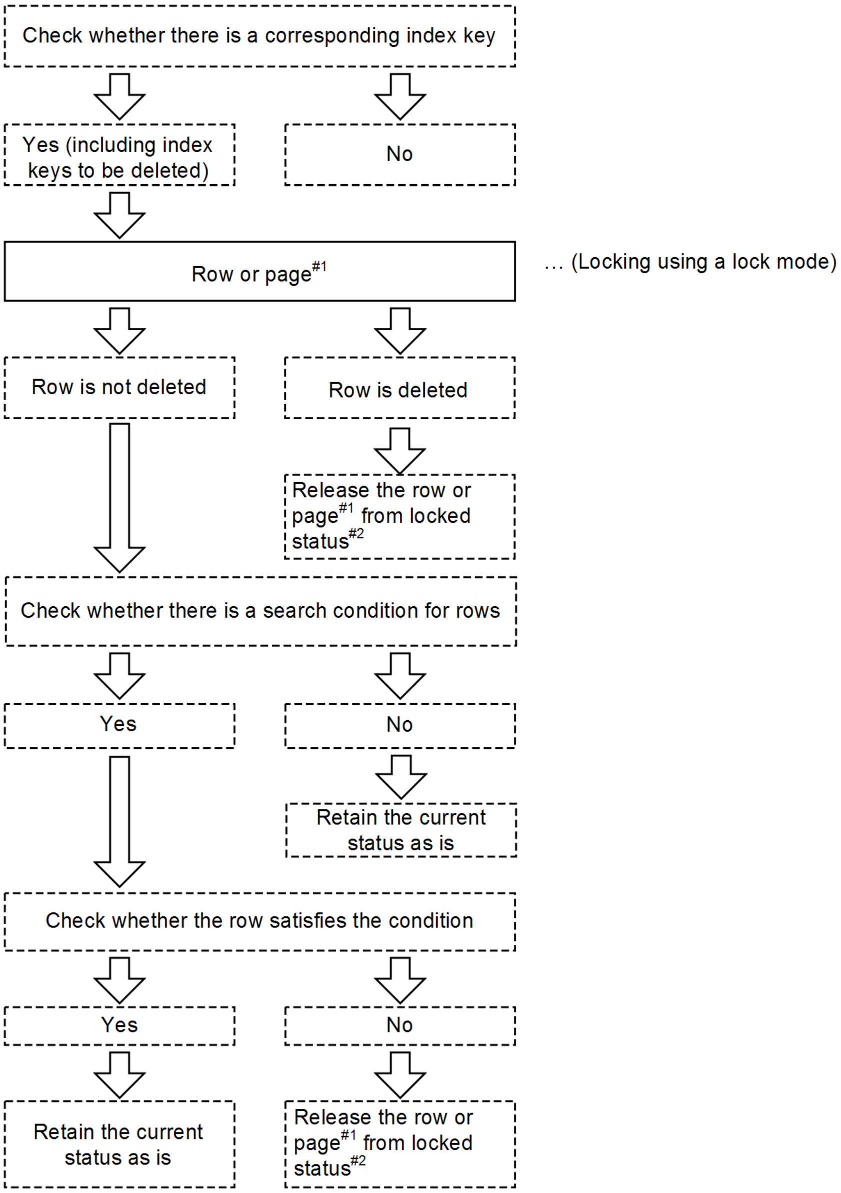

②
No search condition for row
When WITH EXCLUSIVE LOCK is specified
When WITH SHARE LOCK is specified
When WITHOUT LOCK is specified
Page1 or row
Page1 or row
Page1 or row
··· (applies lock in exclusive mode)
··· (applies lock in shared mode)
··· (applies lock in shared mode)
Releases the lock2 on the page1 or row
③
④
Applicable
Not applicable
When WITHOUT LOCK is specified
When item other than WITHOUT LOCK is specified
Releases the lock2 on the page1 or row
Releases the lock2 on the page1 or row
Maintains the present status

[1] Equivalent to page-locked table.

[2] When the exclusive mode lock is applied to a row that has already been locked in shared mode, the exclusive mode is not released, even when the exclusive mode terminates first. In this case, the exclusive mode is released only after both the processing in the shared mode and the processing in the exclusive mode have terminated.

(3) Order of locking uncommitted data to be deleted

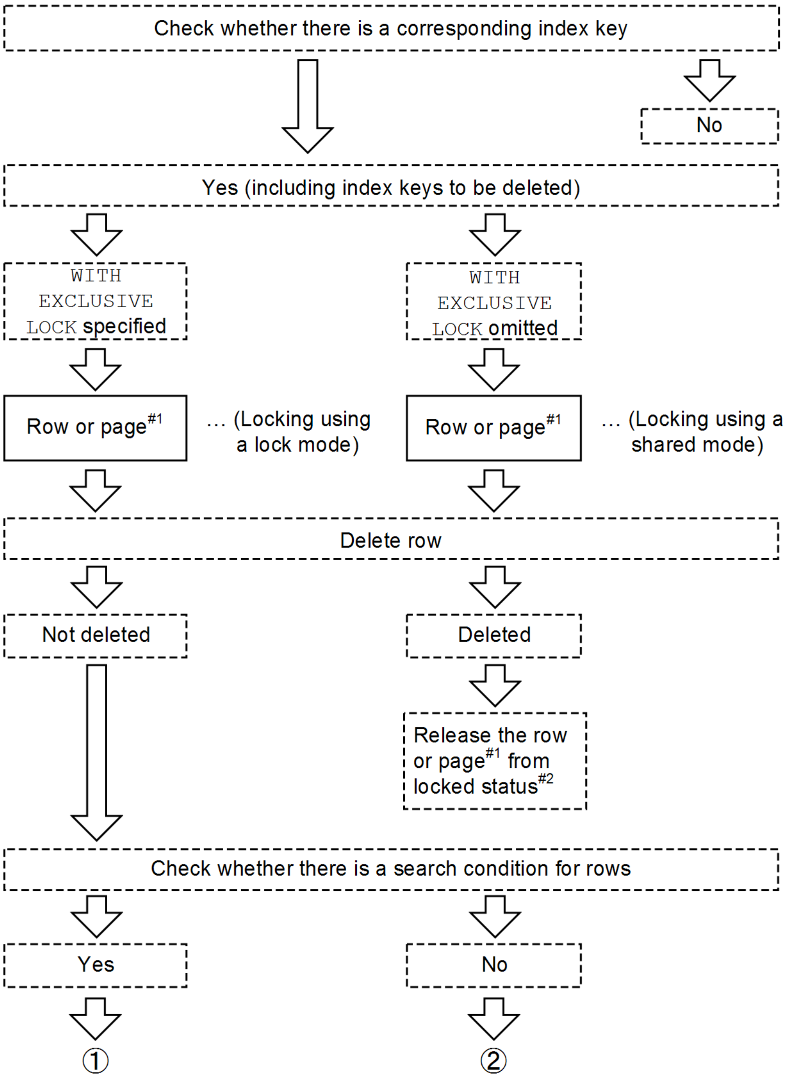
(a) Searching for the data that satisfies a condition by using the DELETE or UPDATE statement

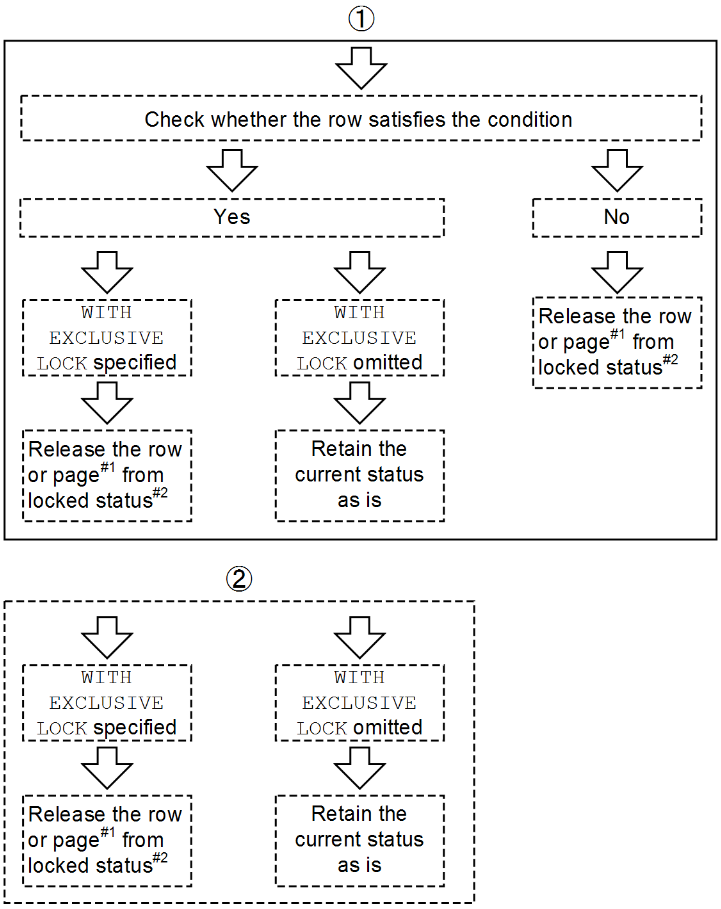

Check whether there is a corresponding index key
Yes (including index keys to be deleted)
No
Row or page#1
... (Locking using a lock mode)
Row is not deleted
Row is deleted
Release the row or page#1 from locked status#2
Check whether there is a search condition for rows
Yes
No
Retain the current status as is
Check whether the row satisfies the condition
Yes
No
Retain the current status as is
Release the row or page#1 from locked status#2

#1: For a table locked by page, the page is locked.

#2: If a lock mode is applied while a shared mode is in effect, the lock mode is not released even after the processing using the lock mode has terminated.

(b) For SELECT or FETCH statement


Check whether there is a corresponding index key
No
Yes (including index keys to be deleted)
WITH EXCLUSIVE LOCK specified
WITH EXCLUSIVE LOCK omitted
Row or page#1
… (Locking using a lock mode)
Row or page#1
… (Locking using a shared mode)
Delete row
Not deleted
Deleted
Release the row or page#1 from locked status#2
Check whether there is a search condition for rows
Yes
No
①
②


①
Check whether the row satisfies the condition
Yes
No
WITH EXCLUSIVE LOCK specified
WITH EXCLUSIVE LOCK omitted
Release the row or page#1 from locked status#2
Release the row or page#1 from locked status#2
Retain the current status as is
②
WITH EXCLUSIVE LOCK specified
WITH EXCLUSIVE LOCK omitted
Release the row or page#1 from locked status#2
Retain the current status as is

#1: For a table locked by page, the page is locked.

#2: If a lock mode is applied while a shared mode is in effect, the lock mode is not released even after the processing using the lock mode has terminated.

3.4.10 Lock sequence for rows

HiRDB accesses data on the basis of the access path determined by SQL optimization. Normally, this access path determines the lock sequence for rows. However, the lock sequence for rows can vary depending on conditions during SQL execution. The following are examples of cases where the lock sequence for rows depends on conditions during SQL execution:

- A parallel server is used together with multiple back-end servers to access data
- A snapshot method is used

This subsection discusses the lock sequence for rows based on the access path and the cases where the lock sequence for rows varies.

(1) Lock sequence for rows based on the access path

HiRDB accesses data on the basis of the path determined by SQL optimization. The lock sequence for rows is determined by this access path.

The following example explains the basic sequence of data access.

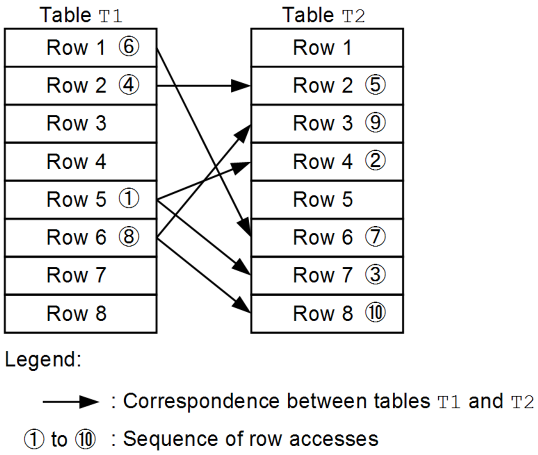
Example

SQL statement to be executed:

```
    select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

Conditions:

- HiRDB/Single Server is used.
- Indexes have been defined for `t1.c1` and `t2.c1`.
- The index for `t1.c1` is used to access `t1`.
- For each row of `t1`, the index for `t2.c1` is used to perform nest-loop join.
- `t1`'s rows 5, 2, 1, and 6 satisfy the condition of `t1.c1 > 3`, and the index is used to access the rows in this order.
- The rows in `t2` that satisfy the rows in `t1` are as follows:
    - `t2`'s row 6 satisfies `t1`'s row 1.
    - `t2`'s row 2 satisfies `t1`'s row 2.
    - `t2`'s rows 4 and 7 satisfy `t1`'s row 5, and the index is used to access the rows in this order.
    - `t2`'s rows 3 and 8 satisfy `t1`'s row 6, and the index is used to access the rows in this order.
- The snapshot method is not used.



(2) Case where the lock sequence for rows varies

This subsection discusses the case where the lock sequence for rows varies according to SQL execution conditions.

(a) HiRDB/Parallel Server

When tables are searched or updated in the parallel mode in order to improve performance, HiRDB pre-reads data and performs parallel data read operations. Therefore, the sequence of data accesses among joined tables might differ from what it would be on a single server, which also means that the lock sequence for rows would be different as well. As a result, deadlocks might occur.

- Partitioned-table search

The following example explains the sequence of data accesses in a partitioned-table search.

Example

SQL statement to be executed:

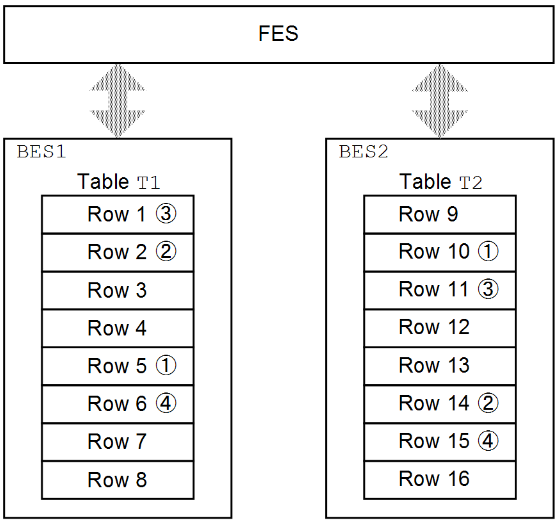
```
select * from t1 where t1.c1>3
```

Conditions:

- HiRDB/Parallel Server is used.
- For the table `t1`, rows 1 through 8 are stored in `BES1`, and rows 9 through 16 are stored in `BES2`.
- Table `T1`'s rows 1, 2, 5, 6, 10, 11, 14, and 15 satisfy the condition `t1.c1`

> 3. In BES1, the index is used to access rows 5, 2, 1, and 6 in this order. In BES2, the index is used to access rows 10, 14, 11, and 15 in this order.

- The snapshot method is not used.



Legend:

① to ④ : Sequence of row accesses in the same back-end server

Explanation:

The access sequence within each back-end server can be determined, but the access sequence for BES1 and BES2 cannot be determined because data can be pre-read or it can be read in the parallel mode. Also, the access sequence for the rows in BES1 and BES2 cannot be determined.

- Join search

The following example explains the sequence of data accesses in a join search.

Example

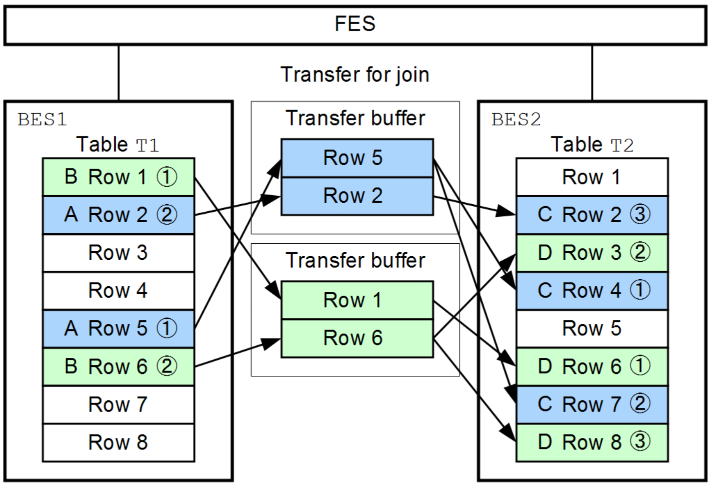
SQL statement to be executed:

```
select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

Conditions:

- HiRDB/Parallel Server is used.
- Indexes have been defined for t1.c1 and t2.c1.

- The index for `t1.c1` is used to access `t1`.
- For each row of `t1`, the index for `t2.c1` is used to perform nest-loop join.
- `t1`'s rows 5, 2, 1, and 6 satisfy the condition of `t1.c1 > 3`, and the index is used to access the rows in this order.
- The rows in `t2` that satisfy the rows in `t1` are as follows:
  - `t2`'s row 6 satisfies `t1`'s row 1.
  - `t2`'s row 2 satisfies `t1`'s row 2.
  - `t2`'s rows 4 and 7 satisfy `t1`'s row 5, and the index is used to access the rows in this order.
  - `t2`'s rows 3 and 8 satisfy `t1`'s row 6, and the index is used to access the rows in this order.
- The snapshot method is not used.
- The number of rows transferred between back-end servers is 2.



Legend:

① to ③ : Sequence of data accesses in the same back-end server (BES)

A : Transfer row group A

B : Transfer row group B

C : Group of rows to be compared with transfer row group A

D : Group of rows to be compared with transfer row group B

Explanation:

In the figure, the access sequence for A through D is as follows:

- Within each of the groups A through D, rows are accessed in the order indicated by the circled numbers.
- B and C are accessed after A has been accessed.
- The access sequence for B and C cannot be determined because data can be pre-read or it can be read in the parallel mode.
- The access sequence for rows in B and C cannot be determined because data can be pre-read or it can be read in the parallel mode.

(b) When using the snapshot method

When the snapshot method is used, the lock sequence for rows changes each time an SQL statement is executed.

This subsection explains the sequence of locking for rows in the following three cases:

- When the snapshot method is used (only one transaction is executed at a time)
- When the snapshot method is used (multiple transactions are executed concurrently)
- When the snapshot method is not used

The following SQL statement is used for the example.

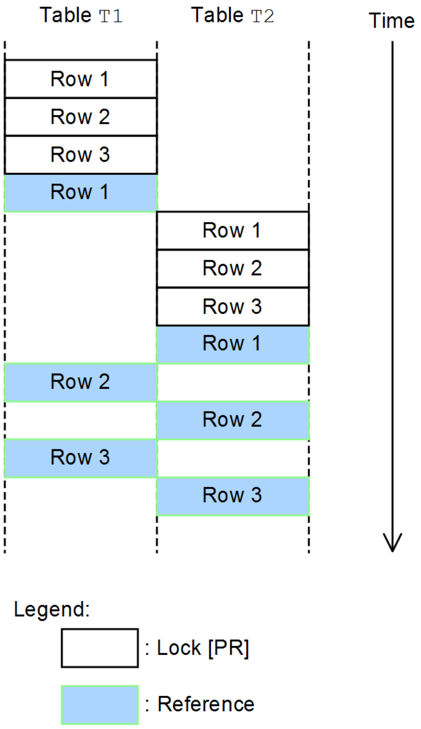
Example

```
select * from t1,t2 where t1.c1=t2.c1 and t1.c1>3
```

- When the snapshot method is used (only one transaction is executed at a time)

  If the snapshot method is used and the page to be referenced contains a row that is being updated (including rows retrieved by the `WITH EXCLUSIVE LOCK` lock option), all rows on the page are locked.

  The following shows the sequence of data accesses during SQL execution:



Explanation:

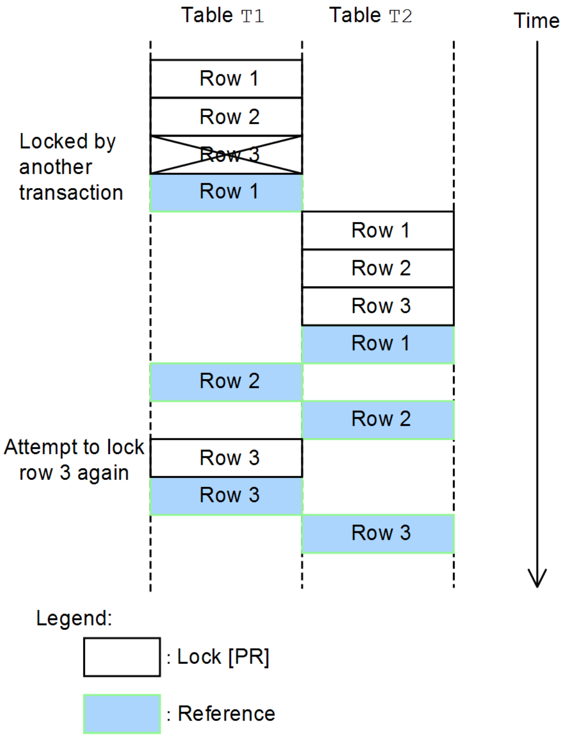
- The row 1s in tables `T1` and `T2` indicate that the values of columns specified in the join condition are the same.

- All rows in tables T1 and T2 are stored on the same page.

- When the snapshot method is used (multiple transactions are executed concurrently)

  If the snapshot method is used and the page to be referenced contains a row that is being updated (including rows retrieved by the WITH EXCLUSIVE LOCK lock option), all rows up to that row are locked. After that, the lock processing is canceled and the rows that have been locked so far are referenced. The rows that were not locked will be locked when they are actually referenced.

  The following shows the sequence of data accesses during SQL execution.



Explanation:

- The row 1s in tables T1 and T2 indicate that the values of columns specified in the join condition are the same.
- All rows in tables T1 and T2 are stored on the same page.
- Table T1's row 3 is locked in the EX mode by another transaction.
- If the snapshot method is used to perform a join search and there is

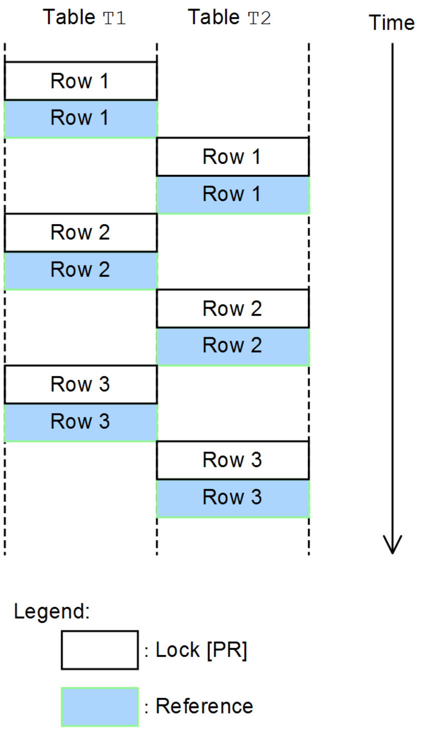
another UAP that updates the tables or indexes, or that uses the WITH EXCLUSIVE LOCK lock option to perform a search, the lock sequence for rows will vary even for the same SQL statement.

- Because the snapshot method locks all pages to be referenced, the lock sequence for rows varies when index page splitting occurs.

- When the snapshot method is not used

  When the snapshot method is not used, one row is locked at a time.

  The following shows the sequence of data accesses during SQL execution:

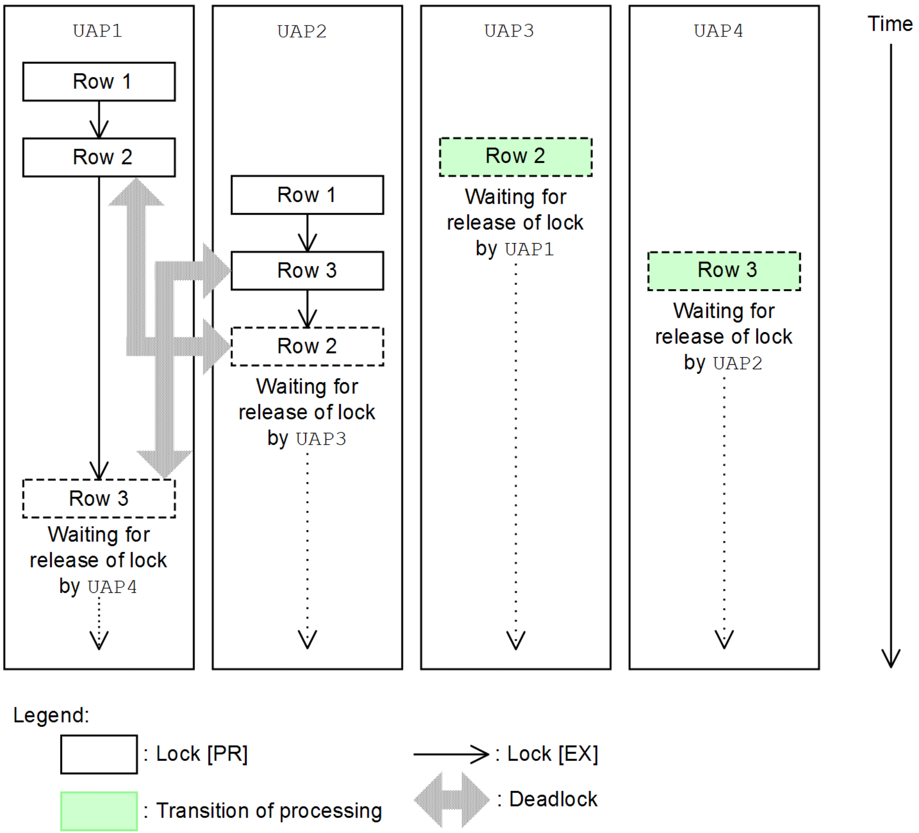


Explanation:

The row 1s in tables T1 and T2 indicate that the values of columns specified in the join condition are the same.

(3) How to handle deadlocks

If the lock sequence for rows varies and a UAP that locks rows in the EX mode is added, deadlock might result even from concurrent execution of multiple identical SQL statements. The following shows an example of such deadlock:


UAP1
Row 1
Row 2
Row 3
Waiting for release of lock by UAP4
UAP2
Row 1
Row 3
Row 2
Waiting for release of lock by UAP3
UAP3
Row 2
Waiting for release of lock by UAP1
UAP4
Row 3
Waiting for release of lock by UAP2
Time
Legend:
: Lock [PR]
: Transition of processing
: Lock [EX]
: Deadlock

Explanation:

- UAP1 and UAP2 lock rows in the PR mode.
- UAP1 and UAP2 execute the same SQL statement, but the lock sequence for rows varies as follows for the reason discussed in (2) above:

  UAP1: Locks rows 1, 2, and 3 in this order.

  UAP2: Locks rows 1, 3, and 2 in this order.
- UAP3 and UAP4 locks rows in the EX mode.

You can take the following actions to handle deadlocks:

- Perform the search with the WITHOUT LOCK NOWAIT or the WITHOUT LOCK WAIT lock option specified.
- Execute the LOCK TABLE statement in IN EXCLUSIVE MODE before performing the search.

- Re-execute the transaction.

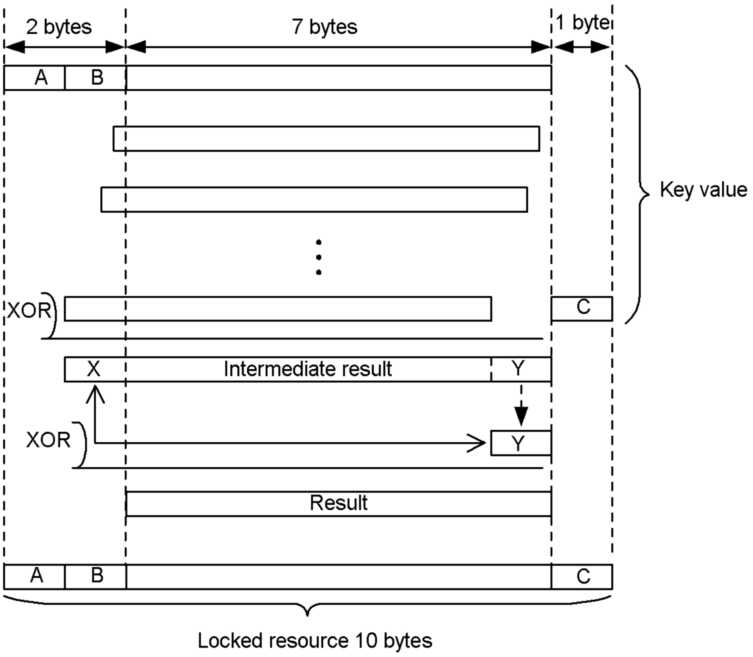
For details about how to handle deadlocks, see *3.4.4 Deadlocks and corrective measures*.

3.4.11 Creating locked resources for index key values

If a key value for an index exceeds 10 bytes, the system creates a different locked resource for the index key value, according to the value that was specified for the `pd_key_resource_type` operand in the system definitions. For details about the `pd_key_resource_type` operand of the system definitions, see the *HiRDB Version 9 System Definition* manual.

Figure 3-14 shows how a key value locked resource is created when `TYPE1` is specified in the `pd_key_resource_type` operand. *Figure 3-15* shows how a key value locked resource is created when `TYPE2` is specified.

Figure 3-14: Creation of a key value locked resource when pd_key_resource_type=TYPE1 is used



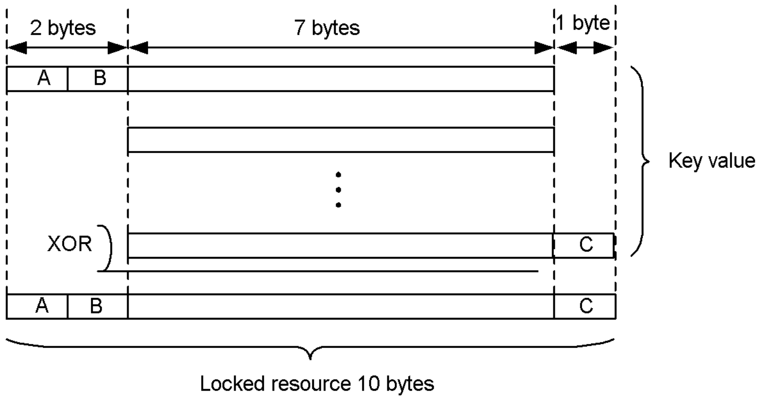
A, B, and C indicate byte 1, byte 2, and the last byte, respectively.

If the key value length exceeds 10 bytes, the system removes the first two bytes and the last byte of the key value, extracts the remaining data in 7-byte units, and applies `exclusive-OR` while bit-shifting the units. The bit shift operation logically shifts each unit by the remainder after the extraction count is divided by eight and applies exclusive-OR for 8-byte data to the units.

The system stores the exclusive-OR result (intermediate result) to an 8-byte area and applies exclusive-OR to the first (X) and last (Y) bytes of the intermediate result to create a 7-byte data value (result).

The system combines the 7-byte data (result) with the first two bytes and the last byte that were removed initially and sets the resulting 10-byte data value as the locked resource of the index key value.

Figure 3-15: Creation of a key value locked resource when pd_key_source_type=TYPE2 is used



A, B, and C indicate byte 1, byte 2, and the last byte, respectively.

If the key value length exceeds 10 bytes, the system removes the first two bytes and the last byte of the key value, extracts the remaining data in 7-byte units, and applies exclusive-OR. The system combines the exclusive-OR result with the first two bytes and the last byte that were removed initially, and sets the resulting 10-byte data value as the locked resource of the index key value.

3.5 Use of a cursor

You can use a cursor in a UAP to extract retrieval results.

To use a cursor, declare the cursor with `DECLARE CURSOR` or allocate the cursor with `ALLOCATE CURSOR`.

This section explains the effects of using a cursor and issues to consider when using a cursor.

3.5.1 Notes on table operations when a cursor is used

(1) How operations that do not use a cursor relate to cursor updatability and whether an operation that uses a cursor is performed

Once you declare or allocate a cursor and open it with an `OPEN` statement, you can extract data and perform other operations such as referencing and updating. However, after the cursor is opened, whether or not operations that do not use a cursor can be performed depends on the specification of the `FOR READ ONLY` or `FOR UPDATE` clause and whether or not an operation that uses a cursor (updating or deletion) is performed. The `FOR READ ONLY` and `FOR UPDATE` clauses are specified in the cursor declaration and in the `SELECT` statement identified by either the SQL statement identifier specified in the cursor declaration or the extended statement name specified in the cursor allocation.

The table below shows the relationships between cursor updatability and operations that do not use a cursor. When the SQL optimization option for suppressing creation of update-SQL work tables is specified, the restrictions on operations that do not use a cursor are relaxed.

Table 3-25: Relationships between cursor updatability and operations that do not use a cursor

Condition				**Operation that does not use a cursor**							
Specification of cursor updatability		**Process that uses a cursor**		**SQL optimization option for suppressing creation of update-SQL work tables not applied**				**SQL optimization option for suppressing creation of update-SQL work tables applied and index key value no-lock facility used**			
		UD	**Del**	**Ret**	**UD**	**Del**	**Add**	**Ret**	**UD**	**Del**	**Add**
Static SQL	`FOR READ ONLY` clause specified	NP	NP	Y	Y	Y	Y	Y	Y	Y	Y

Condition				Operation that does not use a cursor							
Specification of cursor updatability		Process that uses a cursor		SQL optimization option for suppressing creation of update-SQL work tables not applied				SQL optimization option for suppressing creation of update-SQL work tables applied and index key value no-lock facility used			
		UD	Del	Ret	UD	Del	Add	Ret	UD	Del	Add
	FOR UPDATE OF column name specified	NP	NP	Y	CU	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	CU	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		P	P	Y	Y	Y	Y	Y	Y	Y	Y
	FOR UPDATE clause specified	NP	NP	Y	Y	Y	Y	Y	Y	Y	Y
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	Y	Y	Y	Y	Y	Y	Y
		P	P	Y	Y	Y	Y	Y	Y	Y	Y
	None of the above[#1]	NP	NP	Y	N	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	Y	Y	Y	Y	Y	Y	Y
		P	P	Y	Y	Y	Y	Y	Y	Y	Y
Dynamic SQL	FOR READ ONLY clause specified	NP	NP	Y	Y	Y	Y	Y	Y	Y	Y
	FOR UPDATE OF column name specified	NP	NP	Y	CU	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	CU	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		P	P	Y	Y	Y	Y	Y	Y	Y	Y

Condition				Operation that does not use a cursor							
Specification of cursor updatability		Process that uses a cursor		SQL optimization option for suppressing creation of update-SQL work tables not applied				SQL optimization option for suppressing creation of update-SQL work tables applied and index key value no-lock facility used			
		UD	Del	Ret	UD	Del	Add	Ret	UD	Del	Add
	FOR UPDATE clause specified	NP	NP	Y	Y	Y	Y	Y	Y	Y	Y
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	Y	Y	Y	Y	Y	Y	Y
		P	P	Y	Y	Y	Y	Y	Y	Y	Y
	None of the above	NP	NP	Y	N	N	N	Y	Y[#2]	Y[#2]	Y[#2]
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	Y	Y	Y	Y	Y	Y	Y
		P	P	Y	Y	Y	Y	Y	Y	Y	Y

The abbreviations in the column headers denote the following:

UD: Updating

Del: Deletion

Ret: Retrieval

Add: Addition

Legend:

P: Performed

NP: Not performed

Y: Can be performed.

CU: Specified column can be updated.

N: Cannot be performed.

#1: Specification of the `FOR UPDATE` clause is assumed when the same post source contains an update or deletion in which the `CURRENT OF` cursor name is specified.

#2: If the index being used in the retrieval that uses the cursor is updated, the retrieval results that were obtained with the cursor are not guaranteed. An example of this case and countermeasures are shown as follows.

Example:

```
CREATE INDEX X1 ON T1(C1);
DECLARE CR1 CURSOR FOR SELECT C1 FROM T1 WHERE C1>0;
```

The cursor that was declared is used to execute the following FETCH and UPDATE statements repeatedly:

```
FETCH CR1 INTO :XX;
UPDATE T1 SET C1=10;
```

The line that was updated to C1=10 is retrieved again.

Countermeasure:

Implement one of the following countermeasures:

- Change the search conditions so that the update value in the UPDATE statement does not satisfy the search condition.

  Example: WHERE C1>0 AND C1 <>10

- Delete the problem-causing column from the configuration columns of the index used in the retrieval. However, note that when a configuration column of the index is deleted, the performance may drop if the column was one that significantly narrowed the search described by the search conditions. Also note that deleting a configuration column of the index increases the number of index key duplications and may increase the incidence of lock-release waiting and deadlock. Therefore, check the potential effects of this countermeasure thoroughly before applying it.

(2) Using multiple cursors simultaneously

To use multiple cursors for updating the same table simultaneously, you must specify all columns to be updated in the FOR UPDATE clause of the individual cursor declarations or dynamic SELECT statements. For example, to use cursor 1 to update column 1 and cursor 2 to update column 2, specify both column 1 and column 2 in the FOR UPDATE clause when you declare cursor 1 and cursor 2. If only column 1 is specified for cursor 1 or only column 2 is specified for cursor 2, an error occurs during updating.

3.5.2 FOR UPDATE and FOR READ ONLY clauses

To use a cursor to perform row updating, deletion, or insertion on a table being retrieved, you must define the cursor with DECLARE CURSOR or ALLOCATE CURSOR. When you define the cursor, specify the FOR UPDATE (including FOR UPDATE OF) and FOR READ ONLY clauses according to the processing contents of the UAP.

A good way to update or delete a row that uses a cursor without updating nearly all of the retrieved rows is to specify `WITH SHARE LOCK` as the lock option. If a lock option is not specified, `WITH EXCLUSIVE LOCK` is assumed.

Care must be taken, because specifying the `FOR UPDATE` (or `FOR UPDATE OF`) clause or the `FOR READ ONLY` clause may result in a significant drop in processing efficiency in some cases.

The following table lists issues to be considered when the `FOR UPDATE` (or the `FOR UPDATE OF`) clause and the `FOR READ ONLY` clause are specified.

Table 3-26: Specifying FOR UPDATE and FOR READ ONLY clauses

Application		Consideration
`FOR UPDATE` clause	Specified for a table being retrieved using a cursor when the rows for which the cursor is used will be updated or deleted, followed by updating, deletion, or addition of rows for which the cursor is not used.	To guarantee correct operation even when the target index is updated during retrieval of a row for which the cursor is used, a work table is created internally during the first `FETCH`; creation of this work table involves overhead during retrieval.
`FOR UPDATE OF` clause	Specified for a table being retrieved using a cursor when only some of the columns will be updated.	When the index assigned to a column specified by its column name is used for retrieval, a work table is created internally during the first `FETCH`; creation of this work table involves overhead during retrieval.
`FOR READ ONLY` clause	Specified when another cursor will be used for updating (or deletion or insertion) during retrieval using a cursor or for updating (or deletion) by directly specifying a search condition.	When another cursor will be used for updating during retrieval using a cursor, a work table is created internally during the first `FETCH` so that there will be no impact on the processing result; creation of this work table involves overhead during retrieval.

Even if the `FOR UPDATE` or `FOR READ ONLY` clause is not specified, a work table may be created internally during the first `FETCH`, so overhead creation must still be taken into account.

No internal work table is created when only retrieval is to be performed; in this case, do not consider overhead.

3.5.3 Cursor declarations and locks

When a `FETCH` or a single-row `SELECT` statement is executed, the lock mode that has priority is based on the lock option specified in the cursor declaration, dynamic `SELECT` statement, or preprocessing for the single-row `SELECT` statement. If the cursor declaration does not specify a lock option, the lock mode is determined by the data guarantee level (if a data guarantee level is not specified, the default is 2). The data guarantee level is specified with `PDISLLVL` in the client environment definitions or with ISOLATION LEVEL in the SQL compile options specified in the procedure or trigger definitions. When the data guarantee level is used, the lock mode is also affected by whether updating (or deletion) using a cursor is specified and whether WITH EXCLUSIVE LOCK is assumed during FOR UPDATE processing.

The specification for assuming WITH EXCLUSIVE LOCK during FOR UPDATE processing requires specifying PDFORUPDATEEXLOCK in the client environment definitions and the data guarantee level (specification of FOR UPDATE EXCLUSIVE) in the SQL compile options specified in the procedure or trigger definitions.

The lock mode that takes effect during execution depends on the lock option specified in the cursor declaration (DECLARE CURSOR), the lock option in the dynamic SELECT statement specified in the cursor declaration (DECLARE CURSOR) and the cursor allocation (ALLOCATE CURSOR), or the lock option specified in the single-row SELECT statement. The following table shows the relationships between the lock option specified during cursor declaration or dynamic SELECT statement preprocessing, and the lock option specified during table operations.

For details about the lock option in the cursor declaration, see the manual *HiRDB Version 9 SQL Reference*.

Table 3-27: Relationships between the lock option specified during cursor declaration or dynamic SELECT statement preprocessing and the lock option specified during table operations

Lock option in SQL statement		**WITH EXCLUSIVE LOCK assumed during FOR UPDATE processing**	**Data guarantee level**	**Update permission using cursor[#1]**	**Lock option during table manipulation and assumed value in FOR UPDATE clause**
Specified	WITH EXCLUSIVE LOCK	--[#2]	--[#2]	No	WITH EXCLUSIVE LOCK
				Yes	WITH EXCLUSIVE LOCK FOR UPDATE
	WITH SHARE LOCK			No	WITH SHARE LOCK
				Yes	WITH SHARE LOCK FOR UPDATE
	WITHOUT LOCK WAIT			No	WITHOUT LOCK WAIT
				Yes	WITHOUT LOCK WAIT FOR UPDATE

Lock option in SQL statement		WITH EXCLUSIVE LOCK assumed during FOR UPDATE processing	Data guarantee level	Update permission using cursor[#1]	Lock option during table manipulation and assumed value in FOR UPDATE clause
	WITHOUT LOCK NOWAIT			No	WITHOUT LOCK NOWAIT
				Yes	Error
Not specified		Yes	2	No	WITH SHARE LOCK
				Yes	WITH EXCLUSIVE LOCK FOR UPDATE
			1	No	WITHOUT LOCK WAIT
				Yes	WITHOUT LOCK WAIT FOR UPDATE
			0	No	WITHOUT LOCK NOWAIT
				Yes	WITHOUT LOCK WAIT FOR UPDATE
		No	2	No	WITH SHARE LOCK
				Yes	WITH EXCLUSIVE LOCK FOR UPDATE
			1	No	WITHOUT LOCK WAIT
				Yes	WITHOUT LOCK WAIT FOR UPDATE
			0	No	WITHOUT LOCK NOWAIT
				Yes	WITHOUT LOCK WAIT FOR UPDATE

Legend:

--: Does not apply.

Notes

Depending on which lock option is specified, the following conditions may occur during execution:

- When WITH SHARE LOCK is specified

  Because rows in the table will be changed from the protected retrieve mode to the exclusive mode during updating, deadlock may occur.

- When WITHOUT LOCK WAIT is specified

  Incorrect updating (double updating) or a deletion error may occur depending on other transactions.

- When WITHOUT LOCK NOWAIT is specified

  If an SQL statement updates a table retrieved with WITHOUT LOCK NOWAIT specified, an error will occur.

#1: Update using a cursor is permitted in the following cases:

- The FOR UPDATE clause is specified.
- The FOR UPDATE clause is not specified, but there is an UPDATE or DELETE statement that specifies the same cursor (the cursor specified in the cursor declaration).

Update using a cursor is not permitted in the following case:

- The FOR UPDATE clause is not specified, and there is also no UPDATE or DELETE statement that specifies the same cursor (the cursor specified in the cursor declaration).

#2: The lock option in the SQL statement has priority regardless of the specified contents.

HiRDB might have pre-read rows targeted for retrieval by the FETCH statement. Because pre-read rows are locked, more rows might be locked than the rows acquired by the FETCH statement. In such a case, if the number of resources to be locked is estimated based on the number of rows to be retrieved by the FETCH statement, rather than on the actual number of rows retrieved, a shortage of lock buffer space might occur. If you use a cursor, estimate the number of resources to be locked based on the actual number of rows retrieved. To limit the number of resources to be locked, you must narrow down the rows by using search conditions. If the target rows cannot be narrowed down, you must consider disabling locking (such as by changing the locking units). For details about disabling locking, see *3.4.8 Locking by UAPs*.

3.5.4 Holdable cursor

(1) Overview

A holdable cursor does not close, even when a COMMIT statement is executed.

To use a holdable cursor, declare the cursor by specifying UNTIL DISCONNECT or WITH HOLD in the DECLARE CURSOR statement. When these statements are specified, the cursor remains open until execution of a CLOSE, DISCONNECT, or ROLLBACK statement (including ROLLBACK and DISCONNECT processing that is executed implicitly if an error occurs).

(2) Advantages of using a holdable cursor

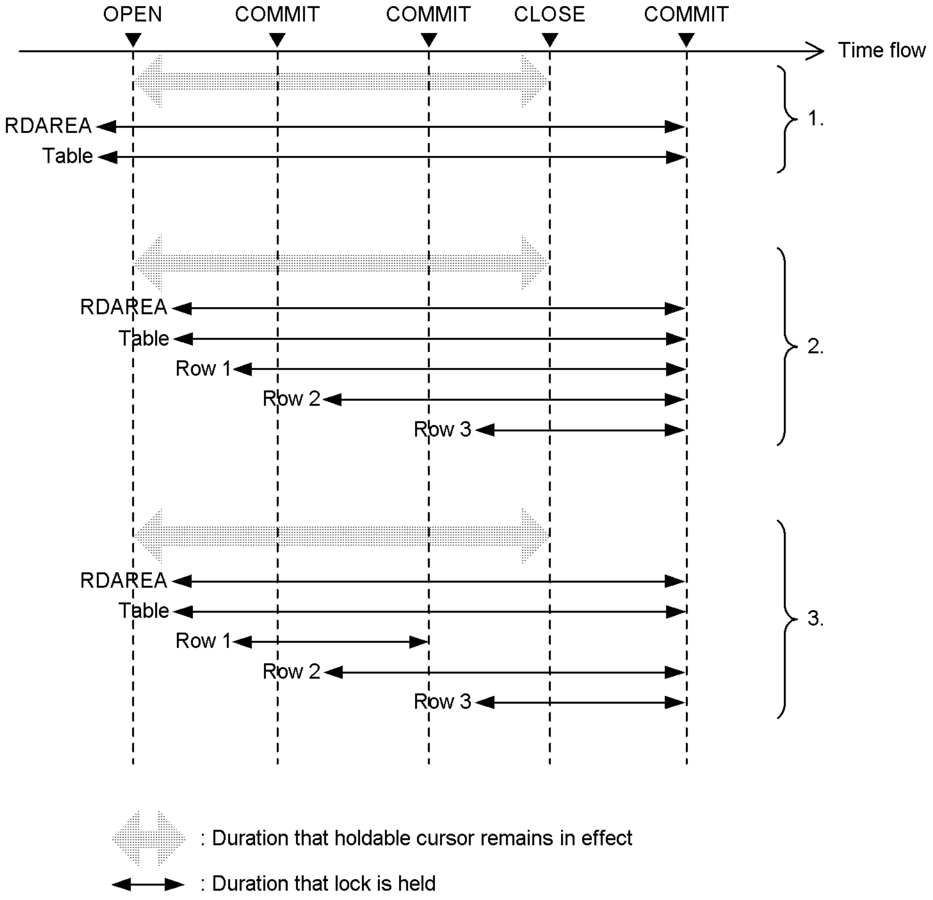
Using a holdable cursor can reduce the incidence of locked resources, because a COMMIT statement can be executed while retrieving or updating a large amount of data. Moreover, because a COMMIT statement can be executed while keeping the cursor open, a synchronization point can be activated even when a large amount of data is being retrieved or updated (i.e., when a transaction executes for an extended period of time), thus reducing the restart time.

(3) Processing using a holdable cursor

When a holdable cursor is used, deletion of a work table file and release of work table buffer space are executed during commit processing after the holdable cursor used to create the work table file has closed.

When a holdable cursor is opened, each back-end server process becomes occupied even if there are no transactions. Therefore, the maximum number of server processes must be estimated carefully when a holdable cursor is to be used.

The locked resources that are inherited beyond a transaction differ depending on whether a LOCK statement with UNTIL DISCONNECT specified is executed and whether a search in which a work table is created or a parallel scan is executed. The locked resources that are inherited are shown below.



Explanation:

The numbers in the figure are explained below.

No.	Execution of LOCK statement with UNTIL DISCONNECT specified	Search in which work table is created or parallel scan	Inherited locked resources
1	Executed	Not applicable	Only the resource of the LOCK statement
2	Not executed	Executed	All resources
3		Not executed	Only the resource at the cursor position

When a UAP executing in an OLTP environment is using a holdable cursor, use must be specified in the `pd_oltp_holdcr` operand of the HiRDB system definition.

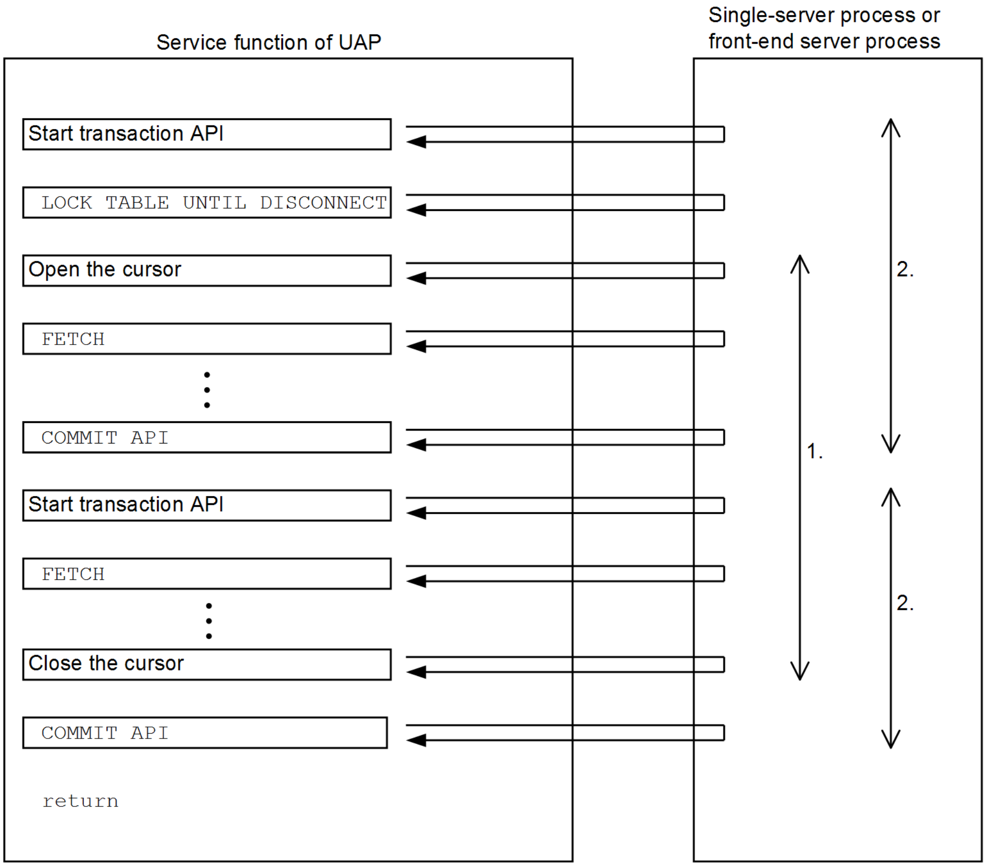
For a UAP executing in an OLTP environment to use a holdable cursor, the following conditions must be satisfied:

- The UAP must use an X/Open-compliant API to access HiRDB.
- The service function of the UAP using the holdable cursor must perform cursor postprocessing sometime after the holdable cursor is opened but before the service function returns control.#

#: The cursor postprocessing procedure is described below:

1. Close the cursor.
2. Execute the `ROLLBACK` statement.
3. Execute the `DISCONNECT` statement.
4. Terminate the UAP.

The SQL execution sequence from the UAP service function is shown below. Note the sequential relationships between the start transaction API, `OPEN` cursor, `CLOSE` cursor, and `COMMIT API` steps in the figure.



Explanation:

1. Duration that the cursor is held
2. Duration of the transaction

3.5.5 Examples of cursor use

This section shows examples in which a cursor is used.

(1) Example of updating a table while retrieving rows with a cursor

This example discounts the price (PRICE) values by 10% while using a cursor (CR1) to retrieve all rows from a stock table (STOCK).

```
      :
EXEC SQL BEGIN DECLARE SECTION;
  char xpcode[5]  ; ..........................................1
  char xpname[17];
```

```
..........................................1
  char xcolor[3];
..........................................1
  int xprice;   ..........................................1
  int xsquantity;  .......................................1
EXEC SQL END DECLARE SECTION;
   :
EXEC SQL DECLARE CR1 CURSOR FOR
    SELECT * FROM STOCK
     FOR UPDATE OF PRICE; ..............................2
EXEC SQL OPEN CR1; .....................................3
EXEC SQL FETCH CR1
INTO:xpcode,:xpname,:xcolor,:xprice,:xsquantity;.......4
EXEC SQL UPDATE STOCK
    SET PRICE=0.9*:xprice
    WHERE CURRENT OF CR1; ............................5
EXEC SQL CLOSE CR1; ..................................6
   :
```

Explanation:

1. Declares an embedded function to be used in retrieval, update, and insertion.
2. Declares cursor `CR1`. `FOR UPDATE OF` *column-name* is specified in this statement because cursor `CR1` will be used to update only the `PRICE` column.
3. Opens cursor `CR1`.
4. Fetches the value from the `PRICE` column of the row indicated by cursor `CR1` and places the value in embedded variable (`:xprice`).
5. Discounts the `PRICE` value by 10% (`0.9*:xprice`).
6. Closes cursor `CR1`.

(2) Example of updating while retrieving rows with a cursor and then inserting rows

This example updates a stock table (`STOCK`) while using a cursor (`CR1`) to retrieve all rows from the table. The example then inserts a row without using the cursor (`CR1`).

```
    :
EXEC SQL BEGIN DECLARE SECTION;
  char xpcode[5]; ....................................1
  char xpname[17] ; ..................................1
  char xcolor[3] ; ...................................1
  int xprice ; .......................................1
  int xsquantity ; ...................................1
EXEC SQL END DECLARE SECTION;
          :
```

```
EXEC SQL DECLARE CR1 CURSOR FOR
    SELECT * FROM STOCK
      FOR UPDATE;       .................................2
EXEC SQL OPEN CR1;         .................................3
EXEC SQL FETCH CR1
 INTO:xpcode,:xpname,:xcolor,:xprice,:xsquantity; ......4
EXEC SQL UPDATE STOCK
    SET QUANTITY=:xsquantity+100
    WHERE CURRENT OF CR1; ............................5
EXEC SQL INSERT INTO STOCK
VALUES(:xpcode,:xpname,:xcolor,:xprice,:xsquantity); ...6
EXEC SQL CLOSE CR1; ...................................7
          :
```

Explanation:

1. Declares an embedded function to be used in retrieval.
2. Declares cursor `CR1`. Cursor `CR1` is used to update the table, and the `FOR UPDATE` clause, is specified for row insertion without using cursor `CR1`.
3. Opens cursor `CR1`.
4. Fetches the values from the row indicated by cursor `CR1` and places the values in the embedded variables.
5. Adds 100 to the `QUANTITY` value.
6. Inserts a row into the `STOCK` table without using cursor `CR1`.
7. Closes cursor `CR1`.

(3) Example of using a holdable cursor

This example modifies the price (`PRICE`) values to 50% of the original values while using a cursor (`CR1`) to retrieve all rows from a stock table (`STOCK`). The cursor (`CR1`) is left open and is used for another manipulation.

```
      :
EXEC SQL BEGIN DECLARE SECTION:
  char xpcode[5] ; ...................................1
  char xpname[17] ; ..................................1
  char xcolor[3] ; ...................................1
  int xprice ;      ....................................1
  int xsquantity ; ..................................1
END DECLARE SECTION ;
      :
EXEC SQL LOCK TABLE STOCK
    IN EXCLUSIVE MODE UNTIL DISCONNECT; .................2
      :
EXEC SQL DECLARE CR1 CURSOR WITH HOLD FOR
```

```
    SELECT * FROM STOCK
      FOR UPDATE OF PRICE ..............................3
EXEC SQL OPEN CR1; ....................................4
EXEC SQL FETCH CR1
  INTO :xpcode,:xpname,:xcolor,:xprice,:xsquantity; ....5
EXEC SQL UPDATE STOCK SET PRICE=0.5*:xprice
  WHERE CURRENT OF CR1; ................................6
Decision for executing next COMMIT statement in
1000-row units                       ...................7
EXEC SQL COMMIT;  ......................................8
Execution of the next CLOSE statement after all rows
have been updated  .....................................9
EXEC SQL CLOSE CR1; ....................................10
      :
```

Explanation:

1. Declares an embedded variable (for example, `:xprice`) to be used in retrieval and update.
2. Locks the `STOCK` table with a `LOCK` statement in which `UNTIL DISCONNECT` is specified, so that a holdable cursor can be used. This statement also specifies a lock mode (`IN EXCLUSIVE MODE`) because the cursor is used for updating the table.
3. Declares cursor CR1. The cursor declaration specifies `WITH HOLD` because the cursor is a holdable cursor. The `PRICE` column is specified in the `FOR UPDATE OF` clause because `PRICE` is the only column to be updated.
4. Opens cursor CR1.
5. Fetches the value from the `PRICE` column in the row indicated by cursor `CR1` and places it in embedded variable (`:xprice`).
6. Modifies the `PRICE` value to 50% of the original value (`0.5*:xprice`).
7. Specifies the decision for executing the next `COMMIT` statement for each 1,000 rows to be updated or for continuing the update process if the `COMMIT` statement is not to be executed.
8. Commits the update process.
9. Specifies the decision for executing the next `CLOSE` statement if there are no rows to be updated or continuing the update process if there are still rows to be updated.
10. Closes cursor `CR1`.

3.6 SQL error identification and corrective measures

When a UAP is used to execute an SQL statement, it is important to ascertain whether or not the SQL statement executed correctly.

This section explains how to determine whether or not an SQL statement executed correctly and the measures to be taken when an error is detected.

3.6.1 Error identification

(1) Return codes

HiRDB sets up return codes (`SQLCODE` and `SQLSTATE`) when SQL statements execute. However, HiRDB does not set return codes for declaration statements, such as `DECLARE CURSOR`. The following variables can be used to reference the return codes:

- `SQLCODE`
- `SQLSTATE`

An SQL statement's execution status can be determined by referencing the `SQLCODE` and `SQLSTATE` variables.

The following table shows the relationships between SQL statement execution status and the values set in variables.

Table 3-28: Values set in variables and SQL statement execution status

SQL statement execution status		**SQLCODE variable value**	**SQLWARN0 value**	**SQLWARN6 value**	**SQLSTATE variable value**
Normal termination	Without warning	`0`	`'△'`	--	`'00000'`
	With warning[#4]	`0`	`'W'`	--	`'01`*nnn*`'`[#1] (*nnn* ≠ `R00`)
		`>0 (≠ 100, 110)`	--	--	`'R01R00'`
	Without data[#3]	`110`	--	--	`'R2000'`
No data		`100`	--	--	`'02000'`
Termination with error	Without implicit rollback	`-1` to `-1999`	--	`'△'`	'*mmnnn*'[#2]
	With implicit rollback	`-1` to `-1999`	`'W'`	`'W'`	`'40`*nnn*`'`[#1]

mm: Class

nnn: Subclass

--: No value is set.

#1: The SQLSTATE subclass is set in *nnn*. For details about SQLSTATE, see the manual *HiRDB Version 9 Messages*.

#2: The SQLSTATE class is set in *mm*. For details about SQLSTATE, see the manual *HiRDB Version 9 Messages*.

#3: This status occurs when a search using a list is executed and a row that was present when the list was created is not returned.

#4: The warning information is set in the SQLWARN1 to SQLWARNF areas or is indicated by the SQLCODE value (positive number other than 100). When warning information is set in the SQLWARN1 to SQLWARNF areas, W is set in SQLWARN0. Therefore, when SQLWARN0 contains W, the SQLWARN1 to SQLWARNF areas should also be checked.

For details about the contents of the SQLWARN0 to SQLWARNF areas, see *A. SQL Communications Area*.

If warning information is indicated with an SQLCODE value (positive number other than 100), the SQLSTATE subclass (*nnn*) becomes R00. The following table shows the relationships among the SQLSTATE, SQLCODE, and SQLWARN0 values when normal termination with a warning occurs.

Table 3-29: Relationship among SQLSTATE, SQLCODE, and SQLWARN0 values when normal termination with a warning occurs

SQLSTATE value	SQLCODE value	SQLWARN0 value
01*nnn* (*nnn* ≠ R00)	0	W
01R00	Positive number other than 100	Null or 'W'

(a) SQLCODE=100 or SQLSTATE='02000'

The UAP determines that there are no more rows to be retrieved.

This setting is useful for determining the following:

- There are no more rows to be fetched with the FETCH statement.
- No row was selected with the 1-row SELECT statement.
- There were no rows to be updated with the INSERT, DELETE, or UPDATE statement.

(b) SQLCODE<0 or SQLSTATE='mmnnn '(mm is not '00', '01', or '02', or mm is not '00', '01', '02', or 'R2' when a search using a list is executed)

The UAP determines that an SQL error occurred.

If an SQL error occurred, implicit rollback may also have occurred. If SQLWARN6='W' or SQLSTATE='40*nnn*', the UAP determines that implicit rollback occurred.

To identify the SQL statement that caused the error, check the SQL trace information. For details about the SQL trace information, see *11.1.1 SQL tracing*.

(c) Values other than (a) or (b)

The UAP determines that the SQL statements terminated normally. Normal termination may come with warning information. If SQLWARN0='W', SQLCODE is a positive value other than 100, or SQLSTATE='01*nnn*', the UAP determines that normal termination with warning occurred.

When a search using a list is executed, normal termination without any data (a row that was present when the list was created has been deleted) may occur. If SQLCODE is 110 or SQLSTATE is 'R2000', the UAP must determine that normal termination without any data occurred and skip the processing for selection rows.

For details about normal termination with warning, see *Table 3-29*.

(2) Corrective measures for detected errors

When you detect an error, use the following procedure:

1. Output or display the return codes.
2. If the cause of the error cannot be determined on the basis of the return code alone, specify display or output of additional return code information. It is also possible to display either the SQL statement at which the error occurred or information that can be used to identify the affected SQL statement.

   The following table shows the additional return code information and the item that is referred to by the information.

   Table 3-30: Additional return code information and items referred to by the information

Additional information	Referenced item
Message concerning SQLCODE#	SQLERRML field in SQL Communications Areas and contents of SQLERRMC field

#: If the FETCH statement is re-executed after an error has occurred, HiRDB returns the return code for the previous error; however, variable parts of such an error message may not be applicable.

1. Cancel the transaction (ROLLBACK or abort the UAP). If a UAP transaction is

rolled back implicitly by deadlock, the following processing is executed:

Normal UAP:

If a transaction is rolled back implicitly, the next SQL statement that is executed becomes the start of a new transaction (the ROLLBACK or DISCONNECT statement can also be executed).

UAP executing in an OLTP environment:

If a transaction is rolled back implicitly, HiRDB cannot accept any statements except DISCONNECT or ROLLBACK from the UAP executing in the OLTP environment.

If an X/Open-compliant UAP is operated as a client in the OLTP environment and a deadlock occurs while the UAP is being executed, the affected transaction must be terminated.

2. Terminate the UAP or start a transaction (new execution of a different transaction or re-execution of the same transaction).

   Before re-executing the same transaction, you must take error correction measures. If the transaction is re-executed before the cause of the error has been eliminated, the system may enter an endless loop. If the same error occurs after re-execution, you may have to terminate the UAP.

3.6.2 Automatic error identification

When WHENEVER statements are used, errors can be detected automatically.

WHENEVER statements can identify the following conditions:

- Error occurrence
- No more rows to be retrieved
- Whether or not warning information is present in normal termination

For details about the WHENEVER statement, see the *HiRDB Version 9 SQL Reference* manual.

(a) Identification of an error occurrence (SQLCODE<0)

Determined by using the WHENEVER statement where SQLERROR is specified. When an error occurs, the system shifts to the specified measure. If an error referencing operation is specified, the return codes and related information can be referenced.

(b) No more rows to be retrieved (SQLCODE=100)

Determined by using the WHENEVER statement where NOT FOUND is specified. By specifying the process to be taken when there are no more lines to be searched, the system shifts to the specified measure.

(c) Whether or not warning information is present in normal termination (SQLWARN0='W', or SQLCODE>0 but SQLCODE ≠ 100)

Determined by using the WHENEVER statement where SQLWARNING is specified. When a measure to be taken when warning information is present in normal termination is specified, the system shifts to the specified measure only when warning information is present.

When a search using a list is executed, normal termination without any data (a row that was present when the list was created has been deleted) may occur. If SQLCODE is 110 or SQLSTATE is R2000, the UAP must determine that normal termination without any data occurred and skip the processing for selection rows.

Chapter

4. UAP Design for Improving Performance and Handling

This chapter describes issues that UAP designers should consider to improve UAP performance and usability.

This chapter contains the following sections:

4.1 Using indexes
4.2 Manipulation of tables
4.3 Stored procedures and stored functions
4.4 Triggers
4.5 SQL optimization
4.6 Data guarantee levels
4.7 Block transfer facility
4.8 Facilities using arrays
4.9 Rapid grouping facility
4.10 Multi-connection facility
4.11 Narrowed search
4.12 File output facility for BLOB data
4.13 Partial update and retrieval of BLOB and BINARY data
4.14 Retrieve first n records facility
4.15 Automatic reconnect facility
4.16 Locator facility
4.17 Facility for returning the total number of hits
4.18 Retrieval, updating, or deletion with an RDAREA name specified
4.19 Automatic numbering facility

4.1 Using indexes

An index is a key created on the basis of values in a specified column in a table to improve processing speed during data retrieval. Using indexes can reduce the number of I/O operations during retrievals.

This section explains the implications on UAP design of using indexes. For details about index definition, see the *HiRDB Version 9 Installation and Design Guide*.

4.1.1 Indexes and processing time

(1) Benefits of using indexes

An index reduces the number of rows to be retrieved, thus reducing processing time. When a multicolumn index is used, fewer I/O accesses to the database are required than when a single index is used.

Retrieval performance is improved when an index is set on the following columns:

- Columns to be used as the condition for narrowing data
- Columns to be used for joining tables
- Columns to be used for sorting or grouping

When many rows are to be updated or when the rows to be retrieved cannot be narrowed, the benefits of defining and using an index are not realized.

In the following cases, the benefits of using an index are not realized, because the number of rows to be retrieved cannot be reduced:

- No search condition is specified
- Many of the rows have the null value or the default value in the column

(2) Drawbacks of using indexes

Because all related indexes are updated during data addition, updating, or deletion, the number of indexes affects processing efficiency. Processing time increases and efficiency declines, unless the number of indexes is minimized to the required number. An SQL error can also occur.

4.1.2 Index priority

When multiple indexes are defined for a table, the HiRDB system usually uses the indexes sequentially, beginning with the one that defines the most efficient condition to narrow the number of retrieved rows. However, depending on the search conditions, the HiRDB system might first use the index it judges to be the most appropriate, regardless of the priority.

There is no need to consider how to retrieve data from a table when creating a UAP.

However, to ensure that HiRDB selects the index best suited to the system, define an index for the column for which the search condition is specified.

4.1.3 Changing indexes during retrieval

During table manipulation, it is possible to add new indexes or to change the configuration of an index to change the processing or to improve retrieval efficiency. However, adding unnecessary indexes reduces retrieval efficiency.

If a UAP is running using a schema, you cannot add or delete an index for the table in the schema.

4.1.4 Notes about index searches

This subsection describes the internal processing for making changes to an index (index maintenance that follows data change), and provides guidelines for UAP design with respect to index searches.

Indexes are used as an efficient means of narrowing down the rows that satisfy specified search conditions, as well as providing high-speed data access and return of search results.

HiRDB achieves a high-response, high-throughput system by enabling multiple transactions to concurrently perform index searches and index changes. However, if searches using indexes are performed while the indexes are being changed, the search results might be affected. One method for preventing inconsistent results is to lock the target table during a search (by executing the `LOCK TABLE` statement). However, this method cannot always be used due to system performance requirements.

If you run applications that execute multiple transactions concurrently and that require a precise sequencing of events, you should apply the UAP design guidelines discussed in this subsection when you develop your applications.

In addition, you should perform index searches with caution when you update columns that compose a multicolumn index (such as a status column).

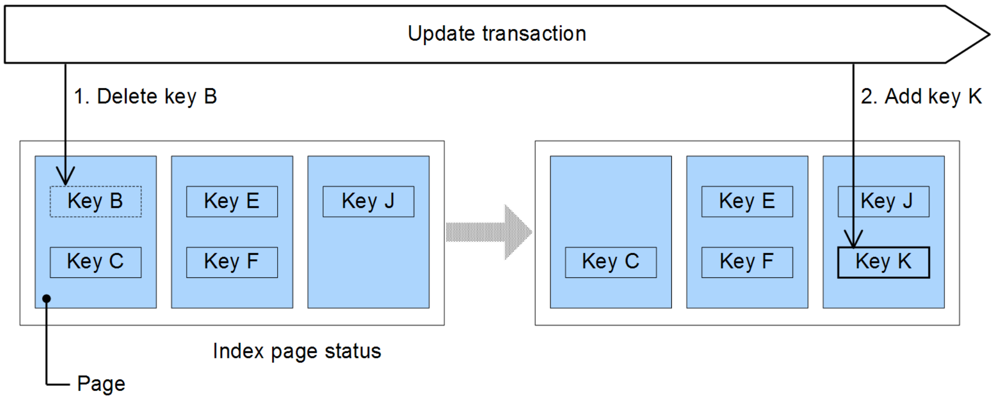
(1) Internal processing for index change

An index change resulting from a change to a value in an index configuration column (`UPDATE` statement) is achieved by two processes, which are the index entry change processing for the pre-update column value and the index entry change processing for the post-update column value.

Index entry change processing for the pre-update column value means deletion of the index key when a row's data item contains the corresponding index key. The index entry change processing for the post-update column value means addition of an index key when a row's data item contains the corresponding index key. Deletion of an index key and addition of an index key are performed in this order in order to prevent an increase in the size of the index during index change processing.

The following figure shows an example of the internal processing for index change.

Figure 4-1: Example of internal processing for index change



Explanation:

This example executes a transaction that updates from B to K the value in a column that has been indexed.

In step 1, the update transaction deletes index key B that corresponds to the pre-update column's value.

In step 2, index key K that corresponds to the post-update column's value is added.

(2) Results of index searches

If you perform an index search while index change processing is underway, the search results might differ. The following are two such cases:

- The row being updated is excluded from the search target.
- The row to be updated appears more than once in the search results.

(a) Case where the row being updated is excluded from the search target

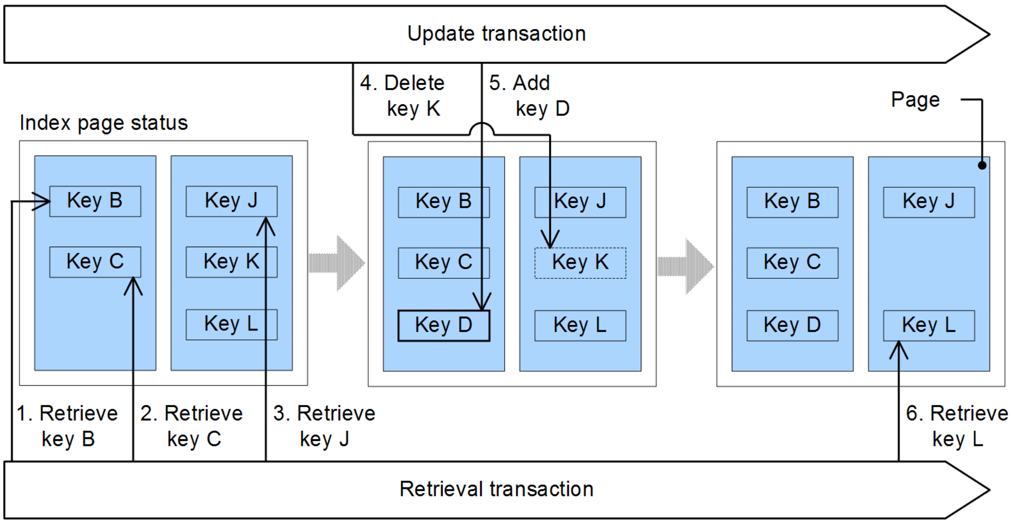
This subsection discusses the case where the row being updated is excluded from the search target.

■ Updating indexes from the pre-search range to the post-search range

If you use the `UPDATE` statement to update an index key from the range for which an index search is currently underway to a range for which an index search has been completed, that index key will be excluded from the search target.

The following figure shows an example of a case where the row being updated is excluded from the search target.

Figure 4-2: Example of case where the row being updated is excluded from the search target (1)



Explanation:

This example executes a transaction that updates from K to D the value in a column that has been indexed; it does this at the same time that another transaction is retrieving the index.

Index retrieval steps 1 through 3 (key J) have been completed.

When key K is retrieved, the update transaction deletes it (step 4) and adds key D (step 5); therefore, the corresponding rows are excluded as a search target.

Supplement:

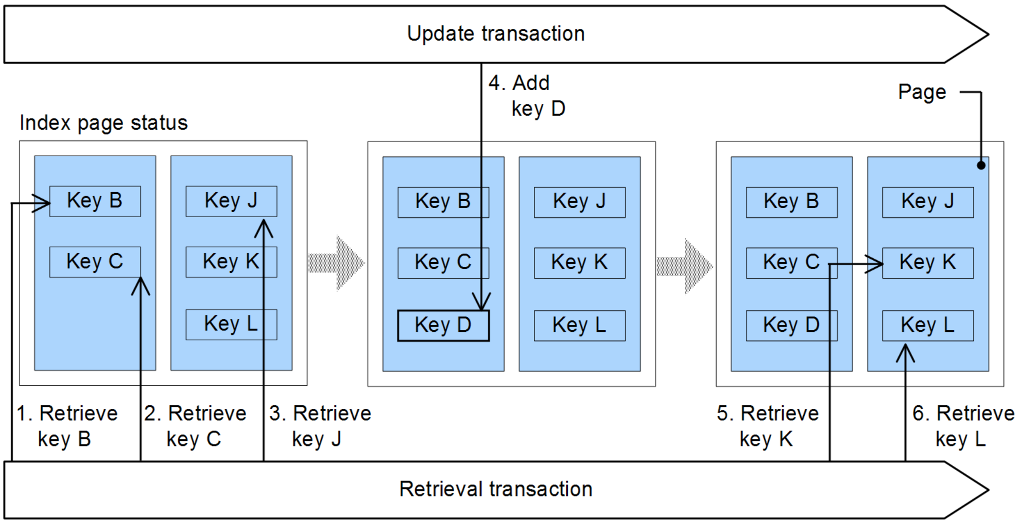
If the index search condition applies to all index keys (for a multicolumn index, all configuration columns), there is no problem because the search target rows no longer satisfy the search condition due to the `UPDATE` statement. However, if the search condition applies to the configuration columns of a multicolumn index that are not to be changed, a problem might arise, depending on the application. For details about countermeasures, see *4.1.4(4) UAP design guidelines*.

■ Adding index keys to the post-search range

If you execute the `INSERT` statement before you perform an index search, but add an index key to the range in which the index search has been completed, that index key is excluded from the search target.

The following figure shows an example of a case where the row being updated is excluded from the search target.

Figure 4-3: Example of case where the row being updated is excluded from the search target (2)



Explanation:

This example executes a transaction that adds a row containing D in a column that has been indexed; it does this at the same time that another transaction is retrieving the index.

Index search step 3 (retrieval of key J) has already been completed in HiRDB's internal processing when the update transaction adds key D. As a result, the row for the added key D is excluded from the search target.

Supplement:

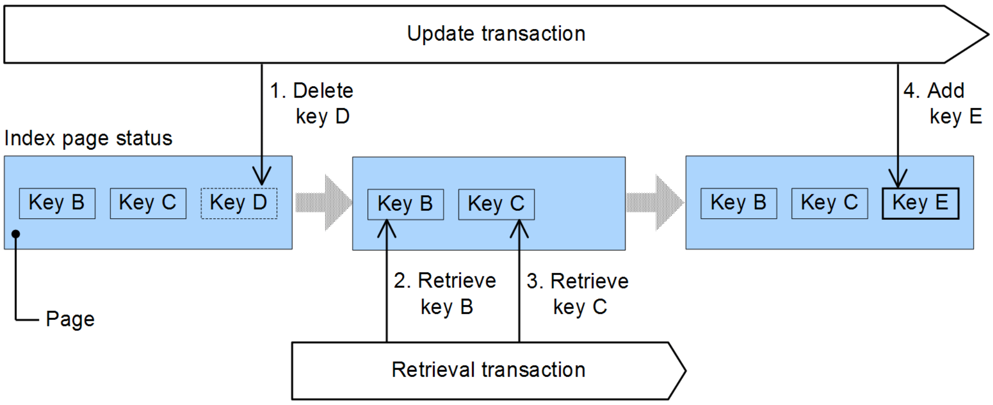
Because the row being updated is excluded from the search target, a problem might arise in applications that demand precise sequencing of events. For details about countermeasures, see *4.1.4(4) UAP design guidelines*.

■ Changing index keys with the UPDATE statement

If, during index change processing, an index search is performed after the index key has been deleted, the corresponding row is excluded from the search target.

The following figure shows an example of a case where the row being updated is excluded from the search target.

Figure 4-4: Example of case where the row being updated is excluded from the search target (3)



Explanation:

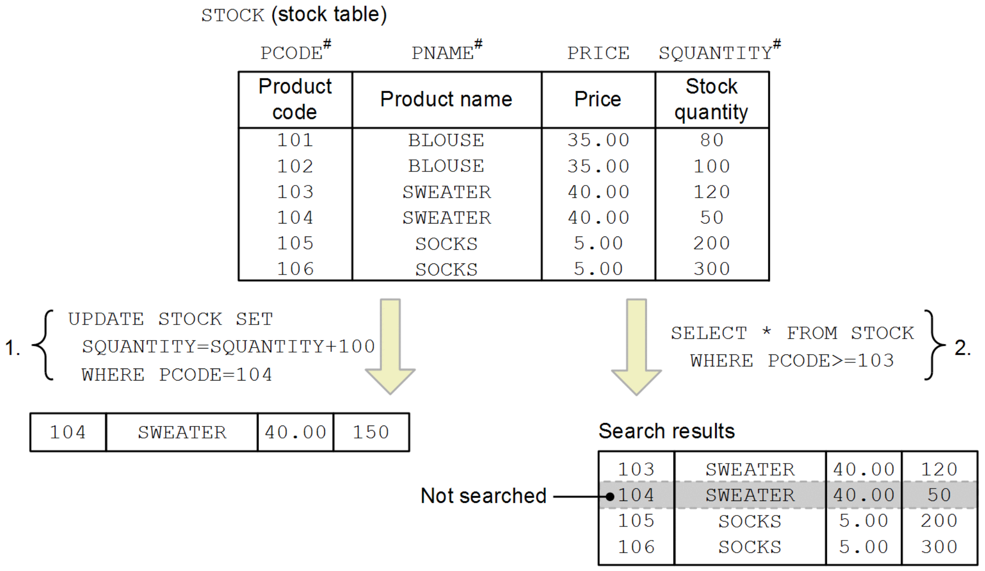
This example executes a transaction that updates from D to E a column that has been indexed; it does this at the same time that another transaction is retrieving the index.

By the time the update transaction has deleted key D in step 1, the retrieval transaction has already completed retrieval of keys B and C in steps 2 and 3. Therefore, the row corresponding to the added key E in step 4 is excluded from the search target.

Supplement:

- If the pre-update index key is the same as the post-update index key, the index is not changed (there is no index key deletion or addition) unless the index satisfies any of the following conditions:
  - The index includes a variable-length string-type configuration column with a defined length of 256 bytes or more.
  - The index has a repetition column as a configuration column
  - It is a substructure index.
- If the index search condition applies to all index keys, there is no problem because the search target rows no longer satisfy the search condition due to the `UPDATE` statement. However, if the search condition applies to the configuration columns of a multicolumn index that are not to be changed, a problem might arise, depending on the application. The following figure shows an example of a case where rows are no longer searched due to a change made to the index keys by the `UPDATE` statement.

Figure 4-5: Example of case where rows are no longer searched due to a change made to the index keys by the UPDATE statement



STOCK (stock table)

PCODE#	PNAME#	PRICE	SQUANTITY#
Product code	Product name	Price	Stock quantity
101	BLOUSE	35.00	80
102	BLOUSE	35.00	100
103	SWEATER	40.00	120
104	SWEATER	40.00	50
105	SOCKS	5.00	200
106	SOCKS	5.00	300

1.
```
UPDATE STOCK SET
  SQUANTITY=SQUANTITY+100
  WHERE PCODE=104
```

104	SWEATER	40.00	150

2.
```
SELECT * FROM STOCK
  WHERE PCODE>=103
```

Search results

103	SWEATER	40.00	120
104	SWEATER	40.00	50
105	SOCKS	5.00	200
106	SOCKS	5.00	300

#: Columns making up a multicolumn index

Explanation:

This example adds the value 100 to the stock quantity because 100 items whose product code is 104 have been delivered.

The example uses a multicolumn index that consists of PCODE, PNAME, and SQUANTITY for searches. In this case, the row being updated in step 1 is not included in the search results because it is not a search target.

For details about countermeasures, see *4.1.4(4) UAP design guidelines*.

(b) Case where the row to be updated appears more than once in the search results

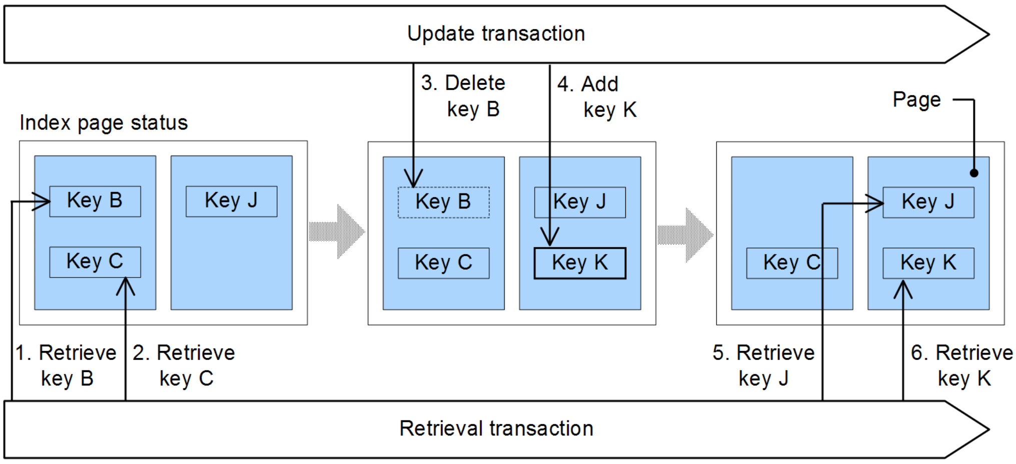
This subsection discusses the case where the row to be updated appears more than once in the search results.

■ Updating indexes from the post-search range to the pre-search range

If WITHOUT LOCK WAIT is specified as the lock option, the row to be updated might appear more than once in the search results because the search results are not guaranteed until transactions are completed. The same applies to the WITHOUT LOCK NOWAIT lock option. Specifically, if you use the UPDATE statement to update an index key from a range for which the index search has been completed to a range for which the index search has not been completed, that index key will be searched again.

The following figure shows an example of a case where the row to be updated appears more than once in the search results.

Figure 4-6: Example of case where the row to be updated appears more than once in the search results



Explanation:

This example executes a transaction that updates from B to K a column that has been indexed; it does this at the same time that another transaction is retrieving the index.

When key B is retrieved in step 1, the update transaction can update the corresponding rows because lock has been released by the lock option.

Before the retrieval transaction retrieves key J in step 5, the update transaction had already updated the corresponding rows (by deleting key B in step 3 and adding key K in step 4). Therefore, the corresponding updated row appears for the second time in the search results in step 6 (retrieval of key K).

Supplement:

This causes no problem if the search results do not need to be precise (such as for statistical information), but it might be a problem if the search results are to be used by other applications that require highly accurate data. For details about countermeasures, see *4.1.4(4) UAP design guidelines*.

(3) Index search processing

The following table shows the possibility of obtaining different search results when an index is searched while the `INSERT`, `UPDATE`, and `DELETE` statements are executing.

Table 4-1: Possibility of obtaining different search results

Subsequent processing (search using index)	Prior processing		
	INSERT statement	UPDATE statement	DELETE statement
`SELECT` statement	MC	MC#	NC
`UPDATE` statement	MC	MC	NC
`DELETE` statement	MC	MC	NC

Legend:

NC: Search results do not change.

MC: Search results might change (search target rows might be excluded from the search).

#

If the `WITHOUT LOCK WAIT` or `WITHOUT LOCK NOWAIT` lock option is used, a row to be updated might appear more than once in the search results.

If the subsequent processing is the `INSERT` statement, there is no possibility of getting different search results because the `INSERT` statement does not use indexes for search processing.

Whether a row to be updated appears more than once in the search results depends on how the index key values before and after update processing are changed. If the direction of the index key value change that takes place when the update transaction updates the index is the same as the direction of the index search by the retrieval transaction, the update target row might appear more than once in the search results, as shown in the following table:

Direction of index search	Direction of index key value change by the UPDATE statement		
	Index key value becomes larger	Index key value becomes smaller	Index key value remains the same (same-value update)
Ascending order (from smaller key value to larger key value)	MC	NC	NC
Descending order (from larger key value to smaller key value)	NC	MC	NC

Legend:

NC: Search results do not change.

MC: Search results might change (the same row might appear more than once in the search results).

(4) UAP design guidelines

If you execute multiple transactions concurrently and develop an application that requires a precise sequence of events, you must lock the applicable table for the UAP and serialize index changes and index search processes. If this affects performance, evaluate the following measures:

1. Do not include in the index configuration columns any items to be updated.
2. If an item to be updated must be included in the index configuration columns for search frequency and condition reasons, evaluate whether only the index configuration columns that do not contain the item to be updated can be used for search conditions during index searching without causing any problem in the application. If there would be a problem, do not use this index in those search conditions.
3. If measure 2 above cannot be employed, include search results re-check processing in the UAP.

4.2 Manipulation of tables

4.2.1 Tables with the FIX attribute

The rows in a table with the `FIX` attribute are fixed in length. Thus, when a table has many columns, processing efficiency is improved if the table is assigned the `FIX` attribute. In a sense, the entire row is manipulated as a single column. This is called *manipulation on a row basis*.

Manipulation of a table on a row basis provides the following advantages over manipulation on a column basis:

- Processing time is shorter.
- Processing time is unaffected by increases in the number of columns to be processed.
- Because one row can be transferred as one item of data, UAPs are easy to create and maintain.

If a table is assigned the `FIX` attribute, manipulating the table on a row basis rather than a column basis improves the processing efficiency for the following operations:

- When all or most of the columns will be retrieved
- When all or most of the columns will be updated
- When data will be inserted

Because an entire row is the target of manipulation, the embedded variables for transferring data must be re-declared if a column is added to the table.

A table with the `FIX` attribute cannot contain variable-length columns or null values. Therefore, manipulation on a row basis can be executed only for a table with no variable-length columns or null values. If manipulating an entire row as a single column will improve efficiency (particularly when there are many columns), consider eliminating columns with variable-length data and columns with null values; make the table attribute `FIX`. Use the following methods to do this:

- Look for short variable-length columns or variable-length columns where only a restricted portion stores data: convert them to fixed-length columns.
- Replace null values with some other values (that is, `0` for numeric data and spaces for character data).

4.2.2 Tables used in numbering

There are two numbering methods:

- Using a table for which the `WITHOUT ROLLBACK` option is specified

- Using the automatic numbering facility

This subsection describes numbering using a table for which the `WITHOUT ROLLBACK` option is specified. For details about the automatic numbering facility, see *4.19 Automatic numbering facility*.

(1) When to use numbering

In actual applications, there are various types of numbering for purposes such as managing form and document numbers. When a user attempts to acquire a form number, another user might also be acquiring a form number at the same time.

When a user attempts to acquire a form number, you must keep track of the form numbers so that the same number is not assigned to two users.

In such a case, one user might be placed on wait status while the other user is acquiring a number. HiRDB provides functions that minimizes the effects of locking while you perform numbering.

(2) Designing the table

In order to perform efficient numbering, you must design the table so as to minimize the effects of locking. HiRDB provides a function that releases locks from table rows and prevents rollback when update processing (including addition and deletion) on the table is completed without having to wait for transmissions to commit. To implement this function, the table designer must specify the `WITHOUT ROLLBACK` option in `CREATE TABLE` when the table is designed.

(3) Condition of application to jobs

When the `WITHOUT ROLLBACK` option is specified during table definition, rollback will not occur when rows are updated. If the UAP or HiRDB system terminates abnormally, the correct rollback will be achieved and data integrity will be maintained for the table for a job that uses assigned numbers when the HiRDB system is restarted, but the specific point in the numbering management table update processing at which rollback occurred cannot be determined. In such a case, the numbers assigned by the numbering process might no longer be used by the job. Therefore, such an application is not suitable for a job that requires consecutive numbers. Apply this method to a job that does not require consecutive numbers.

(4) Example of a table used for managing numbering

The following figure shows an example of a table used for managing numbering.

Figure 4-7: Example of a table used for managing numbering

Number management table

TYPE	NUMBER
Form number	23
Document number	17

Operation table (table that uses form numbers)

FORM NUMBER	ITEM NAME
1	AAA
2	BBB
⋮	⋮
23	WWW
⋮	⋮

Note

For examples of table definitions (`WITHOUT ROLLBACK` option), see the *HiRDB Version 9 Installation and Design Guide*.

(5) Example of a numbering application program

This subsection presents an example of a numbering application program. This example assumes the same transactions for an application program that uses the numbering management table and job table.

Example:

There is a numbering management table that manages form numbers and document numbers. The following shows an example SQL statement that acquires the most recent form number from the numbering management table and uses that number in a job.

```
  INSERT INTO NUMBERING_MANAGEMENT_TABLE VALUE('FORM_NUMBER',1)
....1

                 :

  DECLARE CUR1 CURSOR FOR  .......................2
    SELECT NUMBERING FROM NUMBERING_MANAGEMENT_TABLE
      WHERE TYPE='FORM_NUMBER' FOR UPDATE OF NUMBERING
  OPEN CUR1  ....................................3
  FETCH CUR1 INTO :x_NUMBERING  ......................4
  UPDATE NUMBERING_MANAGEMENT_TABLE SET
NUMBERING=:x_NUMBERING+1  ..........5
      WHERE CURRENT OF CUR1
  CLOSE CUR1  ..................................6

                 :
```

```
    Accessing-the-job-table-by-using-the-acquired-number    . . 7
                          :
```

Explanation:

1. Inserts 1 in the numbering management table as the initial value for the form numbers.
2. Declares the cursor `CUR1` used to retrieve the most recent form number from the numbering management table.
3. Opens the cursor `CUR1`.
4. Retrieves the form number in `x_NUMBERING`.
5. Increments the number (from the most recent number) for the next user to retrieve a form number. When this processing is finished, the row is released from locked status without waiting for commit.
6. Closes the cursor `CUR1`.
7. Performs user-defined processing on the basis of the form number retrieved in `x_NUMBERING`.

Steps 3 through 7 are repeated each time numbering is performed.

(6) Notes about managing more than one type of numbering

(a) About locking

If multiple rows are stored in a table for which the `WITHOUT ROLLBACK` option is specified, and no index is defined for the table, all rows are locked temporarily because they are all subject to search. In such a case, locking might occur between, for example, a form numbering process and a document numbering process. You can prevent this by specifying `YES` in `PDLOCKSKIP` in the client environment definition to perform an unlocked conditional search. If an unlocked conditional search has been performed, only the rows satisfying the search condition are locked and no lock is applied during search processing.

(b) About rollback

If you perform more than one type of numbering, do not update multiple rows with a single SQL statement. The timing at which lock release and rollback no longer occur is set to the point in time where update processing on each row is completed. If a UAP that updates multiple rows terminates abnormally, some of the rows might not be rolled back.

(7) Examples of numbering using stored procedures

Because numbering is often performed in a specific pattern, it is convenient if you register a numbering process as a stored procedure.

This subsection presents three examples of table definition and stored procedures.

Example 1:

This example assigns sequential numbers by using a table with WITHOUT ROLLBACK specified and a stored procedure.

It assigns numbers up to the maximum value of INTEGER, whose initial value is 1 and increment value is 1.

If the maximum value of INTEGER is exceeded, an overflow error is returned. Note that if the default value setting facility (PDDFLNVAL) is used, the null value is assumed (no overflow error occurs), resulting in a NOT NULL constraint violation. If no row containing the initial value has been inserted, the table is treated as being empty, in which case if the UPDATE statement is executed, an error is caused by the cursor not being positioned in a row. If multiple rows have been inserted, only the first row is used and any subsequent rows are ignored.

```
CREATE FIX TABLE
    owner_id.sequence_tbl(sequence_no INTEGER NOT NULL)
    WITHOUT ROLLBACK;
.......................................1
CREATE PROCEDURE owner_id.nextval(OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;
..............................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl
        FOR UPDATE;
    OPEN cr1;
    FETCH cr1 INTO update_no;
...............................3
    SET next_no=update_no;
..................................4
    UPDATE owner_id.sequence_tbl SET sequence_no=update_no+1
      WHERE CURRENT OF cr1;
.................................5
    CLOSE cr1;
..............................................3
  END
.......................................................2
COMMIT WORK;
................................................6
INSERT INTO owner_id.sequence_tbl(sequence_no) VALUES(1);
```

```
...7
COMMIT WORK;
..............................................8

<Assigning sequential numbers>  ..................................9
CALL owner_id.nextval(OUT:xnext_no);
                :
processing-using-assigned-sequential-number-xnext_no
                :
CALL owner_id.nextval(OUT:xnext_no);
                :
```

Explanation:

1. Defines the table `owner_id.sequence_tbl` used to assign the value of `INTEGER`.
2. Defines the procedure `owner_id.nextval` that assigns a sequential number and outputs it by using the `next_no` parameter.
3. Searches the table `owner_id.sequence_tbl` for the column `sequence_no`.
4. Places the retrieved value in the `next_no` parameter.
5. Updates the value of the column `sequence_no` in the table `owner_id.sequence_tbl` by adding the increment value 1 to it.
6. Commits the transaction to apply the table and procedure definitions.
7. Uses the `INSERT` statement to insert a row whose initial value is 1.
8. Commits the transaction to apply the inserted row.
9. Calls the procedure `owner_id.nextval` by using the `CALL` statement, assigns a sequential number, and then acquires the value by using the `next_no` parameter. Each time the `CALL` statement is executed, the next sequential number is assigned.

Example 2:

This example assigns more than one type of sequential number by using a table with `WITHOUT ROLLBACK` specified and a stored procedure.

For each key that identifies the sequence number, this example assigns numbers up to the maximum value of `INTEGER`, whose initial value is 1 and increment value is 1.

If the maximum value of `INTEGER` is exceeded, an overflow error is returned. Note that if the default value setting facility (`PDDFLNVAL`) is used, the null value is assumed (no overflow error occurs), resulting in a `NOT NULL` constraint violation. If no row containing the initial value has been inserted for the key value used to identify sequential numbers, the table is treated as being empty, in which case if the UPDATE statement is executed, an error is caused by the cursor not being positioned in a row. If multiple rows have been inserted for the key value used to identify sequential numbers, only the first row is used and any subsequent rows are ignored.

Note 1

No index can be defined for a table with WITHOUT ROLLBACK specified. In order to prevent lock contention, you must specify PDLOCKSKIP=YES in the client environment definition.

Note 2

Because no index can be defined for a table with WITHOUT ROLLBACK specified, if you use many different types of sequential numbers, provide multiple tables and procedures.

```
CREATE FIX TABLE
    owner_id.sequence_tbl(sequence_key CHAR(30) NOT NULL,
                          sequence_no INTEGER  NOT NULL)
    WITHOUT ROLLBACK;
..........................................1
CREATE PROCEDURE owner_id.nextval(IN input_key CHAR(30),
                                  OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;
.................................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl
        WHERE sequence_key=input_key FOR UPDATE OF
sequence_no;
    OPEN cr1;
    FETCH cr1 INTO update_no;
..................................3
    SET next_no=update_no;
.....................................4
   UPDATE owner_id.sequence_tbl SET sequence_no=update_no+1
      WHERE CURRENT OF cr1;
....................................5
    CLOSE cr1;
..................................................3
  END
...........................................................2
COMMIT WORK;
...................................................6
INSERT INTO owner_id.sequence_tbl(sequence_key,sequence_no)
    VALUES('key_value_1',1);
```

```
.................................7
COMMIT WORK;
..................................................8
INSERT INTO owner_id.sequence_tbl(sequence_key,sequence_no)
    VALUES('key_value_2',1);
.................................7
COMMIT WORK;
..................................................8
        :
```

(*Insert as many rows containing an initial value as there are types of sequential numbers*)

<Assigning sequential number to 'key_value_1'>

```
............................9
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                  :
```

processing-using-sequential-number-xnext_no-assigned-to-'key_value_1'

```
                  :
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                  :
```

<Assigning sequential number to 'key_value_2'>

```
............................9
xinput_key <-- 'key_value_2'
CALL owner_id.nextval(IN :xinput_key,OUT:xnext_no);
                  :
```

processing-using-sequential-number-xnext_no-assigned-to-'key_value_2'

```
                  :
xinput_key <-- 'key_value_2'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
                  :
```

Explanation:

1. Defines the table `owner_id.sequence_tbl` used to assign the value of `INTEGER` for each key that identifies sequence numbers.
2. Defines the procedure `owner_id.nextval` that inputs a key for identifying sequential numbers by using the `input_key` parameter. Then assigns a sequential number and outputs it by using the `next_no` parameter.
3. Specifies a key used to identify sequential numbers for the column `sequence_key` in the table `owner_id.sequence_tbl` and then searches the column `sequence_no`.

4. Places the retrieved value in the next_no parameter.
5. Updates the value of the column sequence_no in the table owner_id.sequence_tbl by adding increment value 1 to it.
6. Commits the transaction to apply the table and procedure definitions.
7. Uses the INSERT statement to insert a row whose initial value is 1 for each key that identifies sequential numbers.
8. Commits the transaction to apply the inserted row.
9. Calls the procedure owner_id.nextval by using the CALL statement, assigns a sequential number, and then acquires the value by using the next_no parameter. Each time the CALL statement is executed, the next sequential number is assigned.

Example 3:

This example assigns sequential numbers by rotating numbers between minimum and maximum values and using a table with WITHOUT ROLLBACK specified and a stored procedure.

If no row containing the initial value has been inserted, the table is treated as empty, in which case if the UPDATE statement is executed, an error is caused by the cursor not being positioned in a row. If multiple rows have been inserted, only the first row is used and any subsequent rows are ignored.

```
CREATE FIX TABLE
     owner_id.sequence_tbl(sequence_no INTEGER NOT NULL)
     WITHOUT ROLLBACK;
.........................................1
CREATE PROCEDURE owner_id.nextval(OUT next_no INTEGER)
  BEGIN
    DECLARE update_no INTEGER;
.................................2
    DECLARE cr1 CURSOR FOR
      SELECT sequence_no FROM owner_id.sequence_tbl FOR
UPDATE;
    OPEN cr1;
    FETCH cr1 INTO update_no;
..................................3
    SET next_no=update_no;
.....................................4
    IF update_no=2147483647 THEN
      SET update_no=-2147483648;
    ELSE
      SET update_no=update_no+1;
    END IF;
....................................................5
```

```
    UPDATE owner_id.sequence_tbl SET sequence_no=update_no
      WHERE CURRENT OF cr1;
....................................6
    CLOSE cr1;
..................................................3
  END
...........................................................2
COMMIT WORK;
...................................................7
INSERT INTO owner_id.sequence_tbl(sequence_no)VALUES(1);
.......8
COMMIT WORK;
...................................................9

<Assigning sequential numbers >
.........................................10
CALL owner_id.nextval(OUT:xnext_no);
                :
processing-using-assigned-sequential-number-xnext_no
                :
CALL owner_id.nextval(OUT:xnext_no);
                :
```

Explanation:

1. Defines the table `owner_id.sequence_tbl` used to assign the value of `INTEGER`.
2. Defines the procedure `owner_id.nextval` that assigns sequential numbers to the column `sequence_no` in the table `owner_id.sequence_tbl` by rotating numbers whose minimum value is -2,147,483,648, maximum value is 2,147,483,647, and increment value is 1, in such a manner that the value is reset to the minimum value once the maximum value is reached.
3. Searches the column `sequence_key` in the table `owner_id.sequence_tbl`.
4. Places the retrieved value in the `next_no` parameter.
5. If the retrieved value is the maximum value (2,147,483,647), the next value will be the minimum value -2,147,483,648; otherwise, it will be the retrieved value plus 1 (increment value).
6. Updates the sequence number in the column `sequence_no` in the table `owner_id.sequence_tbl` to the next value.
7. Commits the transaction to apply the table and procedure definitions.

8. Uses the INSERT statement to insert a row whose initial value is 1.
9. Commits the transaction to apply the inserted row.
10. Calls the procedure owner_id.nextval by using the CALL statement, assigns a sequential number, and then acquires the value by using the next_no parameter. Each time the CALL statement is executed, the next sequential number is assigned.

4.2.3 Tables using character sets

If you define character sets, you can store character string data using a different character set in each table column.

(1) Transferring character string data for which a character set is specified

This subsection describes an example that transfers character string data for which a character set is specified.

Example:

This example searches column C2 (character set EBCDIK) in table T1. The following shows table T1 and the retrieval SQL statement.

- Table T1 is defined as follows (underline indicates the character set specification):

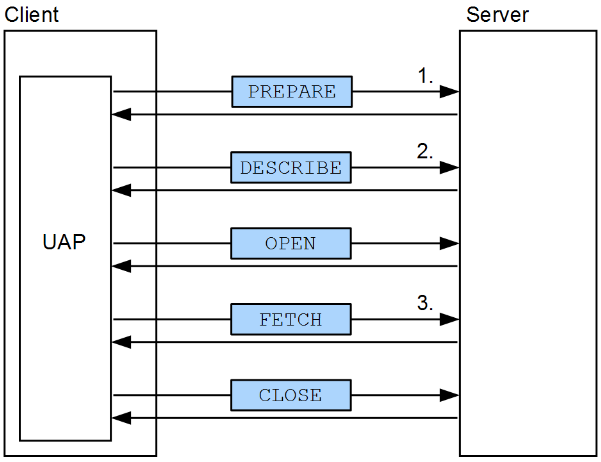
```
CREATE TABLE T1
  (C1 INT, C2 CHAR(30) CHARACTER SET EBCDIK)
```

- The example uses the following retrieval SQL statement:

```
<Declaration of embedded variable>

char DATA[31];
     :
SELECT C2 FROM T1
  WHERE C2 = :DATA
```

The following figure shows how data is transferred.

Figure 4-8: Transfer of character string data specifying a character set



Explanation:

1. Acquires character set information for column `C2` from the data dictionary table and stores it in the SQL object.
2. Sets the character set name for `C2` in the character set descriptor area#.
3. If the character set in the input variable differs from the character set in the character set descriptor area, the character set is converted and then assigned.

   If the character set in the output variable differs from the character set in the character set descriptor area, the character set is converted and then stored.

#

This area contains the character set name in a variable (`DATA`) that was determined dynamically during UAP execution. The contents of the character set descriptor area are used to perform the following processing:

- The character set name specified by the client is sent to the HiRDB server.
- The client receives the items retrieved by the SQL statement preprocessed by the HiRDB server and the character set name in the `?` parameter.

For details about the character set descriptor area, see *Appendix E. Character Set Descriptor Area*.

(2) Character code conversion

If the client and server use different character sets, the server converts the character codes. If the client and server both use a predefined character set, character codes are not converted. However, if PDCLTCNVMODE is specified in the client environment definition, character codes are converted according to the specification.

The following table shows character code conversion between client and server, depending on whether character sets are defined.

Table 4-2: Character code conversion between client and server

Character codes used on the client		Character codes used on the server							
		SJIS		UJIS	UTF-8		LANG-C	CHINESE	CHINESE-GB18030
		UD	EK	UD	UD	U16	UD	UD	UD
SJIS	UD	--	SVR	CLT	CLT	C-S	UN	N	N
	EK	SVR	--	N	N	N	N	N	N
UJIS	UD	N	N	--	N	N	UN	N	N
UTF-8	UD	N	N	N	--	SVR	UN	N	N
	U16C	N	N	N	SVR	--#	N	N	N
UCS-2	UD	N	N	CLT	CLT	C-S	N	N	N
LANG-C	UD	--	SVR	--	--	SVR	--	--	--
CHINESE	UD	N	N	N	N	N	UN	--	N
CHINESE-GB18030	UD	N	N	N	N	N	N	N	--

Legend:

UD: No character set is defined.

EK: EBCDIK is defined as the character set.

U16: UTF16 is defined as the character set.

U16C: UTF16, UTF-16LE, or UTF-16BE is defined as the character set.

SVR: The server converts character codes.

CLT: The client converts character codes (conversion is specified by using `PDCLTCNVMODE` in the client environment definition).

C-S: The client performs the server's character code conversion (specified by using `PDCLTCNVMODE` in the client environment definition) and the server converts the character codes for the character set.

--: Character code conversion is not needed.

UN: Character code conversion is not needed. However, if `NOUSE` is specified in `PDCLTCNVMODE` in the client environment definition, character code conversion is performed (double-byte characters are not converted).

N: Character code conversion cannot be performed.

#

If the client and server use different endians for the character codes, endian conversion is performed.

(a) Access from ODBC and ADO.NET application programs

The conversion rules shown in *Table 4-2* also apply to access from ODBC and ADO.NET application programs.

Note that the following client versions assume the character set descriptor area as follows:

Client version earlier than 08-05

The client observes the conversion rules shown in *Table 4-2*, assuming that there is no character set descriptor area.

Client version 08-05 or later

If the client's character encoding is UCS-2 and the server's character encoding is UTF-8, the client observes the conversion rules shown in *Table 4-2*, assuming UTF-16LE in the character set descriptor area.

(b) Access from OLE DB application programs and JDBC application programs that use a Type2 JDBC driver

These application programs observe the conversion rules shown in *Table 4-2*, assuming that there is no character set descriptor area.

(c) Access from JDBC application programs that uses a Type4 JDBC driver

If the client version is earlier than 08-05 or the server uses a character encoding that is not UTF-8

The JDBC driver converts character codes to those used by the server by using the encoder provided by Java Virtual Machine (JVM). If the server specifies a character set, then the server converts the character codes.

If the client version is 08-05 or later and the server uses UTF-8 character encoding

If the server specifies the UTF16 character set, UTF-16BE is used to transfer data with the server. If the server does not specify the UTF16 character set, the encoder provided by JVM is used to convert character codes to UTF-8.

4.3 Stored procedures and stored functions

This section explains how to define stored procedures and stored functions.

Be sure to create the necessary RDAREA spaces before defining stored procedures and stored functions. For details about the operation of stored procedures and stored functions, see the *HiRDB Version 9 System Operation Guide*.

You can use SQL, Java, or C language to code stored procedures and stored functions. Those coded in SQL are called SQL stored procedures and SQL stored functions. Those coded in Java are called Java stored procedures and Java stored functions, and those coded in C language are called C stored procedures and C stored functions.

The stored procedures and stored functions are referred to collectively as *stored routines*. A stored routine whose owner is `PUBLIC`, meaning all users, is called a *public routine*.

For details about Java stored procedures and Java stored functions, see *9. Java Stored Procedures and Java Stored Functions*. For details about C stored procedures and C stored functions, see *10. C Stored Procedures and C Stored Functions*.

Note

If an error occurs while an SQL stored procedure or SQL stored function is being executed, processing of the SQL stored procedure or SQL stored function terminates at the point when the error occurred (the programs exits from control of the SQL stored procedure or SQL stored function). Therefore, error-handling processes cannot be specified in SQL stored procedures and SQL stored functions.

4.3.1 Defining a stored procedure

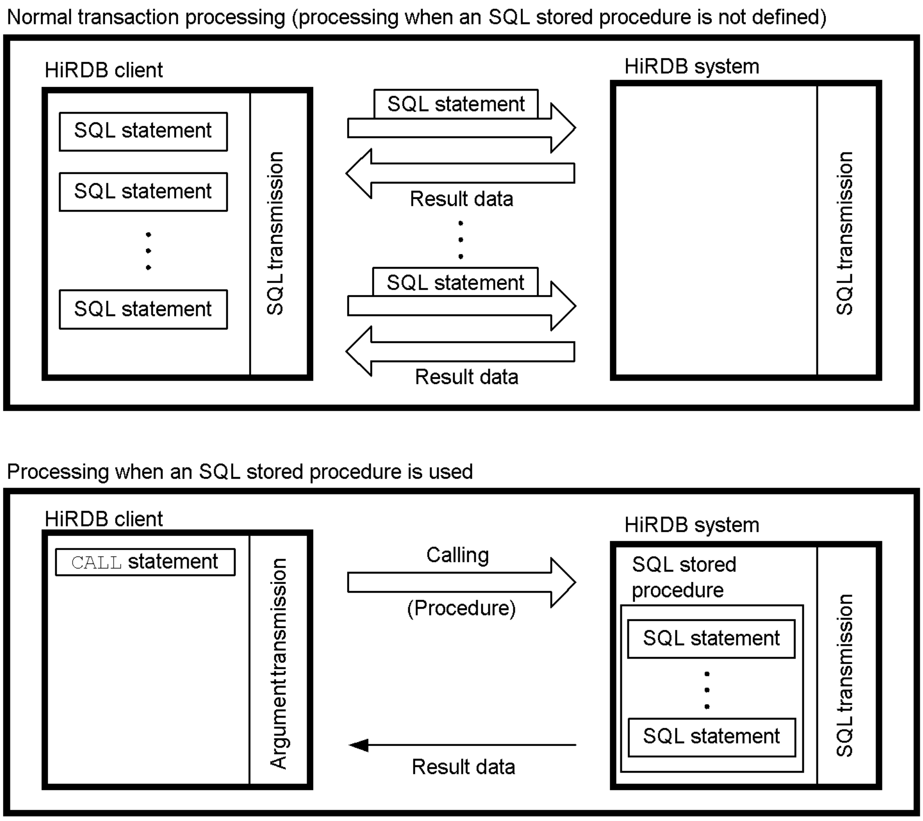
A stored procedure is a facility that registers an SQL-coded database processing procedure to a database as a procedure.

(1) Benefits of using an SQL stored procedure

Manipulating a database may involve searching for data with the `FETCH` statement and then issuing the `UPDATE` or `INSERT` statement, depending on whether or not matching data is found. This process may be repeated many times, resulting in high overhead between the client and the server. This type of database access processing can also be defined in a routine that is stored as a procedure and can then be executed by calling it with a `CALL` statement. Use of a stored procedure reduces the amount of overhead associated with passing and receiving data between the client and the server. Because the SQL statements that are stored in a procedure are stored at the server in a compiled form (as SQL objects), using a stored procedure permits the client and the server to share processing, at the same time reducing the SQL parsing overhead.

The following figure shows the benefits of using an SQL stored procedure.

Figure 4-9: Benefits of using an SQL stored procedure



(2) Defining and executing an SQL stored procedure

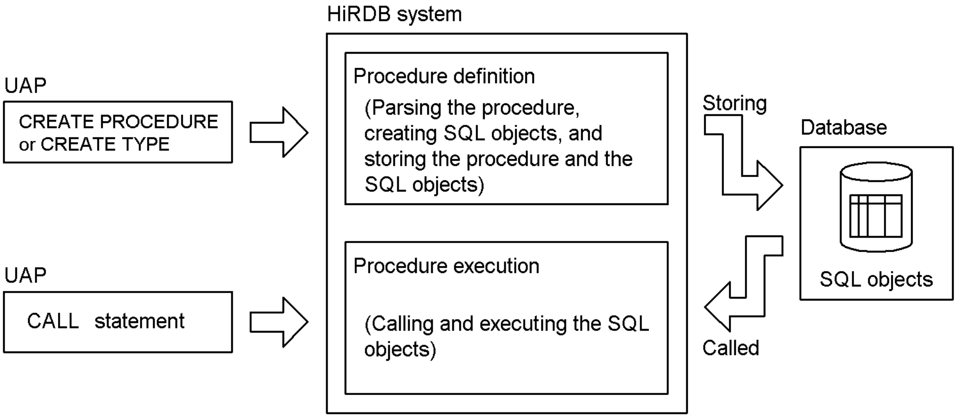
CREATE PROCEDURE or CREATE TYPE stores a defined procedure in a database as an SQL stored procedure; DROP PROCEDURE deletes an SQL stored procedure from the database. Once stored in a database, an SQL stored procedure can be executed by calling it with a CALL statement.

If a procedure has an SQL object that has been invalidated, the ALTER PROCEDURE or ALTER ROUTINE statement can be used to re-create that procedure.

If an SQL stored procedure has already been registered, the pddefrev command can be executed to create definition-type SQL statements for that SQL stored procedure. This command is useful for creating a new SQL stored procedure, the processing of which is similar to that of an existing SQL stored procedure. For details about the pddefrev command, see the *HiRDB Version 9 Command Reference* manual.

The following figure illustrates the definition and execution of an SQL stored procedure.

Figure 4-10: Defining and executing an SQL stored procedure


HiRDB system
UAP
CREATE PROCEDURE or CREATE TYPE
Procedure definition
(Parsing the procedure, creating SQL objects, and storing the procedure and the SQL objects)
Storing
Database
UAP
CALL statement
Procedure execution
(Calling and executing the SQL objects)
SQL objects
Called

Public procedure

If you use a stored procedure defined by another user, you must specify the owner's authorization identifier and routine identifier when you call the stored procedure from within a UAP. However, if `CREATE PUBLIC PROCEDURE` is executed to define the stored procedure as a public procedure, there is no need to specify the owner's authorization identifier when the stored procedure is called from a UAP (only the routine identifier must be specified).

(3) Example of an SQL stored procedure

An example of the definition and execution of an SQL stored procedure that defines SQL statements and statements for controlling the SQL statements (routine control SQL) is shown as follows:

Figure 4-11: Example of an SQL stored procedure

SQL stored procedure definition

```
CREATE PROCEDURE proc_1 (IN fromdate date, IN todate date) -------- 1.
  BEGIN ------------------------------------------------------------ 2.
    DECLARE x_goods_no INTEGER; ------------------------------------ 3.
    DECLARE x_quantity_1, x_total_quantity DECIMAL(17,2); ---------- 3.
    DECLARE cr1 CURSOR FOR
      SELECT goods_no, quantity_1, entrydate FROM tran_t1
        WHERE entrydate BETWEEN fromdate AND todate
        ORDER BY entrydate;

    OPEN cr1;
    while_loop:
    WHILE SQLCODE = 0 DO ------------------------------------------- 4.
      FETCH cr1 INTO x_goods_no, x_quantity_1;
        IF SQLCODE=100 THEN
        LEAVE while_loop; ------------------------------------------ 5.
      END IF;
      SELECT total_quantity INTO x_total_quantity
        FROM master_t1 WHERE goods_no = x_goods_no;
      IF SQLCODE=100 THEN ------------------------------------------ 6.
        INSERT INTO master_t1 VALUES(x_goods_no, x_quantity_1);
      ELSE
        SET x_total_quantity = x_total_quantity + x_quantity_1; ---- 7.
        UPDATE master_t1 SET total_quantity = x_total_quantity
          WHERE goods_no = x_goods_no;
      END IF ------------------------------------------------------- 8.
    END while_loop; ------------------------------------------------ 9.
    CLOSE cr1;
  END; ------------------------------------------------------------- 10.
```

SQL stored procedure call

```
         :
strcpy(e_fromdate, "1996-06-01");
strcpy(e_todate, "1996-06-30");
EXEC SQL CALL proc_1(IN :e_fromdate, IN :e_todate); --------------- 11.
         :
```

Explanation

1. Defines the procedure name and the SQL parameters.
2. Begins compound statements.
3. Declares SQL variables.
4. Specifies repetitive execution of statements.

5. Specifies exiting a statement.
6. Specifies conditional branching.
7. Specifies value assignments.
8. Ends the conditional branch.
9. Ends repetitive executions of statements.
10. Ends the compound statements.
11. Calls the procedure.

Notes

1. For details about the individual SQL statements, see the *HiRDB Version 9 SQL Reference* manual.
2. This example specifies `entrydate` as a selection item in the `SELECT` clause for cursor declaration, so that the data can be sorted according to `entrydate`. However, because `entrydate` values are not referenced, the `FETCH` statement omits the embedded variable corresponding to `entrydate` and does not fetch `entrydate` values.

(4) Debugging an SQL stored procedure

To debug an SQL stored procedure, use `WRITE LINE` statements in a routine control SQL and output the SQL variables and SQL parameters to be referenced to a client file. For details about the `WRITE LINE` statement, see the manual *HiRDB Version 9 SQL Reference*.

An example of specifying `WRITE LINE` statements in an SQL stored procedure is shown below.

```
CREATE PROCEDURE proc_1 (IN fromdate date, IN todate date)
  BEGIN
    ...
    WRITE LINE 'fromdate='||char(fromdate);   ...................1
    WRITE LINE 'todate='||char(todate);   ......................2
    ...
```

Explanation:

1. Converts the value of the `fromdate` SQL parameter to a character string and outputs the string to a file.
2. Converts the value of the `todate` SQL parameter to a character string and outputs the string to a file.

To output the values of the value expressions in the `WRITE LINE` statements from the SQL stored procedure in which the `WRITE LINE` statements were written to a client file, set the PDWRTLNFILSZ client environment definition, and call the SQL stored procedure from the UAP. An example is shown below.

PDWRTLNFILSZ setup example for csh (C shell) (UNIX edition HiRDB client)

```
setenv PDWRTLNFILSZ 4096
```

PDWRTLNFILSZ setup example (Windows edition HiRDB client)

```
PDWRTLNFILSZ=4096
```

Calling the SQL stored procedure:

```
strcpy(e_fromdate, "2003-06-01");
strcpy(e_todate, "2003-06-30");
EXEC SQL CALL proc_1(IN :e_fromdate, IN :e_todate);
```

Contents of output file:

```
fromdate=2003-06-01
todate=2003-06-30
```

Note: The output file is set with PDWRTLNPATH in the client environment definition.

Once debugging is completed, if you no longer need to output the values of the value expressions in the WRITE LINE statements from the SQL stored procedure in which the WRITE LINE statements were specified to a file, omit the PDWRTLNFILSZ client environment definition before executing the UAP. When the PDWRTLNFILSZ specification is omitted, the WRITE LINE statements in the SQL stored procedure are not executed.

(5) Completing a transaction in a stored procedure

(a) SQL statements for completing a transaction

To complete a transaction in a stored procedure, execute one of the following SQL statements in that procedure (note that SQL statements cannot be executed within a C stored procedure):

- COMMIT statement

- ROLLBACK statement

COMMIT is executed automatically when one of the following SQL statements is executed:

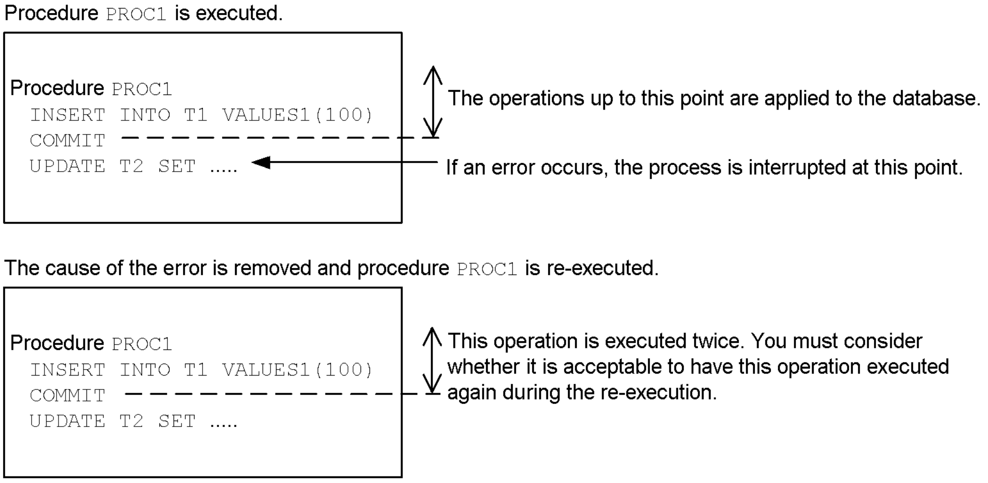
- PURGE TABLE statement
- Definition SQL (in Java stored procedures only)

ROLLBACK is executed automatically when the following condition applies:

- An error that requires ROLLBACK execution occurs.

(b) Notes about re-executing stored procedures

If an error occurs during a stored procedure after a transaction has been completed, execution of the procedure is terminated before completion. If you re-execute the stored procedure that resulted in an error, the procedure processes are executed again from the beginning. You must therefore consider whether the operations performed before the transaction was terminated due to error can be executed twice. An example is shown below.



Procedure PROC1 is executed.

```
Procedure PROC1
  INSERT INTO T1 VALUES1(100)
  COMMIT
  UPDATE T2 SET .....
```

The operations up to this point are applied to the database.
If an error occurs, the process is interrupted at this point.

The cause of the error is removed and procedure PROC1 is re-executed.

```
Procedure PROC1
  INSERT INTO T1 VALUES1(100)
  COMMIT
  UPDATE T2 SET .....
```

This operation is executed twice. You must consider whether it is acceptable to have this operation executed again during the re-execution.

(6) Results-set return facility (limited to SQL stored procedures)

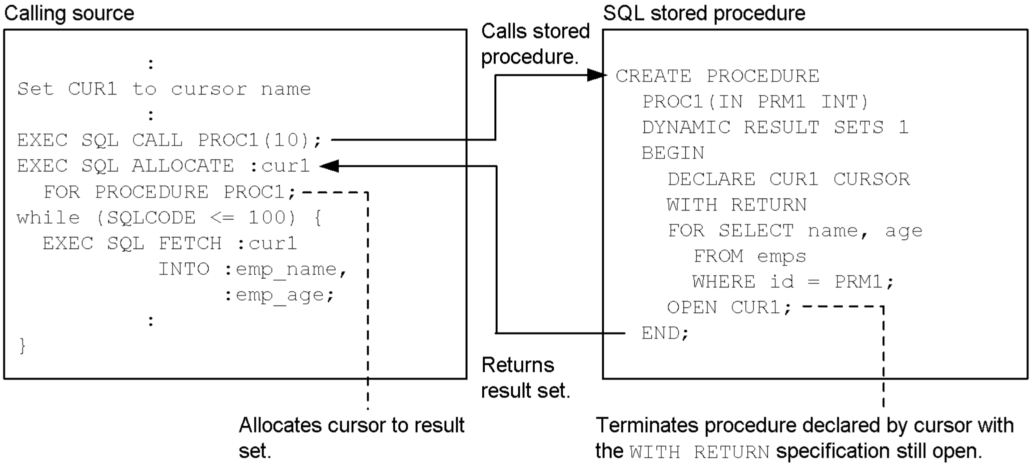
When defining an SQL stored procedure, you can use the results-set return facility by specifying a value of 1 or higher in the DYNAMIC RESULT SETS clause of CREATE PROCEDURE. The results-set return facility cannot be used for SQL stored functions.

(a) What is the results-set return facility?

The results-set return facility allows the calling source of an SQL stored procedure to reference the cursor obtained when the SELECT statement in the SQL stored procedure is executed.

The following figure provides an overview of the results-set return facility.

Figure 4-12: Overview of results-set return facility (for SQL stored procedures)



Calling source

```
          :
Set CUR1 to cursor name
          :
EXEC SQL CALL PROC1(10);
EXEC SQL ALLOCATE :cur1
  FOR PROCEDURE PROC1;
while (SQLCODE <= 100) {
  EXEC SQL FETCH :cur1
           INTO :emp_name,
                :emp_age;
          :
}
```

SQL stored procedure

```
CREATE PROCEDURE
  PROC1(IN PRM1 INT)
  DYNAMIC RESULT SETS 1
  BEGIN
    DECLARE CUR1 CURSOR
    WITH RETURN
    FOR SELECT name, age
      FROM emps
      WHERE id = PRM1;
    OPEN CUR1;
  END;
```

Calls stored procedure.
Returns result set.
Allocates cursor to result set.
Terminates procedure declared by cursor with the WITH RETURN specification still open.

(b) Languages of calling sources that can use the results-set return facility

Listed below are the languages of calling sources that can use the results-set return facility:

- Java
- C
- C++
- COBOL#
- OOCOBOL

#: COBOL can be used if an RDB file input/output function is not used.

(c) Example of using the results-set return facility

In this example, the SQL stored procedure searches the `emps_1` and `emps_2` tables and retrieves `id`, `name`, and `age` data for `id` column values that satisfy the condition `id<10`. The calling source accepts the two result sets and executes them.

Definitions of the SQL stored procedure

```
CREATE PROCEDURE proc2(IN param1 INTEGER) ............... 1
  DYNAMIC RESULT SETS 2 ................................. 2
  BEGIN
    DECLARE CUR1 CURSOR WITH RETURN ..................... 3
       FOR SELECT id,name,age FROM emps_1
            WHERE id < param1 ORDER BY id;
    DECLARE CUR2 CURSOR WITH RETURN ..................... 4
       FOR SELECT id,name,age FROM emps_2
            WHERE id < param1 ORDER BY id;
    OPEN CUR1; .......................................... 5
    OPRN CUR2; .......................................... 6
  END; .................................................. 7
```

Explanation:

1. Defines the procedure name and the parameter.
2. Specifies the number of search result information sets to be returned.
3. Declares the `CUR1` cursor.
4. Declares the `CUR2` cursor.
5. Opens the `CUR1` cursor.
6. Opens the `CUR2` cursor.
7. Terminates the call and returns the result sets.

- Calling source (embedded UAP written in C)

```
#include <stdio.h>
#include <string.h>

main()
{
  EXEC SQL BEGIN DECLARE SECTION;
    struct {
      short len;
      char str[31];
    } cur1;
    int emp_id;
    char emp_name[13];
    int emp_age;
  EXEC SQL END DECLARE SECTION;
  --------(CONNECT process to HiRDB (omitted))--------

  cur1.len = sprintf(cur1.str, "cursor1"); .................... 1

  EXEC SQL CALL PROC(10); ..................................... 2

  If (SQLCODE == 120) { ........................................ 3

    EXEC SQL ALLOCATE GLOBAL :cur1
               FOR PROCEDURE PROC2; ............................ 4


    printf("*** emps_1 ***\n"); ................................ 5
    while (1) { ................................................ 5
      EXEC SQL FETCH GLOBAL :cur1 .............................. 5
                 INTO :emp_id,:emp_name,:emp_age; .............. 5
      if (SQLCODE<0 || SQLCODE==100) break; .................... 5
      printf("ID=%d  NAME=%s  AGE=%d\n", ....................... 5
            emp_id, emp_name, emp_age); ........................ 5
     } ......................................................... 5
    CLOSE GLOBAL :cur1; ........................................ 6


    if (SQLCODE==121) { ........................................ 7
      printf("*** emps_2 ***\n"); .............................. 8
      while (1) { .............................................. 8
    EXEC SQL FETCH GLOBAL :cur1 ................................ 8
                   INTO :emp_id,:emp_name,:emp_age; ............ 8
        if (SQLCODE<0 || SQLCODE==100) break; .................. 8
        printf("ID=%d  NAME=%s  AGE=%d\n", ..................... 8
              emp_id, emp_name, emp_age); ...................... 8
      } ........................................................ 8
      CLOSE GLOBAL :cur1; ...................................... 9
    }
  }
}
```

Explanation:

1. Sets the cursor name.
2. Executes the CALL statement.
3. Determines whether there is a result set to be returned.
4. Assigns a cursor (associates the first result set with the cursor).
5. Outputs information from the first result set.
6. Closes the cursor (associates the second result set with the cursor).
7. Determines whether there is another result set.
8. Outputs information from the second result set.
9. Closes the cursor.

(d) Notes about using the results-set return facility

- Defining the SQL stored procedure with `CREATE PROCEDURE`

  1. Specify `WITH RETURN` in the cursor declarations of the cursors to be returned as result sets.
  2. Of the cursors declared with the `WITH RETURN` specification, only those that are open when the procedure ends are returned as result sets.
  3. If there are two or more result sets to be returned, they are returned in the order that their corresponding cursors were opened.

- Creating the calling source

  1. When a procedure that returns a result state is executed, `SQLSTATE` is set to `0100C` and `SQLCODE` to `120`.
  2. To have an embedded UAP and an SQL stored procedure receive a result set, use the `ALLOCATE CURSOR` statement to allocate a cursor to the group of result sets and associate the cursor with the first result set. If another result set is to be returned, execute the `CLOSE` statement for the cursor that is referencing the previous result set. A cursor is then associated with that subsequent result set. When the `CLOSE` statement is executed and there is a subsequent result set, `SQLSTATE` is set to `0100D` and `SQLCODE` to `121` when that result set is associated with a cursor. If there is no subsequent result set, `SQLSTATE` is set to `02001` and `SQLCODE` to `100`.

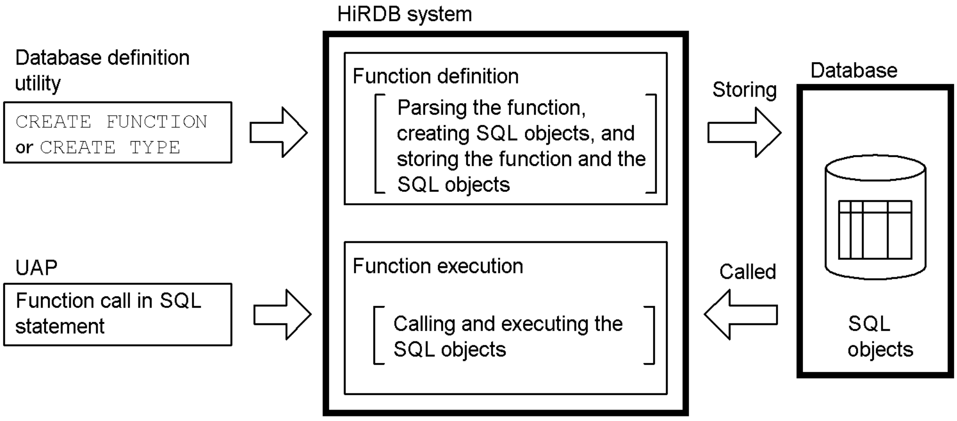
4.3.2 Defining a stored function

A *stored function* is a facility that registers a sequence of SQL-coded database operations to a database as a user-defined function.

(1) Defining and executing an SQL stored function

`CREATE FUNCTION` or `CREATE TYPE` registers a user-defined function in a database as an SQL stored function. `DROP FUNCTION` deletes an SQL stored function from the database. Once registered in a database, a user-defined function can be executed by calling it in an SQL statement. If a function has an SQL object that has been invalidated, the `ALTER ROUTINE` statement can be used to re-create that function. The following figure shows the definition and execution of an SQL stored function.

Figure 4-13: Defining and executing an SQL stored function


HiRDB system
Database definition utility
CREATE FUNCTION or CREATE TYPE
Function definition
Parsing the function, creating SQL objects, and storing the function and the SQL objects
Storing
Database
UAP
Function call in SQL statement
Function execution
Calling and executing the SQL objects
Called
SQL objects

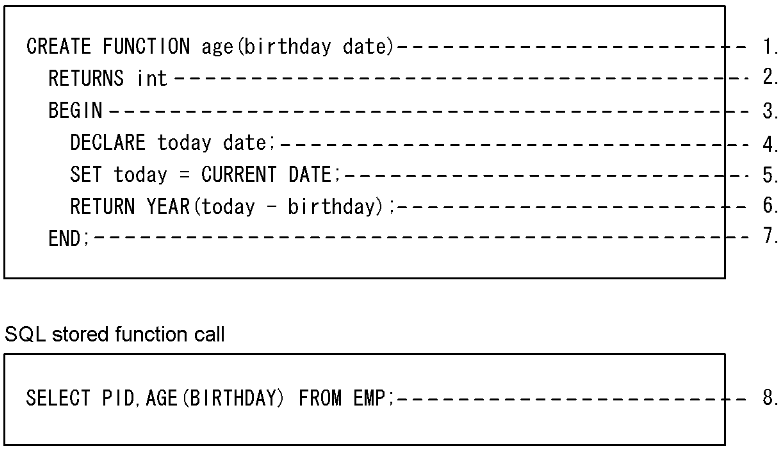
Public function

If you use a stored function defined by another user, you must specify the owner's authorization identifier and routine identifier when you call the stored function from within a UAP. However, if `CREATE PUBLIC FUNCTION` is executed to define the stored function as a public function, there is no need to specify the owner's authorization identifier when the stored function is called from a UAP (only the routine identifier must be specified).

(2) Example of an SQL stored function

The following shows an example of the definition and execution of an SQL stored function that combines multiple routine control SQL statements and defines them as a user-defined function (function).

Figure 4-14: SQL stored function example



SQL stored function definition

```
CREATE FUNCTION age(birthday date)------------------------ 1.
  RETURNS int ------------------------------------------- 2.
  BEGIN ------------------------------------------------- 3.
    DECLARE today date;----------------------------------- 4.
    SET today = CURRENT DATE;----------------------------- 5.
    RETURN YEAR(today - birthday);------------------------ 6.
  END;--------------------------------------------------- 7.
```

SQL stored function call

```
SELECT PID,AGE(BIRTHDAY) FROM EMP;------------------------ 8.
```

Explanation

1. Defines the user-defined function name and the SQL parameters.
2. Specifies the function return value.
3. Begins the compound statements.
4. Declares SQL variables.
5. Specifies value assignments.
6. Specifies return of the function return value.
7. Ends the compound statements.
8. Retrieves the SQL stored function with a function call.

Note

For details about the individual SQL statements, see the *HiRDB Version 9 SQL Reference* manual.

Defining the following functions is helpful.

- Function that calculates the last date of a month containing the specified date

```
CREATE FUNCTION LASTDAY(INDATE DATE) RETURNS DATE
  BEGIN
    DECLARE MM1 INTEGER;
    SET MM1=MONTH(INDATE)-1;
    RETURN (INDATE-MM1 MONTHS+(31-DAY(INDATE))
    DAYS+MM1 MONTHS);
  END
```

- Function that calculates the day of a specified date with an integer from 0 (Sunday) through 6 (Saturday)

```
CREATE FUNCTION DNOFWEEK(INDATE DATE) RETURNS INTEGER
  BEGIN
   RETURN MOD(DAYS(INDATE),7);
  END
```

- Function that calculates the day of a specified date in English

```
CREATE FUNCTION DAYOFWEEK(INDATE DATE) RETURNS CHAR(3)
  BEGIN
   RETURN (CASE MOD(DAYS(INDATE),7) WHEN 0 THEN 'SUN'
    WHEN 1 THEN 'MON'
    WHEN 2 THEN 'TUE'
    WHEN 3 THEN 'WED'
    WHEN 4 THEN 'THU'
    WHEN 5 THEN 'FRI'
    ELSE 'SAT'   END);
  END
```

- Function that calculates the date of the specified day that immediately follows a specified date

```
CREATE FUNCTION NEXTDAY(INDATE DATE, DAYOFWEEK CHAR(3))
  RETURNS DATE
  BEGIN
   DECLARE SDOW, TDOW INTEGER;
   SET TDOW=(CASE, DAYOFWEEK WHEN 'SUN' THEN 0
      WHEN 'MON' THEN 1
      WHEN 'TUE' THEN 2
      WHEN 'WED' THEN 3
      WHEN 'THU' THEN 4
      WHEN 'FRI' THEN 5
      ELSE 6 END);
   SET SDOW=MOD(DAYS(INDATE),7);
   RETURN  (INDATE + (CASE WHEN TDOW>SDOW THEN TDOW-SDOW
      ELSE 7+TDOW-SDOW END) DAYS);
  END
```

(When the day argument is an integer [0 to 6])

```
CREATE FUNCTION NEXTDAY(INDATE DATE, DNOFWEEK INTEGER)
  RETURNS DATE
  BEGIN
    DECLARE SDOW, TDOW INTEGER;
    SET TDOW=DNOFWEEK;
    SET SDOW=MOD(DAYS(INDATE),7);
    RETURN (INDATE + (CASE WHEN TDOW>SDOW THEN TDOW-SDOW
      ELSE 7+TDOW-SDOW END) DAYS);
  END
```

- Function that calculates the year and month (*yyyy-mm*) of a specified date when each month ends on the 20th

```
CREATE FUNCTION YYYYMM20(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    RETURN SUBSTR(CHAR(INDATE+1 MONTH -20 DAYS),1,7);
  END
```

- Function that calculates the year (*yyyy*) of the specified date when each fiscal year ends on March 20

```
CREATE FUNCTION YYYY0320(INDATE DATE) RETURNS CHAR(4)
  BEGIN
    RETURN SUBSTR(CHAR(INDATE-2 MONTHS -20 DAYS)1,4);
  END
```

- Function that calculates the year and quarter (*yyyy-nQ*) of the specified date when each fiscal year ends on March 20

```
CREATE FUNCTION YYYYNQ0320(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    DECLARE WORKDATE DATE;
    SET WORKDATE=(INDATE -2 MONTHS -20 DAYS);
    RETURN (SUBSTR(CHAR(WORKDATE),1,5)||
         SUBSTR(DIGITS((MONTH(WORKDATE)+2)/3),10,1)| |'Q');
  END
```

- Function that calculates the year and half (*yyyy-nH*) of the specified date when each fiscal year ends on March 20

```
CREATE FUNCTION YYYYNH0320(INDATE DATE) RETURNS CHAR(7)
  BEGIN
    DECLARE WORKDATE DATE;
    SET WORKDATE=(INDATE -2 MONTHS -20 DAYS);
    RETURN (SUBSTR(CHAR(WORKDATE),1,5) ||
         SUBSTR(DIGITS((MONTH(WORKDATE)+5)/6),10,1)| |'H');
  END
```

- Function that calculates the number of months between dates (*argument 1 - argument 2*)(extra days are discarded)

```
CREATE FUNCTION MONTHBETWEEN0(INDATE1 DATE, INDATE2 DATE)
  RETURNS INTEGER
  BEGIN
    DECLARE YMINTERDATE INTERVAL YEAR TO DAY;
    SET YMINTERDATE=INDATE1-INDATE2;
    RETURN (YEAR(YMINTERDATE)*12+MONTH(YMINTERDATE));
  END
```

- Function that calculates the number of months between two dates (*argument 1 - argument 2*) to several decimal places. (The number-of-months value for one day is calculated by setting the day of the earlier date as the starting point of each month and then dividing 1 by the number of days in the month with the later date.)

```
CREATE FUCNTION MONTHBETWEEN(INDATE1 DATE,INDATE2 DATE)
  RETURNS DECIMAL(29,19)
  BEGIN
    DECLARE INTERDATE INTERVAL YEAR TO DAY;
    DECLARE DMONTHS DEC(29,19);
    DECLARE YYI,MMI INTEGER;
    DECLARE WDATE DATE;
    DECLARE SIGNFLAG DEC(1);
    IF INDATE1>INDATE2 THEN
      SET INTERDATE=INDATE1-INDATE2;
      SET WDATE=INDATE2;
      SET SIGNFLAG=1;
    ELSEIF INDATE1<INDATE2 THEN
      SET INTERDATE=INDATE2-INDATE1;
      SET WDATE=INDATE1;
      SET SIGNFLAG=-1;
    ELSE RETURN 0;
    END IF;
    SET YYI=YEAR(INTERDATE);
    SET MMI=MONTH(INTERDATE);
    SET WDATE=WDATE+YYI YEARS+MMI MONTHS;
    SET DMONTHS=YYI*12+MMI
      +DEC(DAY(INTERDATE),2)/(DAYS(WDATE+1 MONTH)-
DAYS(WDATE));
    IF SIGNFLAG=1 THEN RETURN DMONTHS;
    ELSE RETURN -DMONTHS;
    END IF;
  END
```

- Function that calculates the number of years between two dates (*argument 1 - argument 2*) to several decimal places (the number-of-years value for one day is calculated by setting the month and day of the earlier date as the starting point of each year and then dividing 1 by the number of days in the year with the later date).

```
CREATE FUNCTION YEARBETWEEN(INDATE1 DATE,INDATE2 DATE)
  RETURNS DECIMAL(29,19)
  BEGIN
    DECLARE INTERDATE INTERVAL YEAR TO DAY;
    DECLARE DYEARS DEC(29,19);
    DECLARE YYI,MMI INTEGER;
    DECLARE WDATE1, WDATE2 DATE;
    DECLARE SIGNFLAG DEC(1);
    IF INDATE1>INDATE2 THEN
       SET INTERDATE=INDATE1-INDATE2;
       SET WDATE1=INDATE1;
       SET WDATE2=INDATE2;
       SET SIGNFLAG=1;
    ELSEIF INDATE1<INDATE2 THEN
       SET INTERDATE-INDATE2-INDATE1;
       SET WDATE1=INDATE2;
       SET WDATE2=INDATE1;
       SET SIGNFLAG=-1;
    ELSE RETURN 0;
    END IF;
    SET YYI=YEAR(INTERDATE);
    SET WDATE2=WDATE2+YYI YEARS;
    SET DYEARS=YYI
         +DEC(DAYS(WDATE1)-DAYS(WDATE2),3)
         /(DAYS(WDATE2+1 YEAR)-DAYS(WDATE2));
    IF SIGNFLAG=1 THEN RETURN DYEARS;
    ELSE RETURN -DYEARS;
    END IF;
  END
```

(3) Rules for determining the called function and the result data type

- A function is called if the counts for authorization identifiers, routine identifiers, and arguments all match, if the argument data types do not include abstract data types, and if the parameter data types perfectly match the argument order. In this case, the data type of the function result is the `RETURNS` clause data type of the called function.
- A function is not called if any of the counts for authorization identifiers, routine identifiers, or arguments do not match.
- If the counts for authorization identifiers, routine identifiers, and arguments all match, but the argument data types include an abstract data type or the parameter data types do not perfectly match the argument order, the called function is determined as follows:
    - If an abstract data type is not included in the arguments

      The HiRDB system checks the arguments sequentially from the leftmost argument and sets the pre-defined data types of the individual arguments as references. The system then calls the function whose parameters have pre-defined data types with priorities that are equal to those of the references. If it does not find such a function, the system looks at the functions whose parameters have pre-defined data types with priorities that are less than those of the references and calls the function with the highest data type priorities. The following table shows the priorities of pre-defined data types. If an abstract data type is not included in the arguments, the called function is uniquely determined during SQL parsing, and the `RETURNS` clause data type of the called function becomes the data type of the function result.

Table 4-3: Priorities of pre-defined data types

Argument data type	Priority
Numeric data	SMALLINT → INTEGER → DECIMAL → SMALLFLT → F LOAT
Character data	CHAR → VARCHAR
National character data	NCHAR → NVARCHAR
Mixed character string data	MCHAR → MVARCHAR

A → B: Indicates that A has a higher priority than B.

- If an abstract data type is included in the arguments

  If an abstract data type is included in the arguments, the function to be called is determined according to the sequence described as follows:

1. Determining the basic function

   The HiRDB system checks the arguments sequentially from the leftmost argument and sets the data types of the individual arguments as references. The system then selects the function whose parameters have data types with priorities that are equal to those of the references and sets that function as the basic function. If it does not find such a function, the system looks at the functions whose parameters have pre-defined data types with priorities that are less than those of the references and selects the function with the highest data type priorities. If a data type is a pre-defined data type, the priority is determined according to *Table 4-3*. If a data type is an abstract data type, the priority is determined according to the following table.

Table 4-4: Priorities of abstract data types

Argument data type	Priority
Abstract data type	Same data type → super type#

#: The super type that is specified directly by the UNDER clause in an abstract type definition has a higher priority than other super types.

A → B: Indicates that A has a higher priority than B.

1. Determining other candidate functions

   If an argument has an abstract data type, the data types of values that can actually be used as data for that argument are the same as the abstract data type in the argument definition and the subtypes of that data type. In addition to the basic function, all functions that have parameters corresponding to the same data type as the abstract data type of the argument, or to the abstract data type of a subtype, become candidates for the called function.

If the basic function is the only candidate function, it becomes the called function. The data type of the function result becomes the `RETURNS` clause data type of the called function.

2. Limiting the candidate functions based on the data type in the `RETURNS` clause

   For each candidate function other than the basic function, the HiRDB system checks whether the `RETURNS` clause data type is compatible with the `RETURNS` clause data type for the basic function. If the data type is not compatible, the function is dropped from the candidate functions. After checking this compatibility for all candidate functions, the HiRDB system determines the data type of the function result based on the `RETURNS` clause data types for the remaining candidate functions. The system performs a set operation (`UNION [ALL]` or `EXCEPT [ALL]`) on the remaining candidates. The resulting data type and data length become the data type and data length of the function result. For details, see the *HiRDB Version 9 SQL Reference* manual.

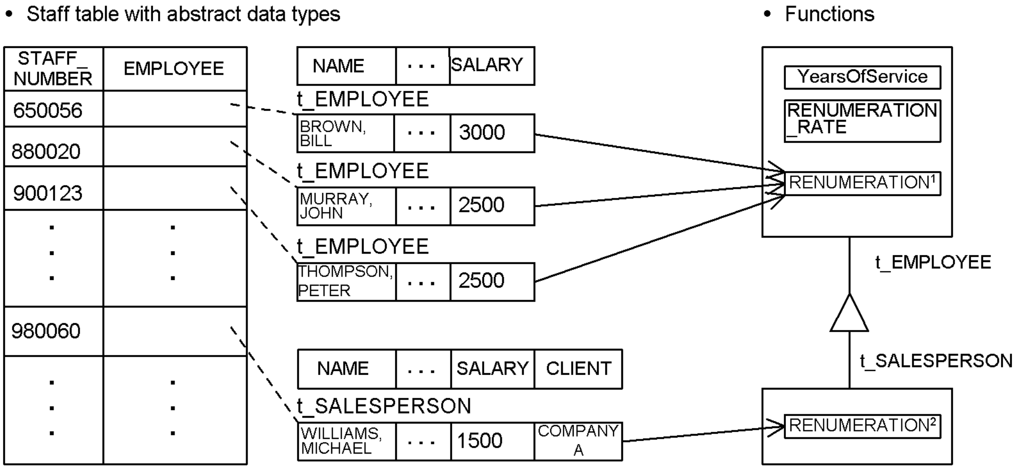
   However, if the data type of the function result is an abstract data type, the abstract data type of the `RETURNS` clause for the basic function is used.

3. Determining the called function when an SQL statement is executed

   If there are two or three functions that cannot be determined uniquely, the HiRDB system determines which one of these candidate functions to call based on the actual data type used for each abstract data type argument when the SQL statement is executed. The system checks the arguments sequentially from the leftmost argument. If the actual value of an argument is a non-null value, the data type of that value is used as a reference. If the actual value is a null value, the data type of that argument is used as reference. From the candidate functions, the HiRDB system selects the function whose parameters have data types with priorities that are equal to those of the references and sets that function as the called function. If it cannot find such a function, the system looks at the functions whose parameters have pre-defined data types with priorities that are less than those of the references and selects the function with the highest data type priorities.

Because HiRDB allows a function to be defined more than once, there may be several candidates for a called function. The called function is determined by how the function call specification and the function definition match. The following figure shows the correspondences between a table with abstract data types and the called function.

Figure 4-15: Correspondences between a table with abstract data types and the called function



Explanation

Suppose that the following SQL statement uses the abstract data type function `REMUNERATION` to retrieve data from the staff table:

```
SELECT STAFF_NUMBER FROM STAFF_TABLE WHERE
  REMUNERATION(EMPLOYEE)>=2000.00
```

In this case, the function for each data type is determined and called according to whether the data for the parameter value is `t_EMPLOYEE` or `t_SALESPERSON`.

For details about the definitions of this staff table, see the *HiRDB Version 9 Installation and Design Guide*.

#1: `REMUNERATION = SALARY` x `REMUNERATION_RATE()`

#2: `REMUNERATION = TOTAL_NUMBER_OF_CLIENTS` x 1,000 + `SALARY` x `REMUNERATION_RATE()`

Examples of determining the called function when abstract data types are included

In the examples below, A, B, and C are abstract data types, C is the super type of B, and B is the super type of A (priority of abstract data type: A → B → C).

Example 1

Prerequisite conditions

Table definition

```
CREATE TABLE T1(C1 C)
```

Function definitions

```
f(A), f(B), f(C)
```

SQL statement

```
SELECT f(C1) FROM T1
```

Results

Basic function

```
f(C)
```

Candidate functions when function call is `f(C1)`

```
f(A), f(B), f(C)
```

Called function

The following table shows which function is called when the SQL statement is executed.

Actual value of T1.C1	Called function
Type A	`f(A)`
Type B	`f(B)`
Type C	`f(C)`
Null value	`f(C)`

Example 2

Prerequisite conditions

Table definition

```
CREATE TABLE T1(C1 C,C2 B)
```

Function definitions

```
f(A,A), f(A,B), f(A,C), f(B,A), f(B,C), f(C,A),
f(C,B), f(C,C)
```

SQL statement

```
SELECT f(C1,C2) from T1
```

Results

Basic function

```
f(C,B)
```

Candidate functions when function call is `f(C1,C2)`

`f(A,A), f(A,B), f(A,C), f(B,A), f(B,C), f(C,A), f(C,B)`

Called function

The `following` table shows which function is called when the SQL statement is executed.

Actual value of T1.C1	Actual value of T1.C2	Called function
Type A	Type A	`f(A,A)`
	Type B	`f(A,B)`
	Null value	`f(A,B)`
Type B	Type A	`f(B,A)`
	Type B	`f(B,C)`
	Null value	`f(B,C)`
Type C	Type A	`f(C,A)`
	Type B	`f(C,B)`
	Null value	`f(C,B)`
Null value	Type A	`f(C,A)`
	Type B	`f(C,B)`
	Null value	`f(C,B)`

4.3.3 Defining and deleting stored functions

This section describes how to define and delete stored functions.

(1) Defining stored functions

- **When a stored function is created, an existing stored function may become invalid**

An existing stored function becomes invalid under the following condition:

- A UAP has called a stored function that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be created.

In this case, use `ALTER ROUTINE` to re-create the stored function that was invalidated.

- **When a stored function is created, an existing stored procedure may become**

invalid

When a stored function is created, an existing stored procedure may become invalid. An existing stored procedure becomes invalid under the following condition:

- The stored procedure calls a stored procedure that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be created.

In this case, use `ALTER PROCEDURE` or `ALTER ROUTINE` to re-create the stored procedure that was invalidated.

- **When a stored function is created, an existing trigger may become invalid**

When a stored function is created, an existing trigger may become invalid. An existing trigger becomes invalid under the following condition:

- The trigger calls a stored function that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be created.

In this case, use `ALTER TRIGGER` or `ALTER ROUTINE` to re-create the trigger that was invalidated.

- **A created stored function becomes invalid**

A stored function may become invalid if it is created under the following circumstances:

1. A plug-in is installed.
2. A stored function that calls a function provided by the plug-in in step 1 is created.
3. A plug-in that is different from the one that was installed in step 1 is installed.

If the plug-ins installed in steps 1 and 3 provide functions that have the same name and same number of parameters, the stored function that was created in step 2 becomes in valid when step 3 is executed.

In this case, use `ALTER ROUTINE` to re-create the stored function that was invalidated.

(2) Deleting stored functions

- **When a stored function is deleted, another stored function may become invalid**

An existing stored function becomes invalid under the following condition:

- A UAP has called a stored function that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be deleted.

In this case, use `ALTER ROUTINE` to re-create the stored function that was invalidated.

- **When a stored function is invalid, a stored procedure that has the same name may become invalid**

  An existing stored procedure becomes invalid under the following condition:

  - A UAP has called a stored procedure that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be created.

  In this case, use `ALTER PROCEDURE` or `ALTER ROUTINE` to re-create the stored procedure that was invalidated.

4.4 Triggers

By defining a trigger, you can execute an SQL statement automatically when an operation (update, insertion, or deletion) is performed on a certain table. To define a trigger, specify information such as the table that defines the trigger, the SQL statement that specifies the trigger operation timing (*trigger-activating SQL statement*), the SQL statement to be executed automatically (*trigger SQL statement*), and the conditions under which that operation is executed (*trigger operation search conditions*). When an SQL statement that satisfies the trigger operation search conditions is executed for the table that defines the trigger, the trigger SQL statement is executed automatically. The following figure provides an overview of triggers.

Figure 4-16: Trigger overview

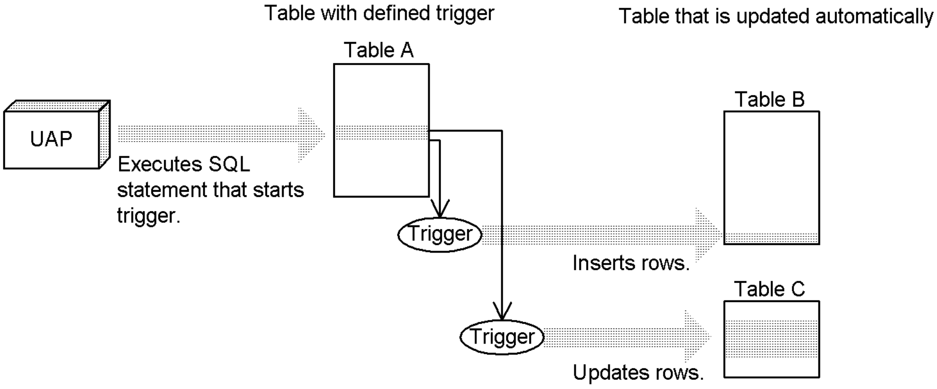


Table with defined trigger
Table that is updated automatically
Table A
Table B
UAP
Executes SQL statement that starts trigger.
Trigger
Inserts rows.
Table C
Trigger
Updates rows.

Explanation:

When the UAP executes an SQL that activates the trigger, table A, which defines the trigger, calls the trigger. If the search conditions for trigger operation are satisfied, the trigger SQL statement (in this case, row insertion for table B and row update for table C) is automatically executed.

If you use a trigger, you do not need to describe the following types of operations in the UAP:

- When a certain table is updated, always update another table.
- When a certain table is updated, always update a certain column in the updated row. (Associate a column with another column.)

For example, suppose that when prices in a product management table are changed, the changes are accumulated in a product management history table. If a trigger is not used, the UAP that updates the product management table must also always update the product management history. If a trigger is used, the UAP that updates the product management table need not be concerned about updating the product management history table because the latter table can be manipulated automatically. By using triggers appropriately as in this example, you can reduce the work load involved in creating a UAP.

When a trigger is defined, the functions, procedures, and trigger SQL objects that use that table become invalid and must be re-created. When a resource (such as a table or index) being used by a trigger is defined, redefined, or deleted, the SQL objects of the trigger become invalid and must be re-created.

For details about triggers, see the *HiRDB Version 9 Installation and Design Guide*.

4.5 SQL optimization

HiRDB features an optimization facility that improves the retrieval efficiency of SQL statements.

Optimization processing includes SQL optimizing modes that use different methods. HiRDB determines the SQL optimizing mode for each SQL based on the specified value of the SQL extension optimizing option and the SQL syntax.

The SQL optimizing mode types are as follows:

- Optimizing mode 1 based on cost (optimization processing method used in HiRDB versions before Version 06-00)
- Optimizing mode 2 based on cost (optimization processing method used in HiRDB versions starting from Version 06-00)

You can also consider the status of the database and specify an optimization method to determine the most efficient access path. There are three types of optimization methods:

- SQL optimization specifications
- SQL optimization options
- SQL extension optimizing options

SQL optimization specifications

SQL optimization specifications can be specified in SQL statements. These optimization methods are applied to the SQL statements that specify the methods.

For details about SQL optimization specifications, see the manual *HiRDB Version 9 SQL Reference*.

SQL optimization options and SQL extension optimizing options

The SQL optimization options and the SQL extension optimizing options are assigned multiple functions from which you can select those that are necessary. The functions specified by using the SQL optimization options are effective with both optimizing mode 1 based on cost and optimizing mode 2 based on cost. The functions specified with the SQL extension optimizing options are effective with only optimizing mode 2 based on cost.

For details about the SQL optimization options and SQL extension optimizing options, see *6.6.4 Environment definition information*.

Notes

Indicators for selecting the SQL optimizing mode are described as follows:

When installing HiRDB for the first time with Version 06-00 or a later version

- Hitachi recommends that you use optimizing mode 2 based on cost.
- If you use optimizing mode 2 based on cost, execute the optimizing information collection facility as necessary to further improve the optimizing precision. For details about the necessity of executing the optimizing information collection utility, see the manual *HiRDB Version 9 Command Reference*.
- The SQL optimization option and the SQL extension optimizing option have recommended values that should be specified. Make sure that these recommended values are specified, and also examine whether other functions can be used.

When upgrading a HiRDB version earlier than Version 06-00

Hitachi recommends that you use optimizing mode 1 based on cost so that you can use the HiRDB system under the same conditions as before the version upgrade. However, because some SQL statements always use optimizing mode 2 based on cost, study the specification values of the SQL extension optimizing option when you start a new operation in the environment that is already constructed. Also, do not change the specification values of the SQL optimization option.

4.5.1 SQL optimizing modes

(1) Features of the SQL optimizing modes

The following table describes the features of the SQL optimizing modes.

Table 4-5: Features of the SQL optimizing modes

SQL optimizing mode	Explanation	Advantages	Disadvantages	Selection method
Optimizing mode 1 based on cost	This is the optimization processing method based on cost for HiRDB versions before Version 06-00. This mode can also be used in HiRDB Version 06-00 and later versions.	Even if HiRDB is upgraded from a version earlier than Version 06-00, searches can be performed with the same access paths used in the earlier version. Access paths are sometimes changed for high-speed retrieval.	The optimal access path cannot always be selected because there are only a few access path candidates. (Access paths are not selected by setting facilities such as hash join as candidates.)	Specify `NONE` or `0` in the SQL extension optimizing option. Some SQL statements always use optimizing mode 2 based on cost. For details, see (2) as follows.

SQL optimizing mode	Explanation	Advantages	Disadvantages	Selection method
Optimizing mode 2 based on cost	This is the optimization processing method based on cost that is used in HiRDB Version 06-00 and later. This mode is designed for fast retrieval.	High-speed retrieval is possible because this mode selects access paths from candidates that combine hashing to join search and subquery processing.	Optimization processing takes time because this mode performs complex optimization processing.	Specify the use of optimizing mode 2 based on cost in the SQL extension optimizing option, or omit the SQL extension optimizing option.

(2) SQL statements that forcibly apply optimizing mode 2 based on cost

Even if optimizing mode 1 based on cost is being used, optimizing mode 2 based on cost is sometimes forcibly applied. The SQL statements that forcibly apply optimizing mode 2 based on cost are as follows:

- Subquery in the SET clause of the UPDATE statement
- Outer join + (inner) join
- COUNT(*) in a set operation result
- Value expression of the DISTINCT set function
- Specification of the query name of a viewed table or WITH clause to an outer join
- Partial updating and retrieval of BLOB and BINARY data
- SQL optimization specification
- Sorting with a value expression with a defined length exceeding 255 bytes
- Retrieve first *n* records
- Retrieval using the BINARY type
- Retrieval of a viewed table or WITH clause containing an internally derived table that becomes a nesting structure with at least two levels
- Matrix partitioning
- Subquery for a joined table
- Application of the MIN or MAX set function to a repetition column
- Row value constructor
- Subquery in the CASE expression
- POSITION function in which value equation 2 is the BLOB type

- Referential constraint
- Check constraint
- Limit release to allow data with a defined length of 256 bytes or more
- Specification of a table targeted for data update, deletion, or addition in a subquery
- Unnesting facility for repetition column in the `FROM` clause
- `LIMIT` clause
- Search in which an internally derived table has two or more nesting layers
- Expansion of the specification location in the query expression body
- Window functions
- `SIMILAR` predicate
- Retrieval using the `XML` type
- Character set
- Retrieval, updating, or deletion with an RDAREA name specified

The application condition and an example of each SQL sentence are shown as follows.

(a) Subquery in the SET clause of the UPDATE statement

- When a scalar or line subquery is specified in the `SET` clause of the `UPDATE` statement

Example

```
UPDATE T1 SET(C1,C2)=(SELECT MAX(C1),MAX(C2) FROM T2) WHERE
C3=1
```

Note

The underlined section is the applicable location.

(b) Outer join + (inner) join

- When an (inner) join is specified in the `FROM` clause

Example

```
SELECT T1.C1,T2.C2 FROM T1 INNER JOIN T2 ON T1.C1=T2.C1
```

Note

The underlined section is the applicable location.

- When a table reference that includes `LEFT [OUTER] JOIN` and any other table reference are delimited with a comma (`,`) and specified in the `FROM` clause

Example

```
SELECT T1.C1,T2.C2 FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1,
T3 WHERE T1.C1=T3.C1
```

Note

The underlined section is the applicable location.

- When *table-reference1* `LEFT [OUTER] JOIN` *table-reference2* is specified in the `FROM` clause, and `LEFT [OUTER] JOIN` is nested and specified in *table-reference2*

Example

```
SELECT T1.C1,T2.C2,T3.C2 FROM T1 LEFT OUTER JOIN
  (T2 LEFT OUTER JOIN T3 ON T2.C1=T3.C1)
  ON T1.C1=T3.C1
```

Note

The underlined section is the applicable location.

(c) COUNT(*) in a set operation result

- When the query expression body specified in the `FROM` clause includes a set operation

Example

```
SELECT COUNT(*) FROM (SELECT C1 FROM T1 UNION SELECT C1 FROM
T2)
```

Note

The underlined section is the applicable location.

(d) Value expression of the DISTINCT set expression

- When a value expression other than a column specification is specified as an argument of the `DISTINCT` set function (`COUNT`, `SUM`, or `AVG`)

Example

```
SELECT AVG(DISTINCT C1+C2) FROM T1
```

Note

The underlined section is the applicable location.

(e) Specification of the query name of a viewed table or WITH clause to an outer join

- When `LEFT [OUTER] JOIN` for the query name of a viewed table or `WITH` clause is specified in the `FROM` clause, and an internally derived table is created from the query name of that viewed table or `WITH` clause

Example

```
WITH W1(C1,C2) AS (SELECT C1,COUNT(*) FROM T1 GROUP BY C1)
  SELECT W1.C1,W1.C2,T2.C2 FROM W1 LEFT JOIN T2 ON
W1.C1=T2.C1
```

Note

The underlined section is the applicable location.

(f) Partial updating and retrieval of BLOB and BINARY data

- When `BLOB`-type data is specified in value expression 1 of the `SUBSTR` scalar function

Example

```
SELECT SUBSTR(C1,1,500) FROM T1
```

Note

The underlined section is the applicable location. `C1` is a `BLOB`-type column.

- When the update target of an `UPDATE` statement is a `BLOB`-type column, and a concatenation operation is specified in the update value

Example

```
UPDATE T1 SET C1=C1||?
```

Note

The underlined section is the applicable location. `C1` is a `BLOB`-type column.

- When the update target of an `UPDATE` statement is a `BLOB`-type column or has the `BLOB` attribute, and a column or component specification is specified in the update value

Example

```
UPDATE T1 SET C1=C2
```

Note

The underlined section is the applicable location. `C1` and `C2` are `BLOB`-type columns.

(g) SQL optimization specification

- When an SQL optimization specification for a used index is specified

Example

```
SELECT T1.C1 FROM T1 WITH INDEX(idx1) WHERE T1.C2<=500
```

Note

The underlined section is the applicable location.

- When an SQL optimization specification for a join method is specified

  Example

  ```
  SELECT T1.C1,T2.C2 FROM T1 INNER JOIN BY NEST T2 ON
  T1.C1=T2.C1
  ```

  Note

  The underlined section is the applicable location.

- When an SQL optimization specification for a subquery execution method is specified

  Example

  ```
  SELECT T1.C1 FROM T1 WHERE T1.C1=ANY
    (HASH SELECT T2.C1 FROM T2 WHERE T2.C2='302S')
  ```

  Note

  The underlined section is the applicable location.

(h) Sorting with a value expression with a defined length exceeding 255 bytes

- When a `CHAR`, `VARCHAR`, `MCHAR`, or `MVARCHAR` expression with a minimum defined length of 256 bytes, or an `NCHAR` or `NVARCHAR` expression with a minimum defined length of 128 characters is specified as the sort key item in an `ORDER BY` clause

  Example 1

  ```
  SELECT C1,C2 FROM T1 ORDER BY C2
  ```

  Note

  The underlined section is the applicable location. `C2` is a `VARCHAR(300)` column.

  Example 2

  ```
  SELECT C1,C3||C4 FROM T1 ORDER BY 2
  ```

  Note

  The underlined section is the applicable location. `C3||C4` is an `NCHAR(150)` value expression.

(i) Retrieve first n records

- When a `LIMIT` clause is specified directly after an `ORDER BY` clause

  Example

  ```
  SELECT PCODE,SQUANTITY FROM STOCK WHERE SQUANTITY>20 ORDER
  BY 2,1 LIMIT 10
  ```

Note

The underlined section is the applicable location.

(j) Retrieval using the BINARY type

- When a `BINARY`-type column is retrieved

Example

```
SELECT C1 FROM T1
```

Note

The underlined section is the applicable location. `C1` is a `BINARY`-type column.

(k) Retrieval of a viewed table or WITH clause containing an internally derived table that becomes a nesting structure with at least two levels

- When a `FROM` clause contains a query specification that specifies the query name of a viewed table or `WITH` clause, and that viewed table or `WITH` clause contains a `FROM` clause for a derived query expression that specifies the viewed table or `WITH` clause that becomes the internally derived table

Example

```
WITH Q1(QC1,QC2) AS (SELECT C1,C2 FROM V1 GROUP BY C1,C2)

SELECT AVG(QC1),QC2 FROM Q1 GROUP BY QC2
```

Note

The underlined section is the applicable location. `V1` is the viewed table that becomes the internally derived table.

(l) Matrix partitioning

- When a retrieval, update, deletion, or list operation is performed on a matrix-partitioned table

Example

```
SELECT * FROM T1
```

Note

The underlined section is the applicable location. `T1` is a matrix-partitioned table.

(m) Subquery for a joined table

- When a query specification containing a joined table is specified and a subquery is specified in the `ON` search condition of the `FROM` clause, in the `WHERE` clause, or in the `HAVING` clause

Example

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.C1=T2.C1
  WHERE T1.C1=ANY(SELECT C1 FROM T3)
```

Note

The underlined section is the applicable location.

(n) Application of the MIN or MAX set function to a repetition column

- When a repetition column in the `FLAT` specification is specified in the `MIN` or `MAX` set function

Example

```
SELECT MIN(FLAT(C1)) FROM T1
```

Note

The underlined section is the applicable location. `C1` is a repetition column.

(o) Row value constructor

- When a row value constructor is specified

Example

```
SELECT * FROM T1 WHERE (C1,C2,C3)>(1,2,3)
```

Note

The underlined section is the applicable location.

(p) Subquery in the CASE expression

- When a subquery is specified in the `CASE` expression

Example

```
SELECT CASE(SELECT C1 FROM T1) WHEN 1 THEN C2 ELSE C1 END
FROM T1
```

Note

The underlined section is the applicable location.

(q) POSITION scalar function in which value expression 2 is the BLOB type

- When the `BLOB` type is specified in value expression 2 of the `POSITION` scalar function

Example

```
SELECT POSITION(? AS BLOB(1K) IN C1) FROM T1
```

Note

The underlined section is the applicable location. `C1` is a `BLOB`-type column.

(r) Referential constraint

- When insertion, update, or deletion is executed for a referenced table or a referencing table

  Example

  ```
  UPDATE T1 SET C1=?
  ```

  Note

  The underlined section is the applicable location. `T1` is a referenced table or a referencing table.

(s) Check constraint

- When insertion or update is executed for a column in which a check constraint is defined

  Example

  ```
  INSERT INTO T1(C1,C2) VALUES(?,?)
  ```

  Note

  The underlined section is the applicable location. `C1` is the column in which a check constraint is defined.

(t) Limit release to allow data with a defined length of 256 bytes or more

- When one of the following expressions is defined in the `GROUP BY` clause
  - `CHAR`, `VARCHAR`, `MCHAR`, or `MVARCHAR` type with a defined length of 256 bytes or more
  - `NCHAR` or `NVARCHAR` type of 128 bytes or more
  - `BINARY` type of 256 bytes or more

  Example

  ```
  SELECT C1,COUNT(*) FROM T1 GROUP BY C1
  ```

  Note

  The underlined section is the applicable location. `T1.C1` is a character string of 256 bytes or more.

- When one of the following value expressions is specified for an argument of a set function
  - `CHAR`, `VARCHAR`, `MCHAR`, or `MVARCHAR` type with a defined length of 256 bytes or more
  - `NCHAR` or `NVARCHAR` type of 128 bytes or more

  - BINARY type of 256 bytes or more

Example

```
SELECT MIN(C1) FROM T1
```

Note

The underlined section is the applicable location. T1.C1 is a character string of 256 bytes or more.

- When a query expression body is specified in a viewed table, a WITH clause, or a FROM clause, an internally defined table is created, and one of the following value expressions is specified in the selection expressions of the internally derived table
  - CHAR, VARCHAR, MCHAR, or MVARCHAR type of 256 bytes or more
  - NCHAR or NVARCHAR type of 128 bytes or more
  - BINARY type of 256 bytes or more

Example

```
WITH W1(C1,C2) AS (SELECT DISTINCT C1,C2 FROM T1)
    SELECT C2,COUNT(*) FROM W1 GROUP BY C2
```

Note

The underlined section is the applicable location. T1.C1 is a character string of 256 bytes or more.

(u) Specification of a table targeted for data update, deletion, or addition in a subquery

- When a table targeted for data update, deletion, or addition is specified in a subquery

Example 1

```
UPDATE T1 SET C1=NULL WHERE C1=(SELECT MIN(C1) FROM T1)
```

Example 2

```
DELETE FROM T1 WHERE C1=(SELECT MIN(C1) FROM T1)
```

Example 3

```
INSERT INTO T1(C1,C2) VALUES((SELECT MIN(C1) FROM T1),NULL)
```

Note

The underlined section is the applicable location.

- When a table to which data is to be added is specified in the query expression body of the INSERT statement

Example

```
INSERT INTO T1(C1,C2) SELECT C1,C2+1 FROM T1
```

Note

The underlined section is the applicable location.

(v) Unnesting facility for repetition column in the FROM clause

- When `FLAT` is specified in the `FROM` clause

Example

```
SELECT C1,C2 FROM T1(FLAT(C1,C2)) WHERE C1<10 AND C2 >20
```

Note

The underlined section is the applicable location. `C1` and `C2` are repetition columns.

(w) LIMIT clause

- When the `LIMIT` clause is specified

Example

```
SELECT PCODE, SQUANTITY FROM STOCK WHERE SQUANTITY > 20 ORDER
BY 2, 1 LIMIT 20, 10
```

Note

The underlined section is the applicable location.

(x) Search in which an internally derived table has two or more nesting layers

- When the `FROM` clause of a query specification that creates an internally derived table also specifies a query specification that becomes an internally derived table

Example

```
SELECT AVG(QC1),QC2 FROM(SELECT C1,C2 FROM V1 GROUP BY C1,C2)
AS Q1(QC1,QC2)
```

Note

The underlined section is the applicable location. `V1` is a view table that becomes an internally derived table.

(y) Expansion of the specification location in the query expression body

- When a set operation is specified in a viewed table, the `WITH` clause, or the `FROM` clause and this query creates an internally derived table

Example

```
WITH V1(C1,C2) AS (SELECT C1,C2 FROM T1 UNION SELECT C1,C2
FROM T2)
    SELECT C1 FROM V1 WHERE C2>0
```

Note

The underlined section is the applicable location.

- When a set operation is specified in the `INSERT` statement

Example

```
INSERT INTO T3 (C1,C2)
    SELECT C1,C2 FROM T1 UNION ALL SELECT C1,C2 FROM T2
```

Note

The underlined section is the applicable location.

- When a set operation is specified in a subquery

Example

```
SELECT C1, C2 FROM T3
    WHERE EXISTS(SELECT C1 FROM T1 EXCEPT SELECT C1 FROM T2)
```

Note

The underlined section is the applicable location.

(z) Window functions

- When a selection expression contains a window function

Example

```
SELECT C1,C2,COUNT(*) OVER() FROM T1
```

Note

The underlined section is the applicable location.

(aa) SIMILAR predicate

- When the `SIMILAR` predicate is specified

Example

```
SELECT C1 FROM T1 WHERE C2 SIMILAR TO '%(b|g)%'
```

Note

The underlined section is the applicable location.

(ab) Retrieval using the XML type

- When a retrieval using the `XML` type is performed

  Example

  ```
  SELECT C1 FROM T1
      WHERE XMLEXISTS('/BOOK INFORMATION[PRICE=1000]'
                 PASSING BY VALUE C2)
  ```

  Note

  The underlined section is the applicable location. `T1.C2` is an `XML`-type column.

(ac) Character set

- When the SQL statement contains a column for which a character set is specified

  Example

  ```
  SELECT C1, C2 FROM T1 WHERE C1='HiRDB'
  ```

  Note

  The underlined section is the applicable location. `T1.C1` is the column for which a character set is specified.

(ad) Retrieval, updating, or deletion with an RDAREA name specified

- When the SQL statement contains the names of RDAREAs at the destination

  Example

  ```
  SELECT C1 FROM T1 IN ('RU01,RU02') WHERE C1='HiRDB'
  ```

  Note

  The underlined section is the applicable location. `RU01` and `RU02` are the RDAREAs at the destination.

(3) Valid scope of the SQL optimization option and SQL extension optimizing option

The following table shows the SQL optimizing modes in which the SQL optimization option and SQL extension optimizing option are valid.

Table 4-6: SQL optimizing modes in which the SQL optimization option and SQL extension optimizing option are valid

SQL optimizing mode	SQL optimization option	SQL extension optimizing option
Optimizing mode 1 based on cost	V	--
Optimizing mode 2 based on cost	V	V

V: The option is valid in this mode.

--: The option is invalid in this mode.

(4) Checking the SQL optimizing mode selected by the optimization process

To check the SQL optimizing mode that was selected by the optimization process for each SQL statement, use the access path display utility. For details about the access path display utility, see the *HiRDB Version 9 Command Reference* manual.

(5) Notes

1. When the SQL optimizing mode is changed, the search performance of an SQL statement may drop because the access path is changed. If the environment being used for actual operation does not allow adequate evaluation of performance, Hitachi recommends that you do not change the SQL optimizing mode.
2. If you are installing HiRDB for the first time, Hitachi recommends that you use optimizing mode 2 based on cost. If you are using another SQL extension optimizing option, use it by adding it to optimizing mode 2 based on cost. By using optimizing mode 2 based on cost, you can select access paths capable of retrieving data faster because the optimization process can select many types of access paths.

   Normally, optimizing mode 2 based on cost is applied because it is the default value for the `pd_additional_optimize_level` operand in the HiRDB system definition. Optimizing mode 2 based on cost is also applied when you use the simple setup tool, `SPSetup.bat`, or an environment setup support tool (such as HiRDEF) to set up your HiRDB environment.
3. If you upgrade HiRDB from a version earlier than 06-00, Hitachi recommends that you continue to use optimizing mode 1 based on cost because you are using HiRDB in the same conditions as before the version upgrade. However, some SQL statements may always use optimizing mode 2 based on cost.
4. Normally, the narrowing condition is considered in the optimization process. However, if a hash join, subquery hash execution is applied to the SQL extension optimizing option and there is no narrowing condition, or if the narrowing condition does not produce much narrowing of the number of rows, a hash join that sets a table with more rows as an inner table may be applied, or a table with more rows may be transferred. In such cases, execute the optimizing information collection utility by using one of the following methods, as necessary. For details about the necessity of executing the optimizing information collection utility, see the manual *HiRDB Version 9 Command Reference* and verify the performance.
   - With data stored in the table, set the optimizing information collection level to `lvl1` (specify `lvl1` in the `-c` option) and execute the optimizing information collection utility. When `lvl1` is specified, the optimizing information collection utility can be executed in a relatively short time

because the utility fetches only information on the number of rows in the table. To fetch the number of rows for all tables in the schema, specify `ALL` in the `-t` option.

- If data cannot be stored in the table or if a test environment is being used, specify the number of rows (`NROWS`) found in the table used in the actual environment, specify the `-s` option for each table, and then execute optimization. The following is an example of the specification in the optimization parameter file when the number of rows in the table is set to 1,000:

  # Table optimization information

  ```
  NROWS     1000        # Total number of rows in table
  ```

5. If you are using optimizing mode 1 based on cost, normally you do not need to execute the optimizing information collection utility. But if you do execute the utility, set the optimizing information collection level to `lvl1`.

4.5.2 Optimization method types

This subsection describes the optimization method types for SQL optimization specifications, SQL optimization options, and SQL extension optimizing options.

(1) SQL optimization specifications

The SQL optimization specifications consist of the following optimization methods:

- SQL optimization specification for a used index
- SQL optimization specification for a join method
- SQL optimization specification for a subquery execution method

(2) SQL optimization options

The SQL optimization options consist of the following optimization methods:

1. Forced nest-loop-join
2. Making multiple SQL objects
3. Increasing the target floatable servers (back-end servers for fetching data)
4. Prioritized nest-loop-join
5. Increasing the number of floatable server candidates
6. Priority of `OR` multiple index use
7. Group processing, `ORDER BY` processing, and `DISTINCT` set function processing at the local back-end server
8. Suppressing use of `AND` multiple indexes

9. Rapid grouping processing
10. Limiting the floatable target servers (back-end servers for fetching data)
11. Separating data collecting servers
12. Suppressing index use
13. Forcing use of multiple indexes
14. Suppressing creation of update-SQL work tables
15. Deriving high-speed search conditions
16. Applying a key condition that includes a scalar operation
17. Facility for batch acquisition from functions provided by plug-ins
18. Facility for moving search conditions into derived table

(3) SQL extension optimizing options

The SQL extension optimizing options consist of the following optimization methods:

1. Application of optimizing mode 2 based on cost
2. Hash join, subquery hash execution
3. Facility for applying join conditions including value expression

4.5.3 Specifying SQL optimization

(1) Locations where SQL optimization can be specified

(a) SQL optimization specifications

SQL optimization specifications can be specified in the following SQL statements:

- Subqueries
- Table expressions
- `DELETE` statement
- `UPDATE` statement

(b) SQL optimization options and SQL extension optimizing options

SQL optimization options and SQL extension optimizing options can be specified at the following locations. Normally, you would specify this options in the system common definitions, so that the options will be valid for all SQL statements.

- `pd_optimize_level`, `pd_additional_optimize_level` operand of the system common definitions
- `pd_optimize_level`, `pd_additional_optimize_level` operand of the front-end server definitions

- PDSQLOPTLVL, PDADDITIONALOPTLVL of the client environment definitions
- SQL compile option (procedure body of ALTER PROCEDURE, ALTER ROUTINE, ALTER TRIGGER, CREATE PROCEDURE, CREATE TRIGGER and CREATE TYPE)

(2) Priority

The priority when SQL optimization options and SQL extension optimizing options are specified in several locations is as follows. If SQL optimization specifications are specified in SQL statements, they have priority over SQL optimization options and SQL extension optimizing options.

(a) Data manipulation SQL statements in locations other than stored routines and triggers

The priority is as follows:

1. PDSQLOPTLVL and PDADDITIONALOPTLVL of the client environment definitions
2. pd_optimize_level and pd_additional_optimize_level operands of the front-end server definitions
3. pd_optimize_level and pd_additional_optimize_level operands of the system common definitions

(b) Data manipulation SQL statements in stored routines and in triggers

The priority is as follows:

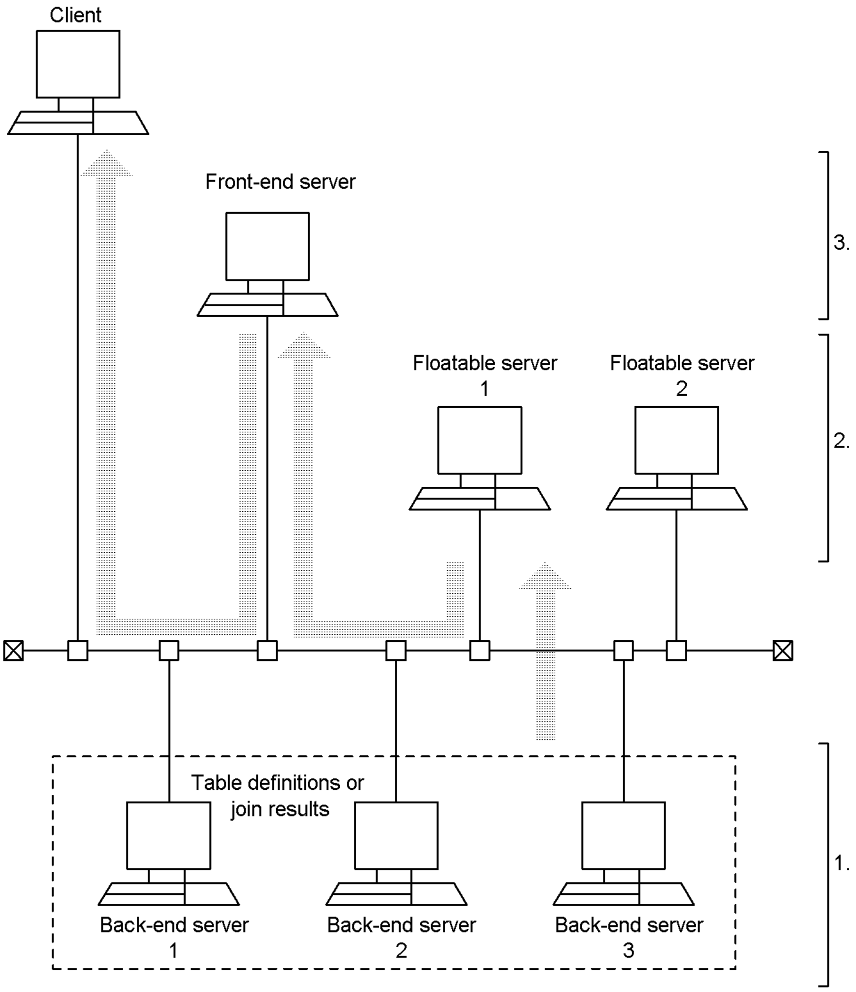
1. SQL command options (procedure body of ALTER PROCEDURE, ALTER ROUTINE, ALTER TRIGGER, CREATE PROCEDURE, CREATE TRIGGER and CREATE TYPE)
2. pd_optimize_level and pd_additional_optimize_level operands of the front-end server definitions
3. pd_optimize_level and pd_additional_optimize_level operands of the system common definitions

4.5.4 Allocating floatable servers (HiRDB/Parallel Server only)

(1) Query processing method in HiRDB

The HiRDB/Parallel Server divides query processing of SQL statements into three main steps and executes the statements. The following figure shows the query processing method of SQL statements in a HiRDB/Parallel Server.

Figure 4-17: SQL statement query processing in a HiRDB/Parallel Server



Explanation

1. The back-end servers fetch data. If the query involves two or more tables, data communication is executed between back-end servers using a join method, and step 1 may be broken down into several levels.
2. The floatable servers perform grouping, sorting, duplicate elimination, and set operation processing. Depending on the processing method, there are times when floatable servers are not used and when data communication is executed between floatable servers, and Step 2 is broken down in several levels.

3. The front-end server collects the query results and transfers the results to the client.

In the HiRDB system, the floatable servers that are used in step 2 automatically allocate the back-end servers that are not accessed with SQL statements to those individual SQL statements. However, if the SQL optimization option is specified, the allocation method for floatable servers can be changed.

The optimization methods related to floatable server allocation are shown as follows. For details about these optimization methods, see (2) as follows:

- Increasing the target floatable servers (back-end servers for fetching data)
- Limiting the target floatable servers (back-end servers for fetching data)
- Separating data collecting servers

When the next optimization method is applied, the number of allocated floatable servers can be increased to the maximum value. The features of this optimization method are described in (3).

- Increasing the number of floatable server candidates

(2) Optimization features related to floatable server allocation

The following table shows the optimization features related to floatable server allocation.

Table 4-7: Optimization features related to floatable server allocation

Optimization method	Advantages	Disadvantages
Omitted	When an SQL statement that fetches data is executed concurrently from the same back-end server, search processing can be executed rapidly because load-imposing processes such as sorting are not allocated to back-end servers for fetching data.	The communication load increases because back-end servers that do not fetch data are allocated as floatable servers.
Increasing the target floatable servers (back-end servers for fetching data)	When this method is combined with "increasing the number of floatable server candidates," the effectiveness of parallel processes such as sorting increases in the floatable servers because all back-end servers are allocated as floatable servers.	When multiple SQL statements are executed concurrently, the concurrent execution performance drops because multiple processes are allocated to the same floatable server. The communication load also increases.

Optimization method	Advantages	Disadvantages
Limiting the target floatable servers (back-end servers for fetching data)	Work division of the back-end servers can be implemented based on table definitions because only those back-end servers in which tables to be used for searching are defined are allocated as floatable servers.	If a large volume of data is stored in a table that has few partitions, all back-end servers cannot be used effectively because the number of floatable servers that can be used decreases.
Separating data collection servers	When data is sent to a data collection server from both that server and a separate server at the same time, the transfer from the same server has priority. Therefore the processing for the separate server is performed later. When separating data collection servers is applied, data is accepted equally for all servers because all servers can be treated as individual servers.	When a single SQL statement contains multiple queries, such as set operations and searches involving subqueries, the concurrent execution performance drops because the same floatable server is used for all of those queries.

(3) Optimization features related to the number of floatable server allocation candidates

The following table shows the optimization features related to the number of floatable server allocation candidates.

Table 4-8: Optimization features related to number of floatable server allocation candidates

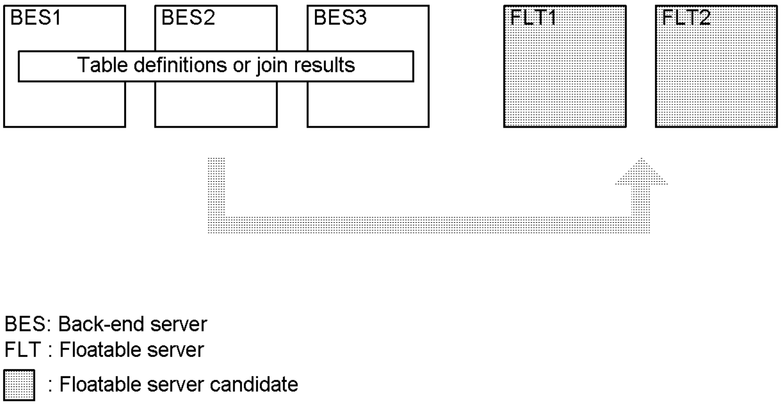
Optimization method	Advantages	Disadvantages
Omitted	In searches involving many data items, more servers are allocated as floatable servers. In searches involving few data items, fewer servers are allocated as floatable servers.	If a narrowing predicate such as `=` or `BETWEEN` is specified in the search conditions, the HiRDB system judges that the number of data items is low and automatically reduces the number of allocated floatable servers. If the `=` or `BETWEEN` specification does not actually narrow the search, the processing load on the servers increases.
Increasing the number of floatable server candidates	In searches involving many data items, the HiRDB system uses all floatable servers so that the search can be performed efficiently.	If the number of data items is small, the concurrent execution performance for SQL statements drops because the HiRDB system uses all floatable servers. Also, when there are many table partitions, the communication load increases because the communication paths between the servers become complex.

(4) Allocating floatable servers at each optimization

(a) When the optimization method is omitted

The following figure shows floatable server allocation when the optimization method is omitted.

Figure 4-18: Floatable server allocation when the optimization method is omitted


BES1
BES2
BES3
Table definitions or join results
FLT1
FLT2
BES: Back-end server
FLT : Floatable server
▨ : Floatable server candidate

Explanation

If increasing the number of floatable server candidates is not specified, the HiRDB system determines the number of floatable servers that is necessary and allocates them from `FLT1` and `FLT2`.

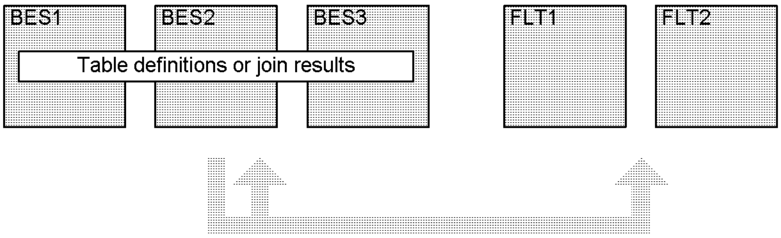
If increasing the number of floatable server candidates is specified, the HiRDB system allocates both `FLT1` and `FLT2` as floatable servers.

Apply this optimization method if the system has back-end servers in which no tables are defined, multiple SQL statements fetch the same data, and the back-end servers for fetching data are to be used only for fetching data.

(b) When increasing the target floatable servers (back-end servers for fetching data) is applied

The following figure shows floatable server allocation when increasing the target floatable servers (back-end servers for fetching data) is applied.

Figure 4-19: Floatable server allocation when increasing the target floatable servers (back-end servers for fetching data) is applied



BES: Back-end server
FLT : Floatable server
▨ : Floatable server candidate

Explanation

If increasing the number of floatable server candidates is not specified, the HiRDB system determines the number of floatable servers necessary and allocates them from `BES1`, `BES2`, `BES3`, `FLT1`, and `FLT2`. However, all of these servers are not necessarily allocated.

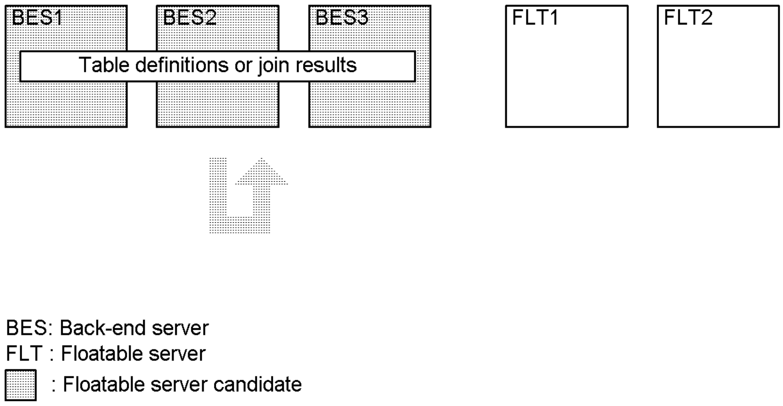
If increasing the number of floatable server candidates is specified, the HiRDB system allocates `BES1`, `BES2`, `BES3`, `FLT1`, and `FLT2` as floatable servers.

Apply this optimization method if the SQL statements are executed individually, and all back-end servers are to be used efficiently.

(c) When limiting the target floatable servers (back-end servers for fetching data) is applied

The following figure shows floatable server allocation when limiting the target floatable servers (back-end servers for fetching data) is applied.

Figure 4-20: Floatable server allocation when limiting the target floatable servers (back-end servers for fetching data) is applied


BES1
BES2
BES3
Table definitions or join results
FLT1
FLT2
BES: Back-end server
FLT : Floatable server
▨ : Floatable server candidate

Explanation

If increasing the number of floatable server candidates is not specified, the HiRDB system determines the number of floatable servers necessary and allocates them from `BES1, BES2`, and `BES3`. However, all of these servers are not necessarily allocated.

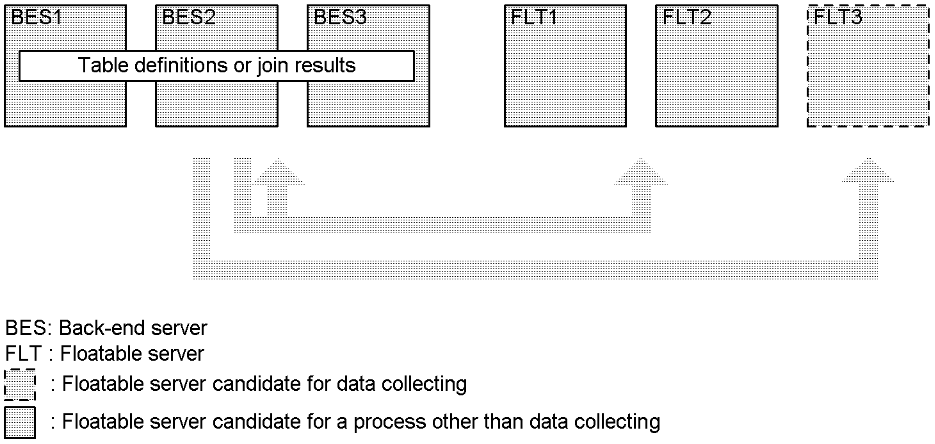
If increasing the number of floatable server candidates is specified, the HiRDB system allocates `BES1`, `BES2`, and `BES3` as floatable servers.

Apply this optimization method if multiple SQL statements are to be executed, each search process accesses tables defined in a different back-end server, and the back-end servers used for the individual tables are to be operated separately.

(d) When separating data collecting servers is applied

The following figure shows floatable server allocation when separating data collecting servers is used.

Figure 4-21: Floatable server allocation when separating data collecting servers is applied


BES1
BES2
BES3
Table definitions or join results
FLT1
FLT2
FLT3
BES: Back-end server
FLT : Floatable server
⬚ : Floatable server candidate for data collecting
▨ : Floatable server candidate for a process other than data collecting

Explanation

For SQL statements that must collect data (data collecting) from multiple BES units into one BES, the HiRDB system allocates `FLT2` from `FLT1`, `FLT2`, and `FLT3` as the server for data collecting. If an SQL statement executes data collecting several times, the HiRDB system always allocates this data collecting server (`FLT3`). If a process other that data collecting is to be executed and increasing the number of floatable server candidates is not specified, the HiRDB system determines the number of floatable servers that is necessary and allocates them from `BES1`, `BES2`, `BES3`, `FLT1`, and `FLT2`. However, all of these servers are not necessarily allocated.

If increasing the number of floatable server candidates is specified, the HiRDB system allocates `BES1`, `BES2`, `BES3`, `FLT1`, and `FLT2` as floatable servers.

Apply this optimization method if the SQL statements do not contain data collecting processes, and increasing the target floatable servers (back-end servers for fetching data) is applied.

4.5.5 Grouping processing methods (HiRDB/Parallel Server only)

The following optimization methods affect grouping processing:

- Rapid grouping processing
- Group processing, `ORDER BY` processing, and `DISTINCT` set function processing at the local back-end server

If the HiRDB system judges that sort and hash processing for grouping are unnecessary, it selects a method that can process the data faster. For details about grouping processing, see the *HiRDB Version 9 Command Reference* manual.

The following table describes the optimization features related to grouping processing methods.

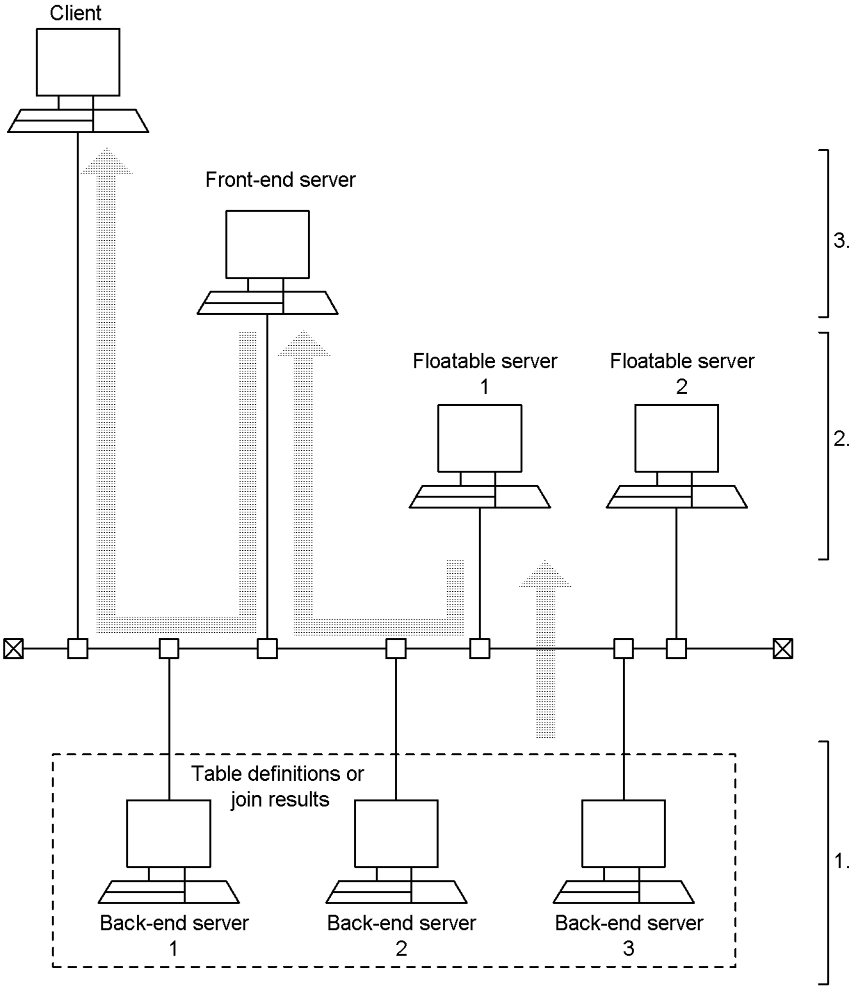
Table 4-9: Optimization features related to grouping processing methods

Optimization method (type of grouping processing method)	Advantages	Disadvantages
Omitted (`FLOATABLE SORT`)	Data can be searched rapidly if the data count values are unevenly distributed among the back-end servers and grouping does not reduce the data count.	If the group count is small and the data count is high, the performance drops because the communication volume increases.
Rapid grouping processing (`HASH`)	Data can be searched rapidly when the group count is small.	If the group count is large, the performance drops because the data is grouped by hashing.
Group processing, `ORDER BY` processing, and `DISTINCT` set function processing at the local back-end server (`LIST SORT`)	Data can be searched rapidly if grouping substantially decreases the data count. Data can also be searched rapidly if the data is grouped by partitioning keys.	If the data count values are unevenly distributed among the back-end servers because of sorting in each back-end server, the performance drops because processing takes a long time in servers that have a high data count.

(a) Grouping processing when the optimization method is omitted

The following figure shows grouping processing when the optimization method is omitted.

Figure 4-22: Grouping processing method when the optimization method is omitted



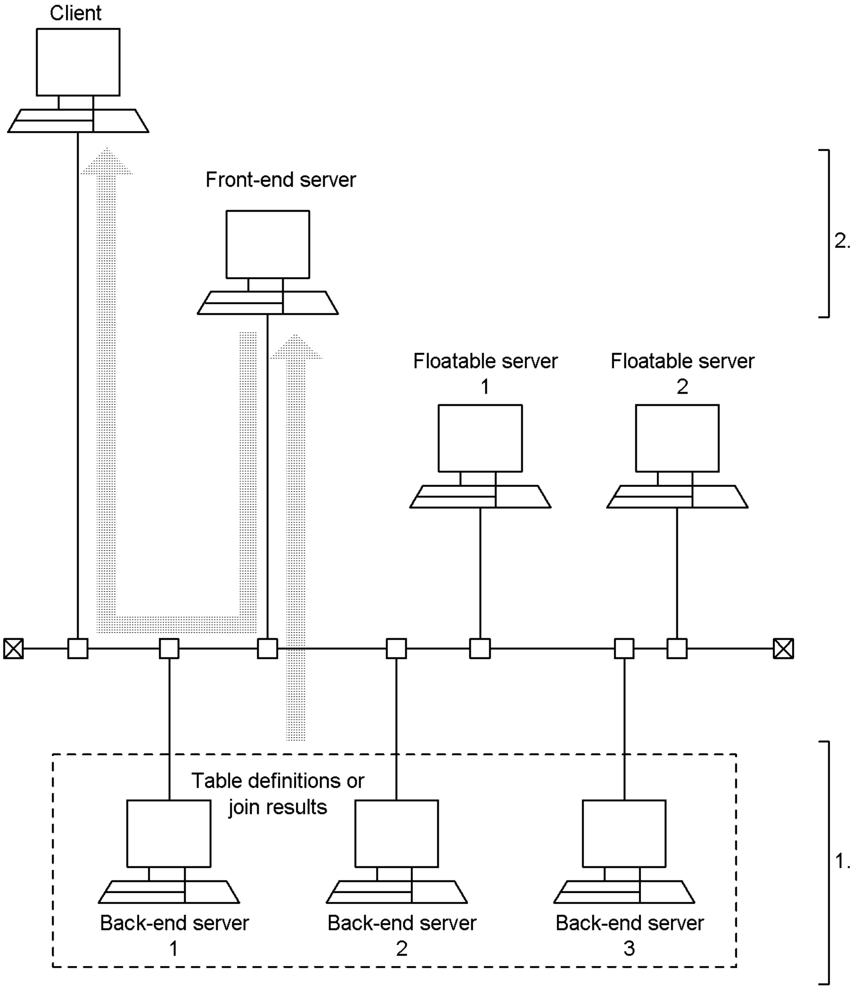
Explanation

1. The back-end servers perform only data fetching.
2. The floatable servers perform only sorting and grouping by grouping columns.
3. The front-end server collects grouping processing results and transfers the results to the client.

(b) Grouping processing when rapid grouping processing is applied

The following figure shows grouping processing when rapid grouping processing is applied.

Figure 4-23: Grouping processing when rapid grouping processing is applied



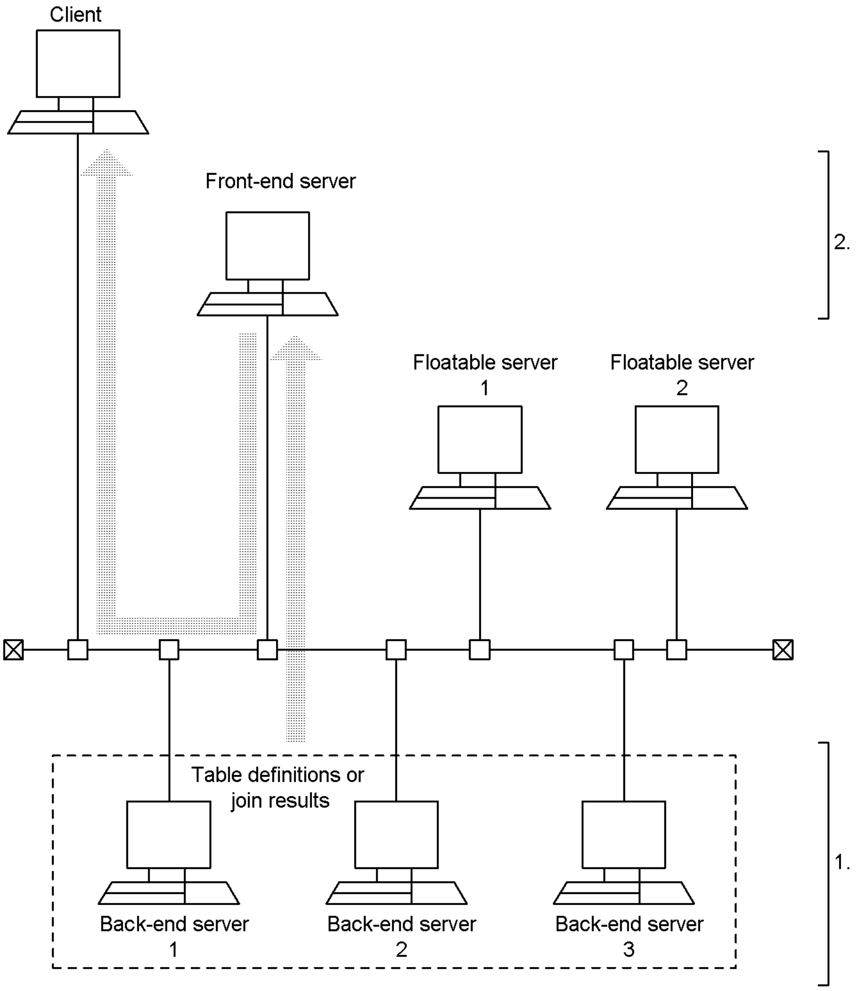
Explanation

1. The back-end servers fetch data, and then hash and group the data by grouping columns. (The floatable servers are not used.)
2. The front-end server collects grouping results from each back-end server, regroups the combined data, and transfers the results to the client.

(c) Grouping processing when group processing, ORDER BY processing, and DISTINCT set function processing are applied at the local back-end server

The following figure shows grouping processing when group processing, ORDER BY processing, and DISTINCT set function processing are applied at the local back-end server. However, this diagram shows the processing when one table is searched.

Figure 4-24: Grouping processing when group processing, ORDER BY processing, and DISTINCT set function processing are applied at the local back-end server



Explanation

1. The back-end servers fetch data, and then sort and group the data by grouping columns. (The floatable servers are not used.)
2. The front-end server collects grouping results from each back-end server, regroups the combined data, and transfers the results to the client.

4.5.6 Join methods

(1) Join method types

The following table describes the join method types (except direct product) and their features. If the join methods found in this table cannot be applied, direct product is applied.

Table 4-10: Join method types and features

Join method	Processing method	Initial data fetching	Advantages	Disadvantages
Merge join	This method sorts the data by join column and executes matching in sequence from the smallest value in the join column.	Slow	The performance degradation is small compared with other methods because even tables with many hits can be joined with a small amount of memory. Data can be searched rapidly if the join column data has already been sorted, and the sort processing for merge join can be cancelled.	If the data in the columns to be joined has not been sorted, the sort processing load increases, and the performance drops.
Nested-loops-join	This method uses join column values from the outer table, searches the index defined in the join column of the inner table, and repeatedly processes nested matches.	Fast	Data can be searched rapidly if the inner table can be narrowed with the index specified in the join column.	If the hit count of the outer table is high, the performance drops because the index is used to search the inner table each time a row is fetched from the outer table.

Join method	Processing method	Initial data fetching	Advantages	Disadvantages
Hash join	This method creates a hash table from the join column of the inner table, hashes the join column of the outer table, and executes matching with the hash table that was created from the inner table.	Fast if the number of hits in the inner table is small (slower than next-loop-join, but faster than merge join)	Data can be searched rapidly when the hit count is low for the inner table and high for the outer table.	If the hit count in the inner table is high, memory usage becomes high. The performance drops because the hits for which memory is unavailable are first saved to a file.
`SELECT-APSL`	If a condition contains a `?` parameter, this method prepares several join method candidates, and determines the optimal search method when the value of the `?` parameter is input.	Differs depending on the search method that is selected	The optimal search method can be selected when the value of the `?` parameter is input.	The optimizing information collection utility (`pdgetcst`) must be executed.# Also, the SQL object size becomes large because several join methods are prepared.

#: In some cases, the optimal access path cannot be selected even if the optimizing information collection utility is executed. For details about the necessity of executing the optimizing information collection utility, see the manual *HiRDB Version 9 Command Reference* and verify the performance.

(2) Processing methods

(a) Merge join

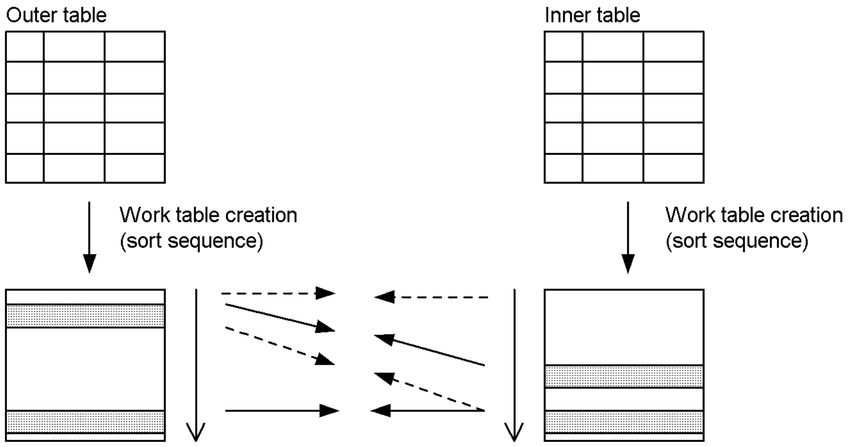
Merge join is effective when the outer table cannot be narrowed very much.

`SORT MERGE JOIN`

This join method fetches rows from the outer and inner table, creates the respective work tables, and sorts the data. The join method then joins the rows if the join condition is satisfied.

The following figure shows the processing of `SORT MERGE JOIN`.

Figure 4-25: Processing of SORT MERGE JOIN

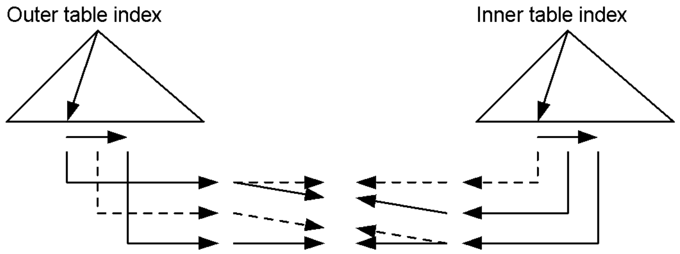


`KEY SCAN MERGE JOIN`

This join method system fetches rows from the outer and inner table by using `KEY SCAN`. The join method then joins the rows if the join condition is satisfied.

The following figure shows the processing of `KEY SCAN MERGE JOIN`.

Figure 4-26: Processing of KEY SCAN MERGE JOIN

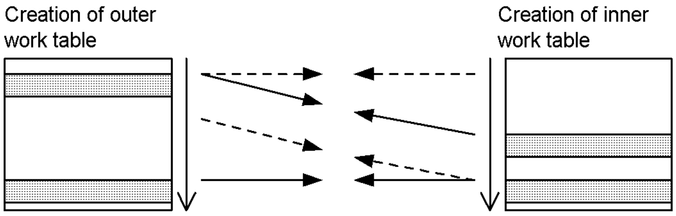


`LIST SCAN MERGE JOIN`

This join method creates work tables from the outer and inner tables, and fetches rows in ascending join column order without sorting the data beforehand. The join method then joins the rows if the join condition is satisfied.

The following figure shows the processing of `LIST SCAN MERGE JOIN`.

Figure 4-27: Processing of LIST SCAN MERGE JOIN



`L-KEY R-LIST MERGE JOIN`

This join method fetches rows from the outer table by using `KEY SCAN`. The method creates a work table for the inner table and fetches rows without first sorting the data. The join method then joins the rows if the join condition is satisfied.

`L-KEY R-SORT MERGE JOIN`

This join method system fetches rows from the outer table by using `KEY SCAN`. The join method creates a work table for the inner table, sorts the data, and fetches rows. The join method then joins the rows if the join condition is satisfied.

`L-LIST R-KEY MERGE JOIN`

This join method creates a work table for the outer table and fetches rows without first sorting the data. The join method fetches rows from the inner table by using `KEY SCAN`. The join method then joins the rows if the join condition is satisfied.

`L-LIST R-SORT MERGE JOIN`

This join method creates a work table for the outer table and fetches rows without first sorting the data. The join method creates a work table for the inner table, sorts the data, and fetches rows. The join method then joins the rows if the join condition is satisfied.

`L-SORT R-KEY MERGE JOIN`

This join method creates a work table for the outer table, sorts the data, and fetches rows. The join method fetches rows from the inner table by using `KEY SCAN`. The join method then joins the rows if the join condition is satisfied.

`L-SORT R-LIST MERGE JOIN`

This join method creates a work table for the outer table, sorts the data, and fetches rows. The join method creates a work table for the inner table, sorts the data, and fetches rows. The join method then joins the rows if the join condition is satisfied.

(b) Nested-loops-join

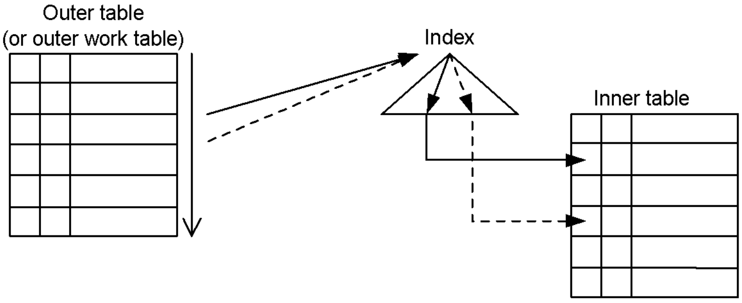
Nested-loops-join is effective if an index is defined in the inner table, and the outer table can be narrowed significantly

`NESTED LOOPS JOIN`

This join method fetches rows one at a time from the outer table, matches them to individual rows in the inner table, and executes nested loop processing that fetches rows that satisfy the join condition.

The following figure shows the processing of `NESTED LOOPS JOIN`.

Figure 4-28: Processing of NESTED LOOPS JOIN



Note

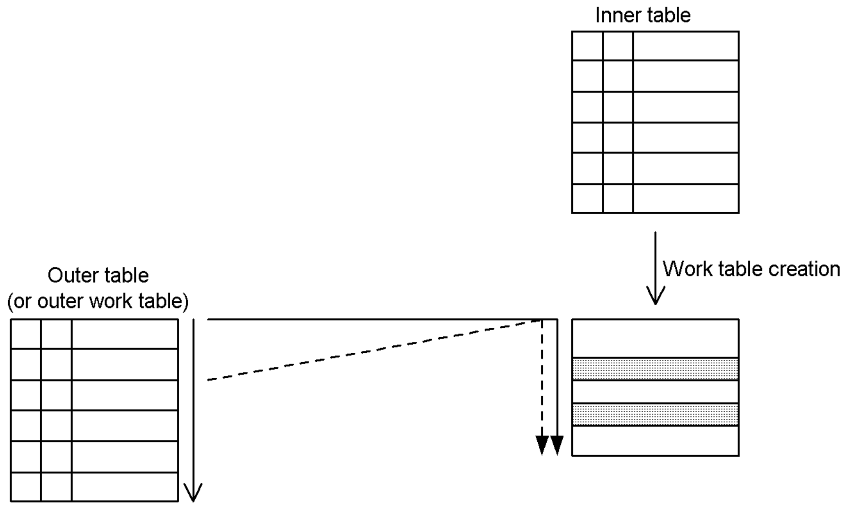
In some cases, an index is used when the outer table is searched.

`R-LIST NESTED LOOPS JOIN`

This join method fetches rows from the inner table and creates a work table. The join method then fetches rows one at a time from the outer table, matches the work table that was created from the inner table to those individual rows, and executes nested loop processing that fetches rows that satisfy the join conditions.

The following figure shows the processing of `R-LIST NESTED LOOPS JOIN`.

Figure 4-29: Processing of R-LIST NESTED LOOPS JOIN



Note

In some cases, an index is used when the outer table is searched.

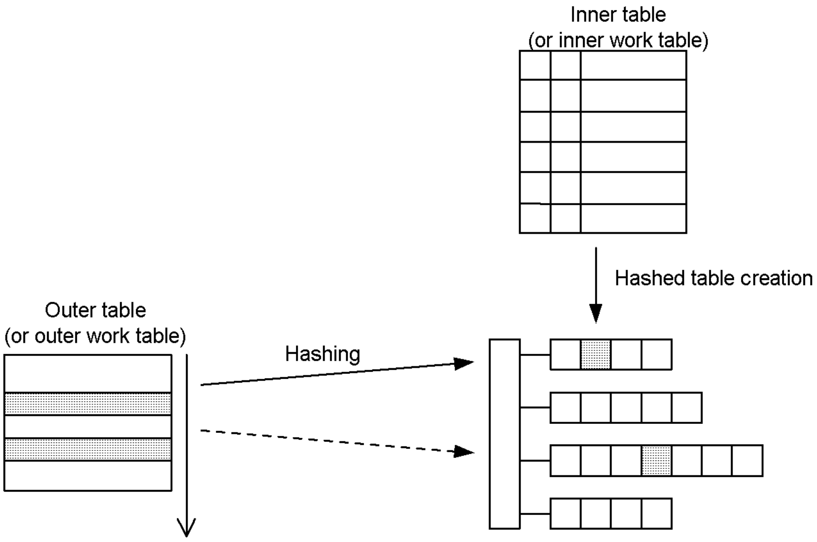
(c) Hash join

HASH JOIN

This join method first hashes the inner table with the join column values and creates a hash table. The join method then hashes the outer table with the join column values each time a row is fetched, and matches the outer table with the hash table that was created from the inner table.

The following figure shows the processing of HASH JOIN.

Figure 4-30: Processing of HASH JOIN



There are four hash join processing methods. The following table describes the hash join processing methods and features.

Table 4-11: Hash join processing methods and features

Processing method	Description	Advantages	Disadvantages	Selection method
Batch hash join	This method performs hash join by expanding the hash table that was created from the inner table into the buffer area for all work tables.	Hash join can be processed rapidly because this method expands the entire hash table in the work table buffer area before executing hash join.	If the hash table for the inner table is large, the system's capability to execute SQL statements simultaneously is diminished because the work table work area becomes large.	Change the hash table size.#

Processing method	Description	Advantages	Disadvantages	Selection method
Bucket partitioning hash join	This method partitions the inner and outer tables into several buckets, creates a hash table from some of the inner table buckets, and expands it in the work table buffer area. This method then saves the remaining buckets in a work table file and reads the contents of inner table buckets that were expanded in the work table buffer area and the outer table buckets of the area with the same value. The method then expands the inner table from the work table file to the work table buffer area a little at a time, and executes hash join. The amount of memory used becomes small, and the processing performance drops slightly.	Hash join can be executed in environments that have a small work table buffer area.	Because the rows of the inner and outer table are first saved to a work table file, the performance drops compared to when hash join is executed with only work table buffer area.	Change the hash table size.#
Continuous hash join	When three or more tables are searched, this method creates hash tables from the tables, except the outermost table, expands the hash tables in the work table buffer area, and executes hash join in succession. The amount of memory used becomes large, and the processing performance improves.	Hash join can be processed rapidly because this method expands the entire hash table in the work table buffer area before executing hash join. Also, hash join can be processed rapidly when only the outermost table is large.	When the number of tables to be executed becomes large, the work table buffer area that is used becomes large.	This method cannot be selected. The HiRDB system automatically selects the optimal method based on the number of table rows.

Processing method	Description	Advantages	Disadvantages	Selection method
Intermittent hash join	When three or more tables are searched, this method executes hash join by saving the join results to a work table file each time tables or work tables are joined.	Hash join involving three or more tables can be executed even in environments that have a small work table buffer area.	The number of I/O operations increases and performance drops because the join results are first saved to a file each time tables or work tables are joined.	This method cannot be selected. The HiRDB system automatically selects the optimal method based on the number of table rows.

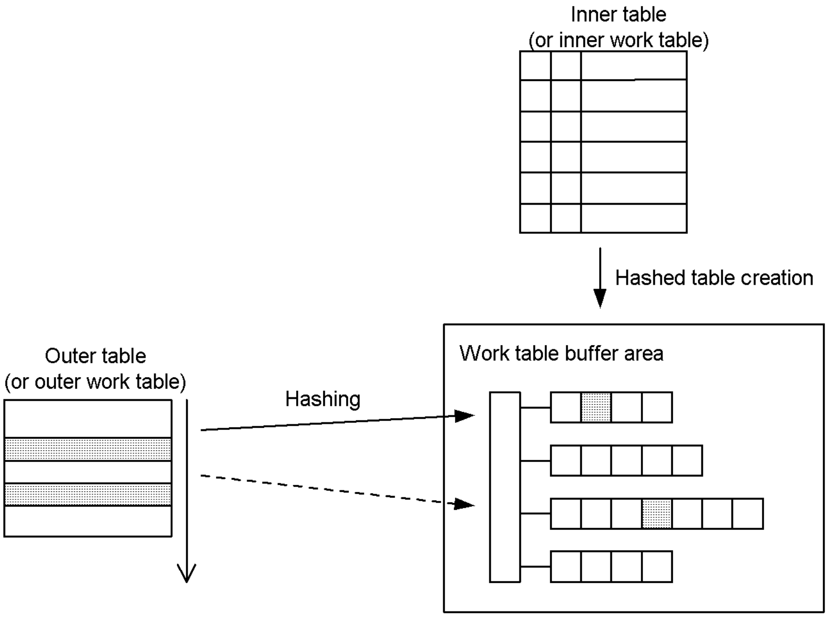
#: For details about how to change the hash table size, see *4.5.10 Preparing for application of hash join and subquery hash execution*.

The processing methods are summarized as follows.

- **Batch hash join**

This processing method expands the entire hash table created from the inner table in the work table buffer area and then executes hash join. The following figure shows the processing method of batch hash join.

Figure 4-31: Processing method of batch hash join



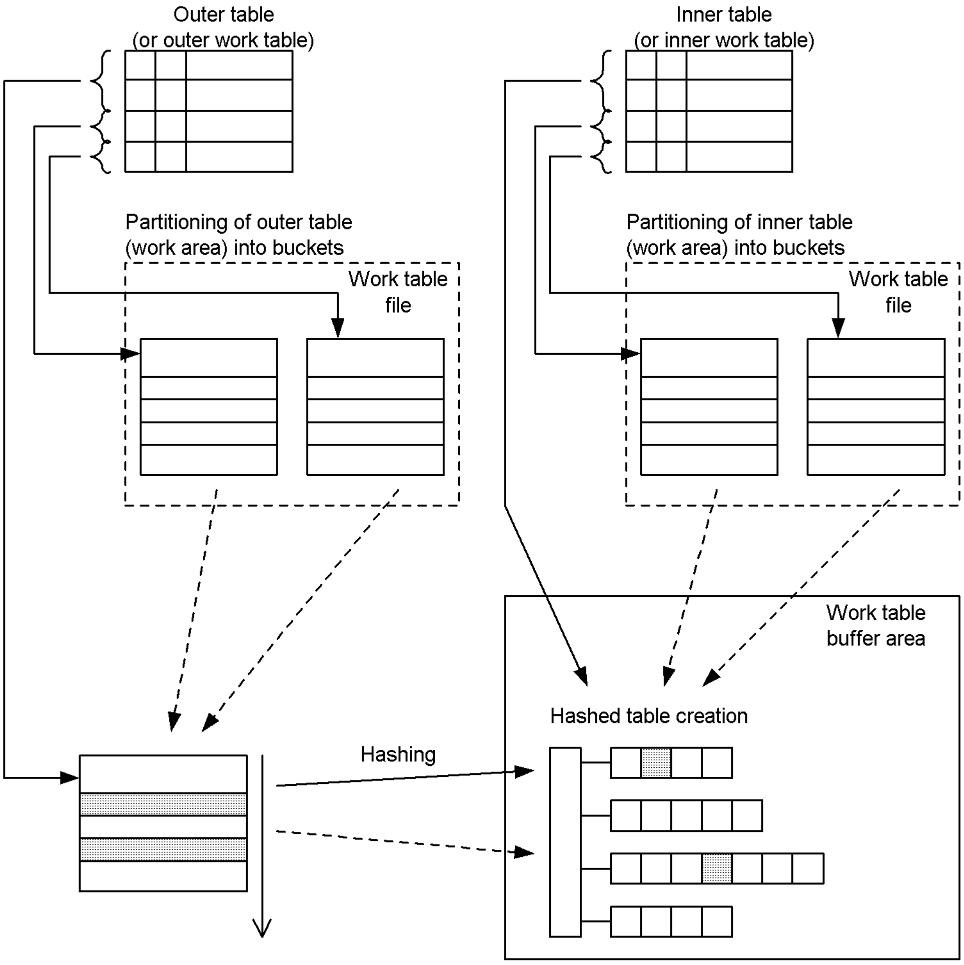
- **Bucket partitioning hash join**

This processing method partitions the inner and outer table into buckets, expands part of the inner table into the work table buffer area, and saves the remaining sections to a work table file.

Bucket partitioning refers to hashing a table with join row values and partitioning the table into multiple small tables. Join processing is executed on an inner table section that was expanded in the work file buffer area. First, a hash table is created from the inner table, and rows are fetched one at a time from the outer table and then merged and joined with the hash table that was created from the inner table. After joining of the tables found in the work table buffer area is completed, the buckets of the outer and inner tables are expanded from the work table files into the work table buffer area, and join processing is executed in the same manner. Processing ends after all tables are expanded into the work table buffer area and joined.

The following figure shows the processing method of bucket partitioning hash join.

Figure 4-32: Processing method of bucket partitioning hash join



- **Continuous hash join**

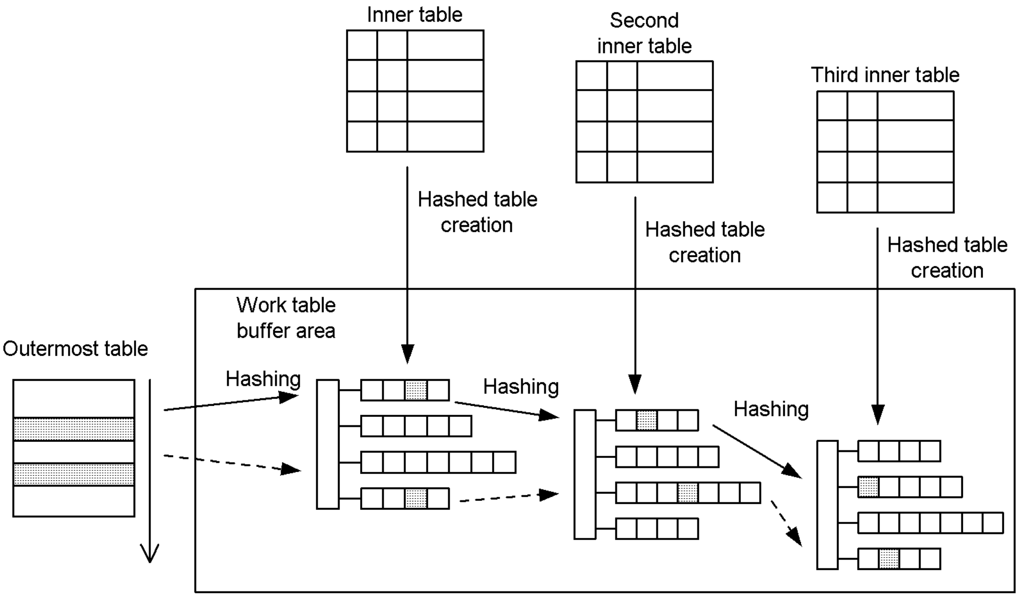
This processing method is applied to searches involving three or more tables.

First, hash tables are created from all target tables except the outermost table and expanded into the work table buffer area. Next, a row is fetched from the outer table, hashed, and then matched and joined with the hash table that was created from the inner table. If the join condition is satisfied, the row is hashed with the join result, and then matched and joined with the hash table.

When joining is completed to the last row or when the condition becomes false, processing returns to the outermost table, the next row is fetched, and join processing is repeated in the same manner. During join processing, if there is a location where the join key value in the inner table is duplicated, processing returns to that location and join processing is repeated. When processing of all duplicate key values is completed, processing returns to the outermost table, the next row is fetched, and join processing is repeated in the same manner.

The following figure shows the processing method of a continuous hash join.

Figure 4-33: Processing method of continuous hash join



■ Intermittent hash join

This processing method is applied to searches involving three or more tables.

First, a hash table is created from the inner table of the first join and expanded in the work table buffer area. Next, rows are fetched one at a time from the outer table, hashed with the join column values of the outer table, and then matched and joined with the hash table that was created from the inner table. After all lines from the outer table have been fetched and joined, processing proceeds to the next join process.

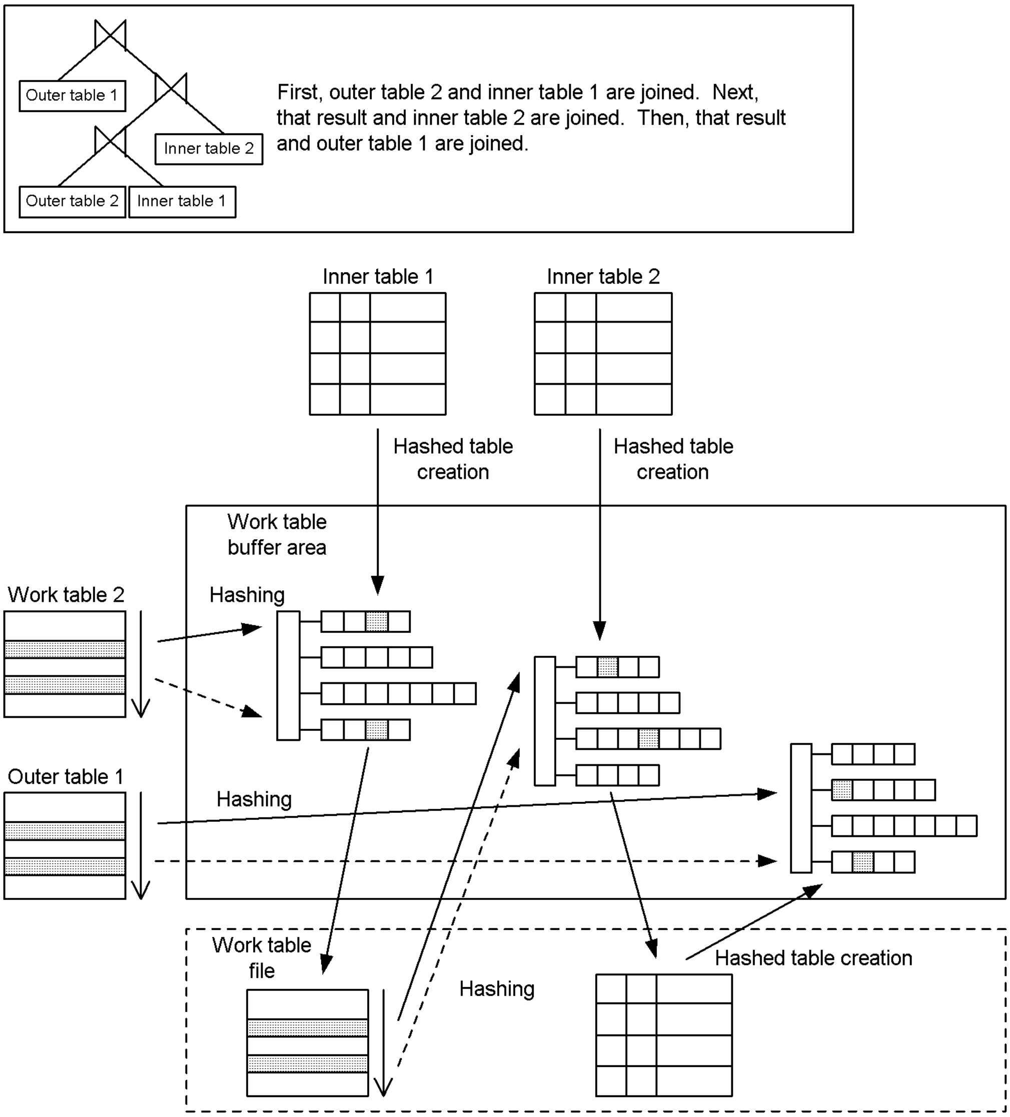
The processing changes depending on whether the join result becomes the outer table or the inner table.

If the join result becomes the outer table, a hash table is created from the next inner table to be joined, and rows are fetched one at a time from the join results and then matched and joined with the hash table that was created from the inner table.

If the join result becomes the inner table, a hash table is created from the join results, and rows are fetched one at a time from the outer table and then matched and joined with the hash table that was created from the join results.

The figure below shows the processing method of intermittent hash join. In the example shown for this processing method, that tables are joined as follows: outer table 1 → ((outer table 2 → inner table 1) → inner table 2).

Figure 4-34: Processing method of intermittent hash join



First, outer table 2 and inner table 1 are joined. Next, that result and inner table 2 are joined. Then, that result and outer table 1 are joined.

(d) SELECT-APSL

`SELECT-APSL` is a method that dynamically determines the join method during SQL execution.

`SELECT-APSL` (HiRDB/Parallel Server only)

If the conditions include the `?` parameter, the optimal join method may change depending on the value of the `?` parameter. Also, if the value of the `?` parameter cannot be determined during SQL optimization processing, the optimal join method cannot be determined. The system therefore determines the join method by calculating the hit ratio during SQL execution.

`SELECT-APSL` is described as follows based on a display example of the access path display utility (`pdvwopt`).

Condition	`T1(outer-table).C1=? parameter`
Reference value	`0.047`

[1] Nest-loop-join

[2] Merge join

Explanation

- If the hit rate of the predicate `T1(outer-table).C1=? parameter` is less than the reference value (`0.047`), nested-loops-join is selected during execution because the hit rate is small and the outer table can be narrowed substantially.
- If the hit rate of the predicate `T1(outer-table).C1=? parameter` is equal to or greater than the reference value (`0.047`), merge join is selected during execution because the hit rate is large and the outer table cannot be narrowed very much.

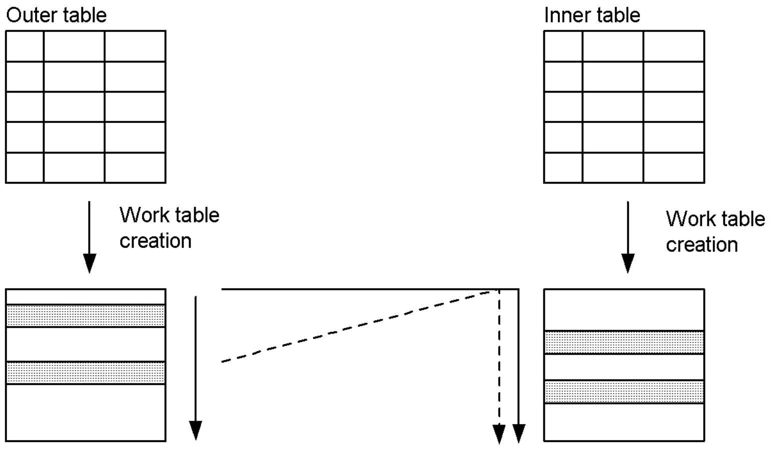
(e) Cross join

`CROSS JOIN`

The `CROSS JOIN` process method combines and joins all rows of the outer table and all rows of the inner table. If there are conditions that apply across both tables, the conditions are judged after the tables are joined.

The following figure shows the `CROSS JOIN` processing method.

Figure 4-35: CROSS JOIN processing method



Note

Depending on the condition, sometimes a work table is not created.

4.5.7 Search Methods

(1) Search method types

The table below describes the search method types (except `LIST SCAN` and `ROWID FETCH`) and their features.

`LIST SCAN` is applied when a work table is created and searched, for example, in a viewed table search or a `WITH` clause query expression. `ROWID FETCH` is applied when a cursor is used.

Table 4-12: Search method types and features

Search method	Processing method	Advantages	Disadvantages
Table scan(`TABLE SCAN`)	This method sequentially searches the pages (data pages) in which the table is stored and references all rows. The initial data fetch is fairly slow.	When all data items are to be searched, the data can be searched rapidly. The data can be searched rapidly even if the search cannot be narrowed with an index.	Even if the search results can be narrowed by conditions, the performance is poor because all data pages are referenced.

Search method	Processing method	Advantages	Disadvantages
Index scan (INDEX SCAN, MULTI COLUMNS INDEX SCAN, PLUGIN INDEX SCAN)	This method executes a binary search of the index, and then each time it retrieves the row identifier of a target data item, it references the database row indicated by that row identifier. The initial data fetch is fast.	The data can be searched rapidly when the search can be narrowed with an index. The data can be searched rapidly even if a cluster key index is used and the search cannot be narrowed very much. The sort processing might be omitted.#1	If the search cannot be narrowed very much with an index, the number of random I/O operations performed on the data pages increases, and performance drops.
Key scan(KEY SCAN, MULTI COLUMNS KEY SCAN, PLUGIN KEY SCAN)	This method executes a binary search of the index and references only the data found in the index (configuration column values or row identifiers of the index). This method is applied when only the configuration columns or row identifiers of the index are to be referenced. The initial data fetch is fast.	Even if the search cannot be narrowed very much with an index, the data can be searched rapidly because only the index pages are referenced and there is no data page input or output. The sort processing might be omitted.#1	None
SELECT-APSL	If a condition contains a ? parameter, this method prepares several join method candidates, and determines the optimal search method when the value of the ? parameter is input. The speed of the initial data fetch differs according to the search method that is actually selected.	When the value of the ? parameter is input, the optimal search method can be selected by considering the narrowing rate obtained with the index.	The optimizing information collection utility (pdgetcst) must be executed.#2 Also, the SQL object size becomes large because several search candidates are prepared.
AND multiple index usage(AND PLURAL INDEXES SCAN)	This method uses multiple indexes, creates multiple work tables, combines product sets, sum sets, and difference sets between the work tables to obtain results. The initial data fetch is slow.	Because the results are obtained by combining, product sets, sum sets, and difference sets, indexes can be used in evaluating the data even when multiple conditions are specified.	This method creates several work tables and sorts the data in each work table. Thus, if the search cannot be narrowed with an index, the performance drops because the number of items to be sorted is large.

Search method	Processing method	Advantages	Disadvantages
OR multiple index usage(OR PLURAL INDEXES SCAN)	This method stores results retrieved by using multiple indexes into one work table, and executes duplicate elimination at the end to obtain results. The initial data fetch is slow.	The data can be searched rapidly if narrowing with an index is possible for the individual search conditions that are combined with the OR operator.	This method uses multiple indexes to search the data, stores the results in one work table, sorts the results, and executes duplicate elimination. Thus, if there are many data items before duplicate elimination, the performance drops.

#1: If indexes can be sorted by ORDER BY or GROUP BY processing, data is retrieved without being sorted. By checking the access path, you can determine whether or not sort processing was omitted.

#2: In some cases, the optimal access path cannot be selected even if the optimizing information collection utility is executed. For details about the necessity of executing the optimizing information collection utility, see the manual *HiRDB Version 9 Command Reference* and verify the performance.

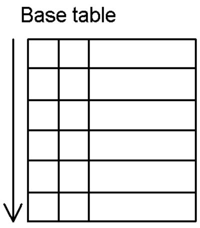
(2) Processing methods

(a) Search using no index

TABLE SCAN

This processing method searches the database pages of a table without using an index. The following figure shows the TABLE SCAN processing method.

Figure 4-36: TABLE SCAN processing method

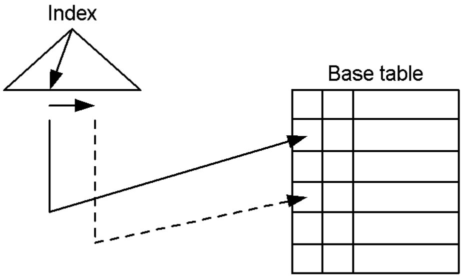


(b) Search using one index

INDEX SCAN

This processing method searches the index pages of a single-column index to narrow the search and then searches the data pages of the table. The following figure shows the INDEX SCAN processing method.

Figure 4-37: INDEX SCAN processing method

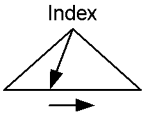


`KEY SCAN`

This processing method searches only the index pages of a single-column index. The data pages are not searched.

The following figure shows the `KEY SCAN` processing method.

Figure 4-38: KEY SCAN processing method

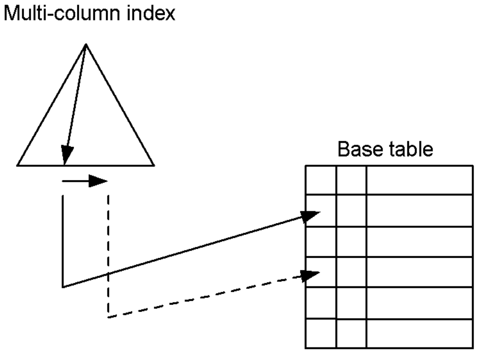


`MULTI COLUMNS INDEX SCAN`

This processing method searches the index pages of a multi-column index to narrow the search and then searches the data pages of the table.

The following figure shows the `MULTI COLUMNS INDEX SCAN` processing method.

Figure 4-39: MULTI COLUMNS INDEX SCAN processing method

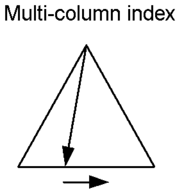


`MULTI COLUMNS KEY SCAN`

This processing method searches only the index pages of a multi-column index. The data pages are not searched.

The following figure shows the `MULTI COLUMNS KEY SCAN` processing method.

Figure 4-40: MULTI COLUMNS KEY SCAN processing method

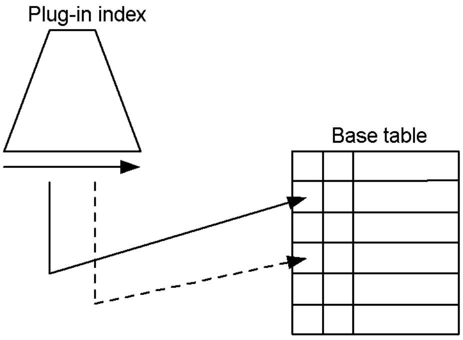


`PLUGIN INDEX SCAN`

This processing method uses a plug-in index to narrow the search and then searches the data pages of the table.

The following figure shows the `PLUGIN INDEX SCAN` processing method.

Figure 4-41: PLUGIN INDEX SCAN processing method



Note

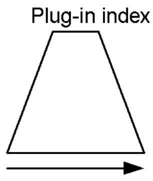
The structure of the plug-in index differs according to the plug-in.

`PLUGIN KEY SCAN`

This processing method searches only the index pages of a plug-in index. The data pages are not searched.

The following figure shows the `PLUGIN KEY SCAN` processing method.

Figure 4-42: PLUGIN KEY SCAN processing method



Note

The structure of the plug-in index differs according to the plug-in.

(3) SELECT APSL

`SELECT-APSL` (for HiRDB/Parallel Server)

If the conditions include the `?` parameter, the optimal search method may change depending on the value of the `?` parameter. Also, if the value of the `?` parameter cannot be determined during preprocessing, the optimal search method cannot be determined. The system therefore determines the search method by calculating the hit ratio during SQL execution.

(4) Search using multiple indexes

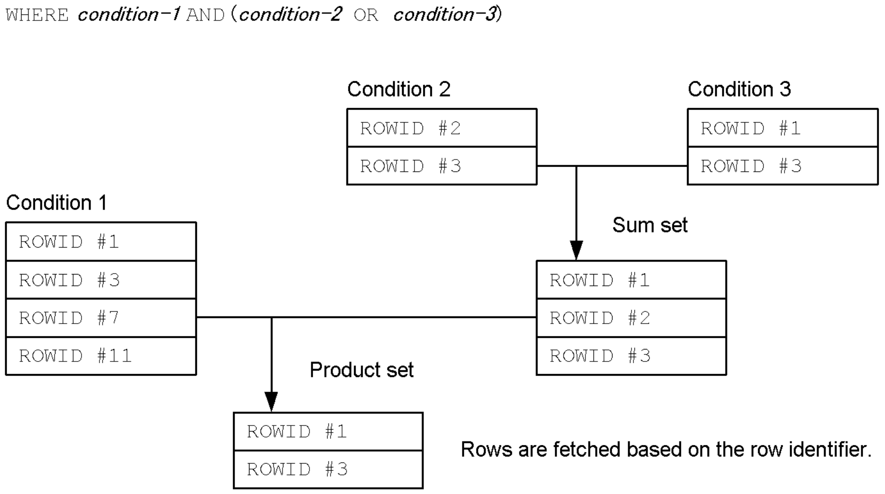
`AND PLURAL INDEXES SCAN`

For search conditions that are combined with the `AND` or `OR` operator, the respective indexes are used to conduct the search, and the row identifiers (`ROWID`) are stored in the respective work tables. These work tables are consolidated into a single work table by forming a product set when the `AND` operator is used, a sum set when the `OR` operator is used, and a difference set when the `ANDNOT` operator (can only be used in the `ASSIGN LIST` statement) is used. Then rows are fetched based on the row identifiers of this work table.

When creating a work table of row identifiers from each condition, HiRDB sometimes uses `TABLE SCAN` to create the work table, even if the condition column does not have an index.

The following figure shows the `AND PLURAL INDEXES SCAN` processing method.

Figure 4-43: AND PLURAL INDEXES SCAN processing method



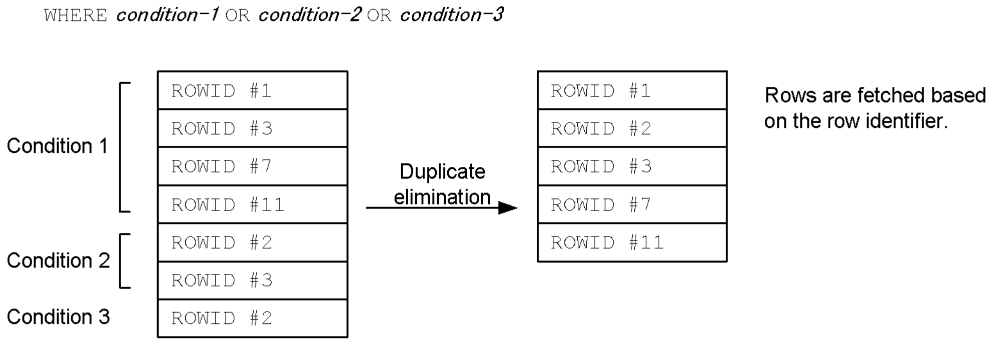
`OR PLURAL INDEXES SCAN`

For search conditions that are combined with the `OR` operator, the respective indexes are used to conduct the search, and the row identifiers (`ROWID`) are stored in one work table. After the duplicate rows in the work table are eliminated by duplicate elimination, the rows are fetched based on the row identifiers.

When creating a work table of row identifiers from each condition, HiRDB sometimes uses `TABLE SCAN` to create the work table, even if the condition column does not have an index.

The following figure shows the `OR PLURAL INDEXES SCAN` processing method.

Figure 4-44: OR PLURAL INDEXES SCAN processing method



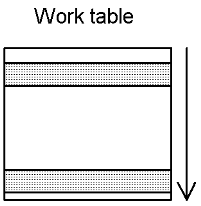
(5) Search of internally created work table

LIST SCAN

This processing method searches a work table that was created internally.

The following figure shows the LIST SCAN processing method.

Figure 4-45: LIST SCAN processing method



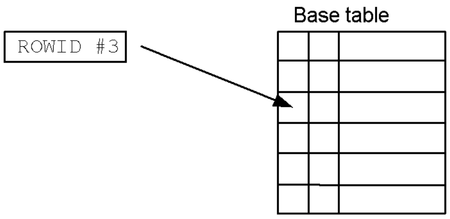
(6) Search using a row identifier

ROWID FETCH

This processing method searches a table by using row identifiers (ROWID) as keys. If the row does not have to be fetched, a search is not executed.

The following figure shows the ROWID FETCH processing method.

Figure 4-46: ROWID FETCH processing method



4.5.8 Execution of subqueries with no external references

(1) Execution method types

Table 4-13 describes the execution formats and features of inquiries that do not have external references. *Table 4-14* describes the optimal execution methods of queries that do not have external references.

Table 4-13: Execution methods and features of subqueries with no external references

Execution method	Processing method	Advantages	Disadvantages
Work table ATS execution	This method obtains the subquery results beforehand and creates a work table. Then when a search using an index is conducted for an external query, this method uses the work table that was created from the subquery results to narrow the search range.	An index can be used for an external query. Therefore, when the number of subquery hits is small and the number of external queries is large, data can be searched rapidly when an index is used to narrow the search range.	When the number of subquery hits is large, the performance drops because a search using an index for the external query must be performed for each row in the subquery results.
Work table execution	This method obtains the subquery results beforehand and creates a work table. Then each time a row of the external query is searched, this method matches the row with the work table that was created from the subquery results and evaluates the predicate that contains the subquery.	This method can be applied to all subquery conditions that require a work table.	The performance drops when the number of external queries is large.
Row value execution	This method obtains the subquery beforehand. (A work table is not created.) Then, when an external query is searched, this method uses the values of the subquery results to evaluate the condition that includes the subquery.	An index can be used for external queries. Therefore, if the number of external queries is large, an index can be used to narrow the search range, and data can be searched rapidly.	The performance drops when the number of external queries is high and the predicates that include subqueries cannot be narrowed using an index.
Hash execution	This method creates a hash table from the subquery results beforehand. Then each time a row of the external query is retrieved, this method hashes the external query value and evaluates the condition that includes the subquery.	Data can be searched rapidly when the number of subquery hits is small and the number of external queries is large.	If the number of subquery hits is large, the work table buffer size to be used becomes large. Although the work table buffer size to be used can be specified, the buffer data must be saved to a work table file when the work table buffer becomes full, and consequently the performance drops.

Table 4-14: Optimal execution method of subqueries with no external references

Subquery	Optimal execution method	
Table subqueries specified on the right side of the =ANY and =SOME quantified predicates and the IN predicate	The method differs depending on the number of data items in the external query or subquery.	
	External queries: Many Subqueries: Few	Work table ATS execution or hash execution is effective.
	External queries: Intermediate Subqueries: Few	
	External queries: Few Subqueries: Few	
	External queries: Many Subqueries: Intermediate	Hash execution is effective
	External queries: Intermediate Subqueries: Intermediate	
	External queries: Few Subqueries: Intermediate	Hash execution or work table execution is effective.
	External queries: Many Subqueries: Many	Hash execution is effective. (Performance improvement cannot be executed because the number of data items is high.)
	External queries: Intermediate Subqueries: Many	
	External queries: Few Subqueries: Many	Hash execution or work table execution is effective. If the predicate is converted to an EXISTS predicate that contains an external reference, HiRDB may be able to conduct the search rapidly.
Table subqueries specified on the right side of quantified predicates (except =ANY and =SOME) and the IN predicate	Work table execution is always applied.	
Subqueries of the EXISTS predicate	Row value execution is always applied.	
Other subqueries (scalar subqueries and row subqueries)		

(2) Processing methods

(a) Work table ATS execution

`WORK TABLE ATS SUBQ`

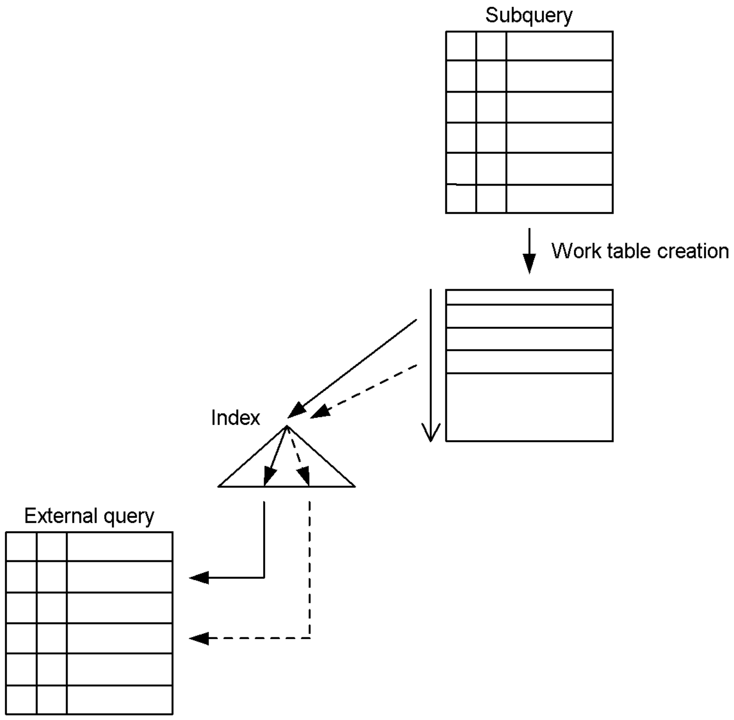
This processing method applies to table subqueries specified on the right side of `=ANY` and `=SOME` quantified predicates and `IN` predicates.

First, HiRDB calculates the values of the subquery selection expression and creates a work table. Next, HiRDB uses an index to retrieve external queries. To retrieve the queries, HiRDB uses the subquery results to narrow the index search range. The query search conditions are `ATS` and `RANGES`.

In some cases, HiRDB executes duplicate elimination (`DISTINCT`) internally for subqueries.

The following figure shows the `WORK TABLE ATS SUBQ` processing method.

Figure 4-47: WORK TABLE ATS SUBQ processing method



An example of a quantified predicate and a comparison predicate is shown as follows.

Example

```
SELECT C1 FROM T1 WHERE C2=ANY(SELECT C2 FROM T2)
```

Note

This example supposes that an index is defined in `T1` (`C2`).

First, table `T2` of the subquery is searched, and a work table is created from the values of `T2.C2`. Next, the values of `T2.C2` are fetched one row at a time from the work table, and a search is conducted by narrowing the search range of the index defined in `T1.C2` of the external query.

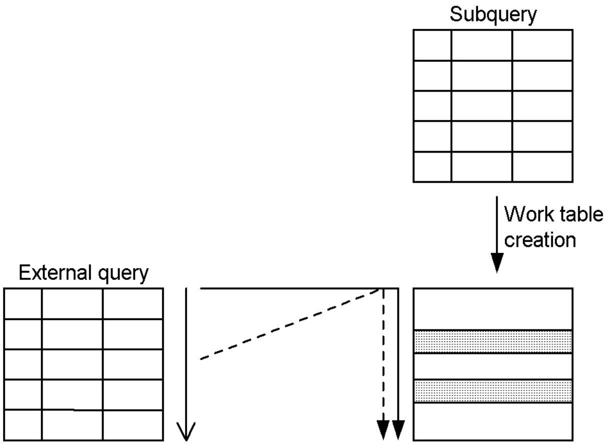
(b) Work table execution

`WORK TABLE SUBQ`

This processing method is applied to table subqueries specified on the right side of quantified predicates and `IN` predicates. First, the values of the subquery selection expression are determined and a work table is created. Next, the outer query is searched. Each time a row of the outer query is searched, the row is matched with the results of the subquery, and the search conditions are evaluated.

The following figure shows the `WORK TABLE SUBQ` processing method.

Figure 4-48: WORK TABLE SUBQ processing method



Example

```
SELECT T1.C1 FROM T1 WHERE T1.C2=ANY(SELECT C2 FROM T2)
```

First, table `T2` of the subquery is searched, and a work table is created from the values of `T2.C2`. Next, the outer query is executed, the rows are fetched one at a time, the `T1.C2` values are matched with the work table that was created from the subquery, and the search conditions are evaluated.

(c) Row value execution

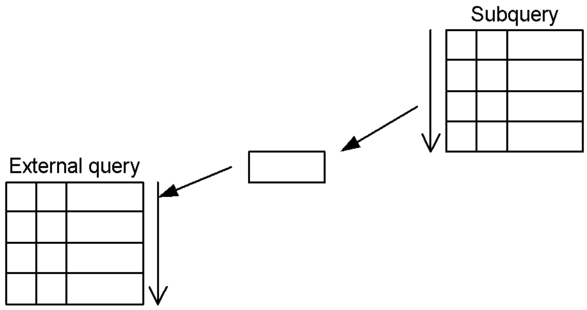
`ROW VALUE SUBQ`

This processing method is applied to row subqueries, scalar subqueries, and EXISTS predicates. With this method, first the value of the selection expression in the subquery is determined. Then, the value of the subquery result is used in evaluating the conditions, including the subquery of the outside query.

With comparison predicates, if HiRDB judges that using an index is better when searching an external query, it uses an index in the search.

The following figure shows the ROW VALUE SUBQ processing method.

Figure 4-49: ROW VALUE SUBQ processing method



An example is shown as follows.

Example

```
SELECT T1.C1 FROM T1 WHERE T1.C2<(SELECT MAX(C2) FROM T2)
```

First, table T2 of the subquery is searched, and the MAX(T2.C2) values are fetched. (A work table is not created.) Next, the condition that includes the subquery in the external query is evaluated with the MAX(T2.C2) values.

(d) Hash execution

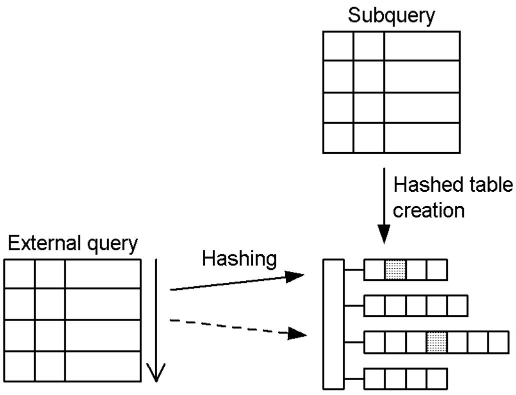
`HASH SUBQ`

This processing method is applied to table subqueries specified on the right side of quantified predicates and IN predicates.

First, the values of the subquery selection expression are determined, and a hash table is created from the selection expression values. Next, the external query is executed, hashed with the column values specified on the left side of the quantified predicate and IN predicate, matched with the hash table that was created from the subquery, and searched.

The following figure shows the HASH SUBQ processing method.

Figure 4-50: HASH SUBQ processing method



An example is shown as follows.

Example

```
SELECT T1.C1 FROM T1 WHERE T1.C2=ANY(SELECT C2 FROM T2)
```

First, table `T2` of the subquery is searched, and a hash table is created from the `T2.C2` values. Next, the external query is executed, hashed with the `T1.C2` values, matched with the hash table that was created from the subquery, and searched.

4.5.9 Execution of subqueries with external references

(1) Execution method types

The following table shows the execution methods and features of subqueries that have external references.

Table 4-15: Execution methods and features of subqueries with external references

Execution method	Processing method	Advantages	Disadvantages
Nested loops work table execution	Each time a row of the external query is searched, this method executes the subquery, creates a work table, and evaluates the condition that includes the subquery.	An index can be used for the subquery search conditions that include a reference column to the outside. Therefore, data can be searched rapidly when a subquery search condition can narrow the search range by using an index. In external query searches, the subquery search can be omitted when the external reference column repeatedly searches a row of the same value.	The performance drops when the number of external query hits is high.
Nested loops row value execution	Each time a row of the external query is searched, this method executes the subquery (a work table is not created) and evaluates the condition that includes the subquery.	An index can be used for the subquery search conditions that include a reference column to the outside. Therefore, data can be searched rapidly when a subquery search condition can narrow the search range by using an index. In external query searches, the subquery search can be omitted when the external reference column repeatedly searches a row of the same value.	The performance drops when the number of external query hits is high.

Execution method	Processing method	Advantages	Disadvantages
Hash execution	This method creates a hash table from the subquery results beforehand. Then each time a row is fetched from the external query, this method hashes the values of the external query and matches them with the hash table.	Data can be searched rapidly when the number of subquery hits, excluding conditions that include external reference columns, is low and the number of external queries is high.	An index cannot be used for conditions that include an external reference column. If the hit count for subqueries that exclude conditions that include an external reference column is high, the size of the work table buffer used becomes large. Although the work table buffer size to be used can be specified, the buffer data must be saved to a work table file when the work table buffer becomes full, and consequently the performance drops. If subqueries are to be joined, conditions that include external reference columns are evaluated after the subqueries are joined.

(2) Processing methods

(a) Nested loops work table execution

NESTED LOOPS WORK TABLE SUBQ

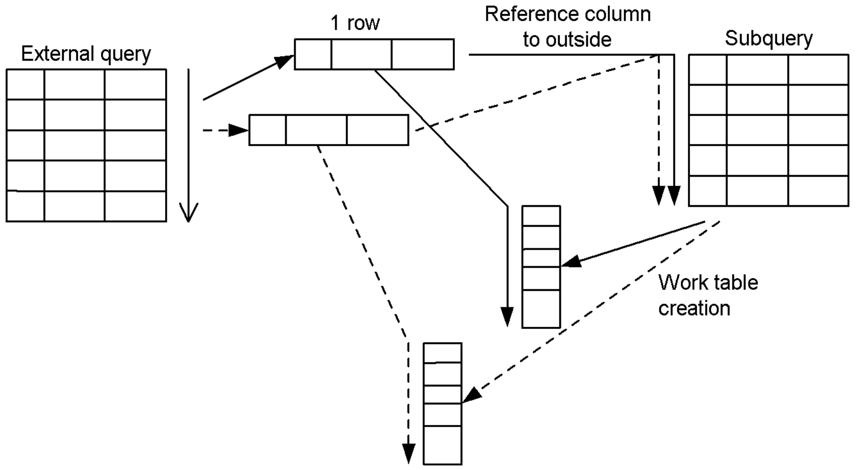
This processing method is applied to table subqueries specified on the right side of quantified predicates and IN predicates.

First, the external query is executed. During the execution, each time a row of the external query is fetched, the values in the external reference column are used to execute the subquery, the values of the subquery selection expression are calculated, and a work table is created. Next, the work table that was created from the subquery is used to evaluate the condition that includes the external subquery.

Because the external query is processed one row at a time, multiple work table areas are never created at the same time. Also, because the subquery is executed for each row in the external query, the performance drops when the external query has many rows.

The following figure shows the NESTED LOOPS WORK TABLE SUBQ processing method.

Figure 4-51: NESTED LOOPS WORK TABLE SUBQ processing method



Example

```
SELECT C1 FROM T1
  WHERE C1=ANY(SELECT C1 FROM T2 WHERE C2=T1.C2)
```

Note

The underlined section is the external reference column.

The external query is executed. The values of the outer reference column (`T1.C2`) are used to execute the subquery for all rows of the external query, and a work table is created from the `T2.C1` values. Next, `T1.C1` is matched with the `T2.C1` work table, and the condition that includes the subquery is evaluated.

(b) Nested loops row value execution

`NESTED LOOPS ROW VALUE SUBQ`

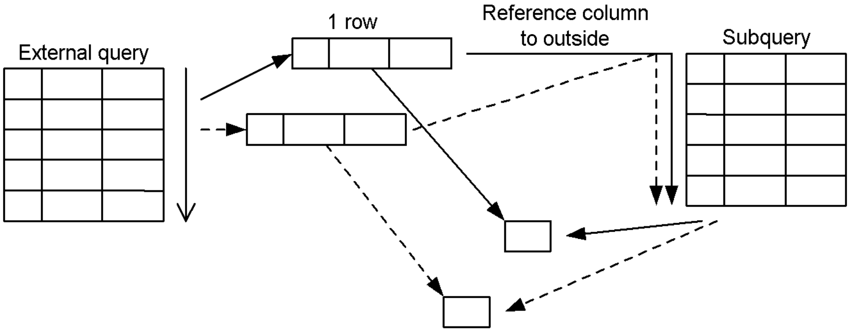
This processing method is applied to row subqueries, scalar subqueries, and `EXISTS` predicates.

First, the external query is executed. During the execution, each time a row of the external query is fetched, the values in the external reference column are used to execute the subquery, and the values of the subquery selection expression are calculated. (A work table is not created.) Next, the values of the subquery results are used to evaluate the condition that includes the subquery of the external query.

Because the subquery is executed for each row in the external query, the performance drops when the external query has many rows.

The following figure shows the `NESTED LOOPS ROW VALUE SUBQ` processing method.

Figure 4-52: NESTED LOOPS ROW VALUE SUBQ processing method



Example

```
SELECT C1 FROM T1
  WHERE C1=(SELECT MAX(C1) FROM T2 WHERE C2=T1.C2)
```

Note

The underlined section is the external reference column.

The external query is executed. The values of the outer reference column (`T1.C2`) are used to search the subquery for all rows of the external query, and the `MAX(T2.C1)` value is fetched. (A work table is not created.) Next, the condition that includes the subquery found in the external query is evaluated.

(c) Hash execution

`HASH SUBQ`

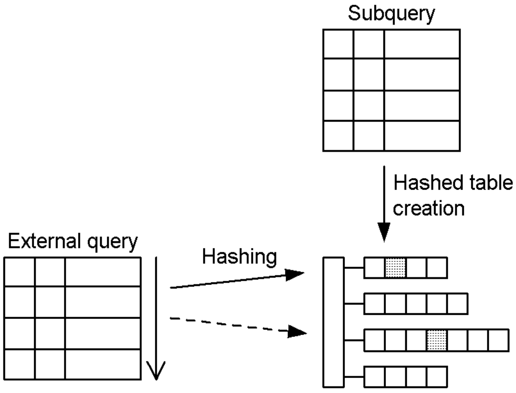
This processing method applies to table subqueries specified in `EXISTS` predicates and on the right side of comparison predicates, quantified predicates, and `IN` predicates.

First, the subquery is executed without the condition that includes the external reference column, and the values of the query selection expression are determined. At this time, the columns that were narrowed by the external reference column from the search condition compared with `=` in the subquery are used to create a hash table. (If the predicate is `=ANY`, `=SOME` or `IN`, the selection expression is used to create the hash table.)

Next, an external query is executed, each fetched row is hashed with the value of the external reference column, matched with the hash table that was created from the subquery, and searched. (If the predicate is `=ANY`, `=SOME`, or `IN`, the columns values specified on the left side of the predicate are also used for hashing.)

The following figure shows the `HASH SUBQ` processing method.

Figure 4-53: HASH SUBQ processing method



Examples of an `EXISTS` predicate and a comparison predicate are shown as follows.

Example 1: EXISTS predicate

```
SELECT T1.C1 FROM T1
  WHERE EXISTS(SELECT * FROM T2 WHERE C1='a' AND C2=T1.C2)
```

Note

The underlined section is the external reference column.

First, the subquery is evaluated without the condition that includes the external reference column, and a hash table is created from the subquery column (`T2.C2`) that has been narrowed by using the external reference column. Next, the external query is executed, hashed with the values of the external reference column (`T1.C2`), and matched with the hash table that was created from the subquery. Then the `EXISTS` predicate is evaluated.

Example 2: Comparison predicate

```
SELECT T1.C1 FROM T1
  WHERE T1.C3<(SELECT T2.C3 FROM T2 WHERE C1='a' AND
C2=T1.C2)
```

Note

The underlined section is the external reference column.

First, the subquery is evaluated without the condition that includes the external reference column, and a hash table is created from the subquery column (`T2.C2`) that has been narrowed by using the external reference column. Next, the external query is executed, hashed with the values of the external reference column (`T1.C2`), and matched with the hash table created from the subquery. Then the condition that includes the external reference column is evaluated. If the result is true, the comparison predicate (`<`) is evaluated.

4.5.10 Preparing for application of hash join and subquery hash execution

This section describes the items that must be set before hash join or hash execution of a subquery can be applied with the SQL extension optimizing option.

(1) Items to be preset

Before hash join or hash execution of a subquery can be applied, the following items must be set:

- **Hash table size**
- **Method for allocating the work table buffer**
- **Work table buffer size**
- **Hashing mode**

(a) Hash table size

Use the `pd_hash_table_size` operand in the system definition or `PDHASHTBLSIZE` in the client environment definition to set the hash table size. Calculate the maximum hash table row length, and then set the hash table size to a value that is equal to or greater than the value obtained from the following formula:

The hash table size must be smaller than the value in the `pd_work_buff_size` or `pd_work_buff_expand_limit` operand in the system definition. If the hash table size is equal to or greater than the value of these operands, an error will occur during hash join or subquery hash execution.

$$\textit{hash-table-size}\ \text{(in kilobytes)} \geq \uparrow(\textit{maximum-hash-table-row-length}\ \text{(in bytes)} \times 2 + 32) \div 128 \uparrow \times 128$$

maximum-hash-table-row-length

For each `SELECT` statement, calculate the hash table row lengths for the following units. Then select the largest calculated value (maximum hash table row length).

- Query specifications that join multiple tables with =
- Subqueries that correspond to one of the following:
  - Table subquery specified on the right side of an `=ANY` quantified predicate
  - Table subquery specified on the right side of an `=SOME` quantified predicate

- Table subquery specified on the right side of an `IN` predicate
- Other subquery that specifies an external reference column with `=` in a search condition

The calculation methods for hash table row length are shown as follows.

Query specification that joins multiple tables with `=`

1. For columns specified in the selection expressions and search conditions of the tables that are linked with `=`, calculate the row length in each table from the following formula:

$$\text{Row length of each table}= \sum_{i=1}^{n} (ai) + 2 \times n$$

n: Number of columns in selection expressions and search conditions for target table

2. From the table row lengths that were calculated in 1, use one that is not the smallest value and calculate the hash table length from the following formula:

$$\begin{array}{l}\text{Hashed table row length} = \\ \text{(Unit: bytes)}\end{array} \left\uparrow \frac{\displaystyle\sum_{j=1}^{m-1} \text{Row length j of each table}}{4} \right\uparrow \text{X } 4 + 6$$

m: Number of tables joined with = in `FROM` clause

Subquery

For columns specified in subquery selection expressions and columns specified in predicates that include an external reference column in the search condition, calculate the hash table row length from the following formula:

$$\begin{array}{l}\text{Hashed table row length} = \\ \text{(Unit: bytes)}\end{array} \left\uparrow \frac{\displaystyle\sum_{i=1}^{n} (ai)+2 \text{ x } n}{4} \right\uparrow \text{x } 4 + 6$$

ai

Data length of the *i*-th data item. For details about data length, see the *HiRDB Version 9 Installation and Design Guide*. However, for character data (including national character data and mixed character data) that is specified only in a selection expression of a table joined with = and has a defined length of 256 bytes or more, the data length becomes 12.

Hash tables of the size calculated previously can store 1,500 to 2,000 rows. If the number of inner tables to be joined or the number of subquery searches is high, bucket partitioning is executed several times, and the performance may not improve. In this case, either calculate and set the hash table size for batch hash join shown as follows or see (2) to tune the hash table size.

hash-table-size-for-batch-hash-join (in kilobytes) = ↑(*number-of-hash-table-data-pages* + *number-of-hash-table-management-table-pages*) ÷ *number-of-one-segment-pages*↑ x 128

number-of-hash-table-data-pages = ↑*number-of-hash-table-rows* ÷ MIN{↓(*hash-table-page-length* 48) ÷ *hash-table-row-length*↓, 255}↑ + 63

number-of-hash-table-management-table-pages = ↑(16 x *number-of-hash-table-rows* + (↑(*number-of-hash-table-data-pages* x *hash-table-page-length* + 16 x *number-of-hash-table-rows*) ÷ (*number-of-one-segment-pages* x *hash-table-page-length*)↑ x 8) + 8) ÷ *hash-table-page-length*↑ x *hash-table-page-length*

number-of-one-segment-pages = ↓(128 x 1024) ÷ *hash-table-page-length*↓

Determine the hash table page length from the hash table row length as shown in the following table.

Hash table row length	Hash table page length
0 to 1,012	4,096
1,013 to 2,036	8,192
2,037 to 4,084	16,384
4,085 to 16,360	32,768

Hash table row length	Hash table page length
16,361 to 32,720	↑(*hash-table-row-length* + 48) ÷ 2048↑ x 2048 *hash-table-row-length*: Number of inner tables to be joined when the targets of batch hash join are joined. If the targets are subqueries, this value is the number of subquery searches excluding the predicates that include outer reference columns in the search conditions.

(b) Method for allocating the work table buffer

The method for allocating the work table buffer must be set to buffer batch allocation (`pool`) in server process units. Therefore specify `pool` in the `pd_work_buff_mode` operand of the system definition.

(c) Work table buffer size

Hash tables are allocated in the work table buffer. If the work table buffer size or the upper limit size for expansion allocation of the work table buffer is smaller than the specified hash table size, an error occurs because of insufficient space in the work table buffer. Therefore, in the `pd_work_buff_size` or `pd_work_buff_expand_limit` operand of the system definition, set a value that is equal to or larger than the value calculated with the following formula:

work-table-buffer-size (in kilobytes) ≥ (*hash-table-size* (in kilobytes) x 2 + 256) x *maximum-number-of-hash-joins-in-*`SELECT`*-statement* + 128

maximum-number-of-hash-joins-in-`SELECT`*-statement*

Calculate the number of hash joins in each `SELECT` statement from the following formula, and set the largest value as the maximum number of hash joins in a `SELECT` statement. The number of hash joins is determined by counting the items that have `HASH JOIN` as the join type in the join processing information that is output by the access path display utility (`pdvwopt`).

number-of hash-joins-in-`SELECT`*-statement* =
((*number-of-tables-joined-with-*=)
(*number-of-query-specifications-joined-with-*=)), +
(*number-of-*=`ANY`*-quantified-predicates*) +
(*number-of-*=`SOME`*-quantified-predicates*) +
(*number-of-*`IN`*-(subquery)-specifications*) +
(*number-of-other-subqueries-that-specify-external-reference-columns-with-*=*-in -search-conditions*)

If multiple cursors are to be opened at the same time and searched, total the values that are calculated for the individual cursors.

Example

```
SELECT A.A1,B.B2,C.C3 FROM A,B,C                    3-1
```

```
WHERE A.A1=B.B1 AND A.A2=B.B2
  AND B.B3=C.C3
  AND A.A1=C.C1
  AND A.A4=ANY(SELECT D.D4 FROM D)               1
  AND A.A5=SOME(SELECT E.E5 FROM E)              1
  AND A.A6 IN(SELECT F.F6 FROM F
           WHERE F.F1=A.A1)                      1
  AND EXISTS(SELECT G.G1 FROM G WHERE G.G1=B.B1) 1
```

In this example, the values are (3-1) + 1 + 1 + 1 + 1, so the number of hash joins in this SELECT statement is 6.

Adding about 4,096 extra kilobytes to the value calculated from the work table buffer size formula shown previously increases the input/output unit size during bucket partitioning, which in turn improves the performance.

If batch hash join without bucket partitioning is to be executed on all data, the operation can be executed if the following formula is satisfied:

work-table-buffer-size (in kilobytes) ≥ *hash-table-size* (in kilobytes) x *maximum-number-of-hash-joins-in-*SELECT*-statement* + 384

(d) Hashing mode

A retrieval that accompanies a hash join or subquery execution is processed by hashing.

You can select the hashing mode with the pd_hashjoin_hashing_mode operand of the system definition or with PDHJHASHINGMODE client environment definition. The default is TYPE1.

TYPE1 is the hashing mode of HiRDB versions before 07-02. If you use hash join for the first time in HiRDB version 07-02 or a more recent version, specify TYPE2.

If you specify TYPE1 in HiRDB version 07-02 or a more recent version, you may not obtain the desired performance. If this happens, specify TYPE2 as the hashing mode, or see *(3) Tuning the hashing mode*, and tune the mode.

(2) Tuning methods for hash table size

(a) Tuning information used

The hash table size can be tuned based on either of the following types of tuning information:

- UAP statistical report (specify client environment definition PDUAPREPLVL)
- UAP statistical information from the statistics analysis utility

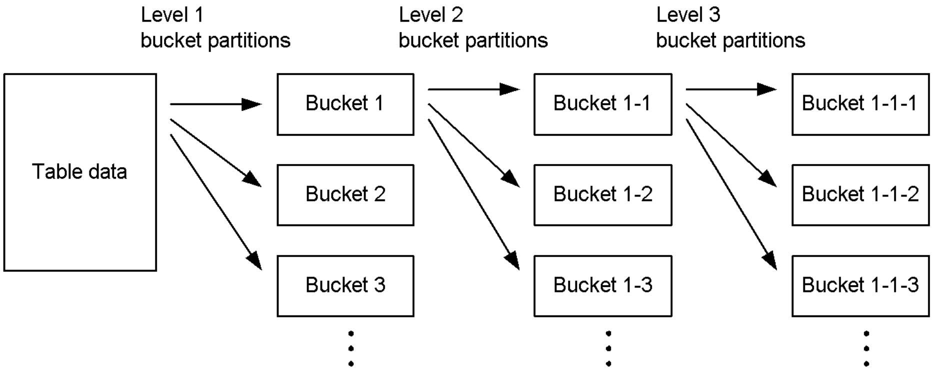
For details about the UAP statistical report, see *11.1.4 UAP statistical report facility*. For details about the statistics analysis utility, see the *HiRDB Version 9 Command Reference* manual.

(b) Items derived from tuning information

When tuning information about the hash table size is obtained, the following items can be determined:

- Whether batch hash join, which expands data into a hash file all at once, or bucket partitioning hash join, which expands data into a hash table in bucket units, is set
- Whether bucket repartitioning is being executed when bucket partitioning hash join is set
- How large the hash table size should be set to execute batch hash join when bucket partitioning hash join is set
- How large the hash table size should be set to avoid bucket repartitioning when bucket partitioning hash join is set

Bucket repartitioning refers to the repetition of bucket partitioning recursively for up to three levels when the bucket size exceeds the hash table size. An example is shown as follows.



The number of partitions in one bucket partitioning is determined from the following formula:

$\textit{number-of-bucket-partitions} = \mathrm{MIN}\{\downarrow(\textit{hash-table-size} \div 2) \div \textit{hash-table-page-length}\downarrow, 64\}$

Even with batch hash join, level 1 bucket partitioning is executed for inner tables to be joined.

(c) Tuning methods

The following table describes the tuning methods for hash table size.

Table 4-16: Tuning methods for hash table size

Tuning information (unit: kilobytes)	Tuning method
Maximum batch hash table size	If a value that is equal to or greater than this value was set as the hash table size, batch hash join without bucket partitioning is set for all data.[#1] If this value exceeds the maximum hash table size that can be specified, batch hatch join cannot be used. If this value is 0, hash join or subquery hash execution has not been performed.
Level 1 maximum bucket size	If a value that is equal to or greater than this value was set as the hash table size, bucket partitioning has terminated at level 1. If bucket partitioning is set to level 2 or higher, specifying this value as the hash table size terminates bucket partitioning at level 1.[#2] If batch hash join without bucket partitioning was executed on all data, 0 is displayed for this value.
Level 2 maximum bucket size	If a value that is equal to or greater than this value was set as the hash table size, bucket partitioning has terminated at level 2. If bucket partitioning is set to level 3 or higher, specifying this value as the hash table size terminates bucket partitioning at level 2.[#2] If level 2 bucket partitioning was not executed even once, 0 is displayed for this value.
Level 3 maximum bucket size	If a value that is equal to or greater than this value was set as the hash table size, bucket partitioning has terminated at level 3. If the hash table size is smaller than this value, each bucket is partially expanded into the hash table, and the processing efficiency becomes worse. In this case, set the hash table size to this value or larger.[#2] In some cases, not applying hash join or subquery hash execution improves the performance. If level 3 bucket partitioning was not executed even once, 0 is displayed for this value.

#1: When the hash table size is increased, the number of bucket partitions at each partitioning execution may increase according to the formula for calculating the number of bucket partitions. Consequently, a hash table size that is larger than the size that was used during tuning information acquisition may be necessary.

If you used tuning information and increased the hash table size, get the tuning information again. If the intended results are not realized, increase the hash table size again according the new tuning information that was acquired.

#2: When the hash table size is increased, the number of bucket partitions at each partitioning execution may increase according to the formula for calculating the number of bucket partitions. Consequently, bucket partitioning may terminate at the intended level even if the hash table size is smaller than the size that was used during tuning information acquisition.

On the other hand, when the hash table size is decreased, the number of bucket partitions at each partitioning execution may decrease. Consequently, bucket partitioning may not terminate at the same level used during tuning information acquisition. You should therefore use this tuning information when the hash table size is increased.

(3) Tuning the hashing mode

(a) Tuning information used

The hash table size can be tuned based on either of the following types of tuning information:

- UAP statistical report (specify client environment definition `PDUAPREPLVL`)
- UAP statistical information from the statistics analysis utility

For details about the UAP statistical report, see *11.1.4 UAP statistical report facility*. For details about the statistics analysis utility, see the manual *HiRDB Version 9 Command Reference*.

(b) Item derived from tuning information

By obtaining tuning information about the hash table size, you can determine the following item:

- Retrieval performance of the specified hashing mode

(c) Tuning mode

The following table shows tuning information for the hashing mode.

Table 4-17: Tuning information for the hashing mode

Tuning information (unit: number)	Description
Maximum number of comparisons	Maximum number of comparisons during hash searching
Total number of comparisons	Total number of comparisons during hash searching
Number of searches	Number of hash table searches
Average number of comparisons	Total number of comparisons / number of searches

Note

If the tuning information value is 0, hash join or subquery hash execution has not been performed.

Tuning method example

Set `TYPE1` and `TYPE2` in client environment definition `PDHJHASHINGMODE`, and get the average number of comparisons from the statistical information for each mode. Compare the average number of comparisons, and set the hashing mode with the smaller value to the pd_hashjoin_hashing_mode operand.

4.5.11 Deriving high-speed search conditions

A *high-speed search condition* refers to a condition derived from a WHERE clause search condition or an ON search condition in a FROM clause by CNF conversion or condition shifting. When high-speed search conditions are derived, the retrieval performance improves because the rows to be retrieved can be narrowed at an earlier stage.

When HiRDB derives high-speed search conditions, it retains the original search conditions used in the derivation. HiRDB can therefore generate just those derived conditions that are optimal conditions without generating derived conditions that cannot be used to narrow the search.

When HiRDB derives high-speed search conditions, it optimizes the search by considering the new derived conditions when it determines the access path (including the table retrieval method, join method, and join sequence). Therefore, when HiRDB derives high-speed search conditions, the access paths may change as described as follows:

- HiRDB determines that the rows to be retrieved can be narrowed down at an early stage, and retrievals with an index become easier to select.
- If OR is specified in a join condition, and the CNF conversion and OR reduction operations extract the join condition outside OR, then nested-loops-join, merge join, and hash join can be applied outside direct products.
- If a limiting condition is specified for only one of the tables to be joined, nested-loops join becomes easier to select. If limiting conditions are specified for both tables, merge join and hash join become easier to select.

When high-speed search conditions are derived from complex conditions, it takes longer to generate the high-speed search conditions and to evaluate the conditions when the search is executed. Therefore, depending on the SQL statements involved, the performance may actually drop instead of improve.

(1) Application scope of high-speed search conditions

Whether or not HiRDB derives high-speed search conditions depends on the specification values of the SQL optimization and SQL extension optimizing options. The following table shows the relationships between the SQL optimization and SQL extension optimizing options and deriving high-speed search conditions.

Table 4-18: Relationships between the SQL optimization and SQL extension optimizing options and deriving high-speed search conditions

Type	Derivation source condition	Derived condition	Specified SQL optimization option or SQL extension optimizing option	
			Do not derive	Derive[#1]
CNF conversion	OR condition for one table	One-table condition	--	--
	OR condition extending across two or more tables	One-table condition	G[#2]	G
		Join condition (*column*=*column*) is extracted by OR reduction	G[#2]	G
		Other condition for two or more tables	--	--
Condition shifting	Join condition for tables A and B, and condition for table A	One-table condition for table B	--	G
	Join condition for tables A and B, and join condition for tables A and C	Join condition for tables B and C	--	--

Legend:

G: HiRDB generates high-speed search conditions.

--: HiRDB does not generate high-speed search conditions.

#1: Specify the value for deriving high-speed search conditions in the SQL optimization options.

#2: If a direct product is specified in the search at the derivation source of high-speed search conditions, high-speed search conditions are generated depending on the conditions described below. The following table describes whether high-speed search conditions are derived, and the derivation condition.

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
Two-table search	Direct product	Join condition (*column*=*column*) cannot be extracted by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	--
				Other two-table condition	N

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
		Join condition (*column*=*column*) can be extracted by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	Y
				Other two-table condition	N
	No direct product	--			N

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
Search in three or more tables	All direct product	Join condition (*column*=*column*) cannot be extracted by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	--
				Other two-table condition	N
		Join condition (*column*=*column*) can be extracted partially by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	Y
				Other two-table condition	N
		Join condition (*column*=*column*) can be extracted by OR reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
			--	Join condition (*column*=*column*)	Y
				Other two-table condition	N
	Partially direct product	Join condition (*column*=*column*) cannot be extracted by OR reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	--
				Other two-table condition	N

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
		Join condition (*column*=*column*) can be extracted partially by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the one-table condition to be derived.	None	N
			The search condition of the derivation source does not include a one-table condition for the same table as for the one-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	Y
				Other two-table condition	N
		Join condition (*column*=*column*) can be extracted by `OR` reduction.	The search condition of the derivation source includes a one-table condition for the same table as for the two-table condition to be derived.	None	N

Search at the derivation source		Derivation condition		High-speed search conditions that are derived	Whether derived or not
			The search condition of the derivation source does not include a one-table condition for the same table as for the two-table condition to be derived.	One-table condition	Y
			--	Join condition (*column*=*column*)	Y
				Other two-table condition	N
	No direct product	--			N

Legend:

Y: Derived

N: Not derived

--: Not applicable

(2) Deriving high-speed search conditions by CNF conversion

CNF conversion refers to converting conditions joined with `OR` (disjunctive normal form (DNF) format) into equivalent conditions joined with `AND` (conjunctive normal form (CNF) format). High-speed search conditions can be derived by applying CNF conversion to `WHERE` clause search conditions or to `ON` search conditions in `FROM` clauses.

(a) Search conditions derived by CNF conversion

The following search conditions are derived by CNF conversion:

- If CNF conversion can generate a one-table condition from a condition that extends across two or more tables joined with `OR`, HiRDB derives the one-table condition as a high-speed search condition. By deriving the one-table condition, HiRDB can narrow the number of items to be joined.
- If all conditions joined with `OR` are included in the same join condition

(*column*=*column* only) for two tables, HiRDB can derive (*join-condition* OR ... OR *join-condition*) by applying CNF conversion to that condition. Then if the same condition can be used to execute duplicate elimination on all join conditions joined with OR (OR reduction), HiRDB derives the join conditions as high-speed search conditions. By deriving the join conditions, HiRDB can eliminate direct product processing and improve the performance.

Furthermore, the specifications for the SQL optimization options and SQL extension optimizing options determine whether HiRDB derives high-speed search conditions. For details about the relationships between SQL optimization options and SQL extension optimizing options, and deriving of high-speed search conditions, see *(1) Application scope of high-speed search conditions*.

(b) Conditions when CNF conversion is not executed

High-speed search conditions are not derived by CNF conversion if any one of the following conditions applies:

- A derived search condition would include a subquery.
- Deriving the derivation-source conditions specified in the ON search condition of an outer join would produce conditions that are confined to the outer table.
- Deriving the derivation-source conditions specified in the WHERE clause of an outer join would produce conditions that are confined to the inner table.
- Deriving the derivation-source conditions specified in the WHERE clause of an outer join would produce conditions for two or more tables.
- If HiRDB derived high-speed search conditions, the maximum nest count of the Boolean operations would exceed 255.
- A search condition is specified in a HAVING clause.
- A derived search condition would be a join condition resulting from direct product.

(3) Deriving high-speed search conditions by condition shifting

Condition shifting refers to deriving a new condition from two or more conditions.

The methods of deriving search conditions by condition shifting are described as follows.

- Deriving high-speed search conditions by shifting a one-table condition through a join condition
- Deriving high-speed search conditions by shifting join conditions (applicable to the UNIX edition only)

Furthermore, the specifications for the SQL optimization options and SQL extension optimizing options determine whether HiRDB derives high-speed search conditions.

For details about the relationships between SQL optimization options and SQL extension optimizing options, and deriving of high-speed search conditions, see *(1) Application scope of high-speed search conditions*.

(a) Deriving high-speed search conditions by shifting a one-table condition through a join condition

If the search conditions consist of a two-table join condition (*column=column* only) and a one-table condition that includes the join column, HiRDB derives a one-table condition for the column in the table to be joined. An example is shown as follows:

```
T1.C1 = T2.C1 AND T1.C1 > 10
 → T1.C1 = T2.C1 AND T1.C1 > 10 AND T2.C1 > 10
```

The underlined section becomes the derived high-speed search condition.

- One-table conditions that are targets for condition shifting

  The one-table conditions that are targets for condition shifting are listed as follows:

  - *column-specification comparison operator* {*value-specification* | *reference-column-to-outside*}

    Condition shifting is executed even when the left and right sides of the comparison operator (`=`, `<>`, `^=`, `!=`, `<`, `<=`, `>`, or `>=`) are switched.

  - *column-specification* `IS [NOT] NULL`
  - *column-specification* `[NOT] IN` `(`*value-specification* `[,` *value-specification*`]...)`
  - *column-specification* `[NOT] LIKE` *pattern-character-string* `[ESCAPE` *escape-character*`]`

    If the join columns have different data lengths, condition shifting is executed only when the pattern character string is a literal and forward matching is applied.

  - *column-specification* `[NOT] XLIKE` *pattern-character-string* `[ESCAPE` *escape-character*`]`

    If the join columns have different data lengths, condition shifting is not executed.

  - *column-specification* `BETWEEN` {*value-specification* | *reference-column-to-outside*} `AND` {*value-specification* | *reference-column-to-outside*}

- *column-specification* `[NOT] SIMILAR TO` *pattern-character-string* `[ESCAPE` *escape-character*`]`

  If the join columns have different data lengths, condition shifting is executed only when the pattern character string is a literal and forward matching that produces a `LIKE`-predicate equivalent is applied.

■ Conditions when condition shifting is not executed

Condition shifting is not executed if any one of the following conditions applies:

- The join condition is an outer join.
- The join condition is an inner join, and condition shifting would take place between a `WHERE` clause search condition and an `ON` search condition in a `FROM` clause. (For an inner join involving three or more tables, HiRDB executes condition shifting between multiple `ON` search conditions.)
- The data type of the join columns is a comparison of fixed length and variable length.
- The data type of the join columns is `FLOAT` or `SMALLFLT`.
- If HiRDB derived high-speed search conditions, the maximum nest count of the Boolean operations would exceed 255.
- A search condition is specified in a `HAVING` clause.

(b) Deriving high-speed search conditions by shifting join conditions (applicable to the UNIX edition only)

If the search conditions consist of a two-table join condition (*column*=*column* only) and a join condition (*column*=*column* only) between a column in one of the two tables and a column in a separate third table, HiRDB derives a new join condition from the remaining two columns that are not linked by a join condition. If correlation names are specified, the tables are viewed as separate tables if the correlation names are different. An example is shown as follows:

```
T1.C1 = T2.C1 AND T2.C1 = T3.C1
 → T1.C1 = T2.C1 AND T2.C1 = T3.C1 AND T1.C1 = T3.C1
```

The underlined section is the rapid search condition that was derived.

■ Conditions when the join condition is not shifted

The join condition is not shifted if any one of the following conditions applies:

- The join condition is an outer join.
- The join condition is an inner join, and condition shifting would take place between a `WHERE` clause search condition and an `ON` search condition in a

FROM clause. (For an inner join involving three or more tables, HiRDB executes condition shifting between multiple ON search conditions.)

- The data type of the join columns is a comparison of fixed length and variable length.
- The data type of the join columns is FLOAT or SMALLFLT.
- If HiRDB derived high-speed search conditions, the maximum nest count of the Boolean operations would exceed 255.
- A search condition is specified in a HAVING clause.

4.6 Data guarantee levels

The *data guarantee level* specifies the transaction point up to which the retrieved data is to be guaranteed. The data guarantee levels range from 0 to 2. Specify the data guarantee level according to operation goals, for example, whether you want to prevent other users from updating data or whether you want to allow other users to reference data being updated.

4.6.1 Specifying the data guarantee level

A data guarantee level can be specified for each UAP. To specify a data guarantee level for each SQL statement, specify a lock option for that SQL statement.

If both a data guarantee level and a lock option are specified concurrently, the lock option specification becomes valid. The following table shows the lock option for each data guarantee level.

Table 4-19: Relationship between data guarantee level and lock option

Data guarantee level	Lock option
0	`WITHOUT LOCK NOWAIT`[#1, #3]
1	`WITHOUT LOCK WAIT`[#3]
2	`WITH SHARE LOCK` or `EXCLUSIVE LOCK`[#2]

#1: When the `FOR UPDATE` clause is specified in a cursor declaration or a dynamic `SELECT` statement, a data guarantee level of `1` is assumed even if `0` is specified.

#2: When the `FOR UPDATE` clause is specified in a cursor declaration or a dynamic `SELECT` statement, `WITH EXCLUSIVE LOCK` is assumed.

#3: `WITH EXCLUSIVE LOCK` is assumed in the following cases:

- `YES` is specified in client environment definition `PDFORUPDATEEXLOCK` when a cursor declaration or dynamic `SELECT` statement in which the `FOR UPDATE` clause is specified is executed.
- `FOR UPDATE EXLOCK` is specified immediately after the data guarantee level in the SQL compile options that are specified when the routine is defined.

The data guarantee level can be specified at the following locations:

- `PDISLLVL` in the client environment definition
- SQL compile option in `ALTER PROCEDURE`

- SQL compile option in ALTER ROUTINE
- SQL compile option in ALTER TRIGGER
- SQL compile option in CREATE PROCEDURE
- SQL compile option in CREATE TRIGGER
- SQL compile option in procedure body of CREATE TYPE

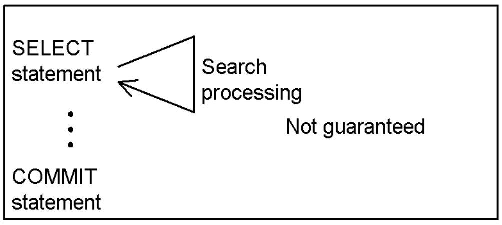
4.6.2 Data guarantee level types

(1) Data guarantee level 0

Specify data guarantee level 0 to allow other users to view data being updated without waiting for update completion. This guarantee level can improve the concurrent execution capability more than the other guarantee levels. However, if the same rows are searched twice in the same transaction, the first and second search results may not be the same.

The following figure shows the data guarantee range of data guarantee level 0.

Figure 4-54: Data guarantee range of data guarantee level 0

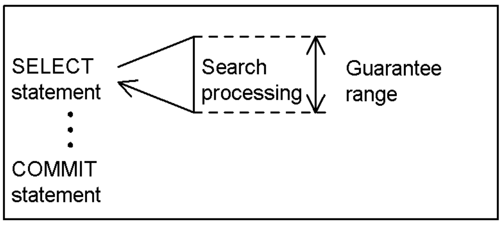


(2) Data guarantee level 1

Specify data guarantee level 1 to prevent other users from updating data that has been searched once until the search processing is completed (until HiRDB finishes viewing the pages or rows). This guarantee level therefore improves the concurrent execution capability. However, if the same rows are searched twice in the same transaction, the first and second search results may not be the same.

The following figure shows the data guarantee range of data guarantee level 1.

Figure 4-55: Data guarantee range of data guarantee level 1

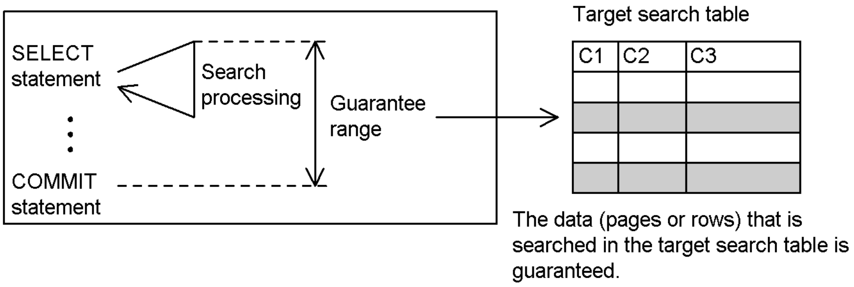


(3) Data guarantee level 2

Specify data guarantee level 2 to prevent other users from updating data that has been searched once until the transaction ends. Data that has been searched is therefore guaranteed until the end of the transaction. However, data that has not been searched is not guaranteed. If the same rows are searched twice in the same transaction, the first and second transaction results may not be the same if there are added rows.

The following figure shows the data guarantee range of data guarantee level 2.

Figure 4-56: Data guarantee range of data guarantee level 2



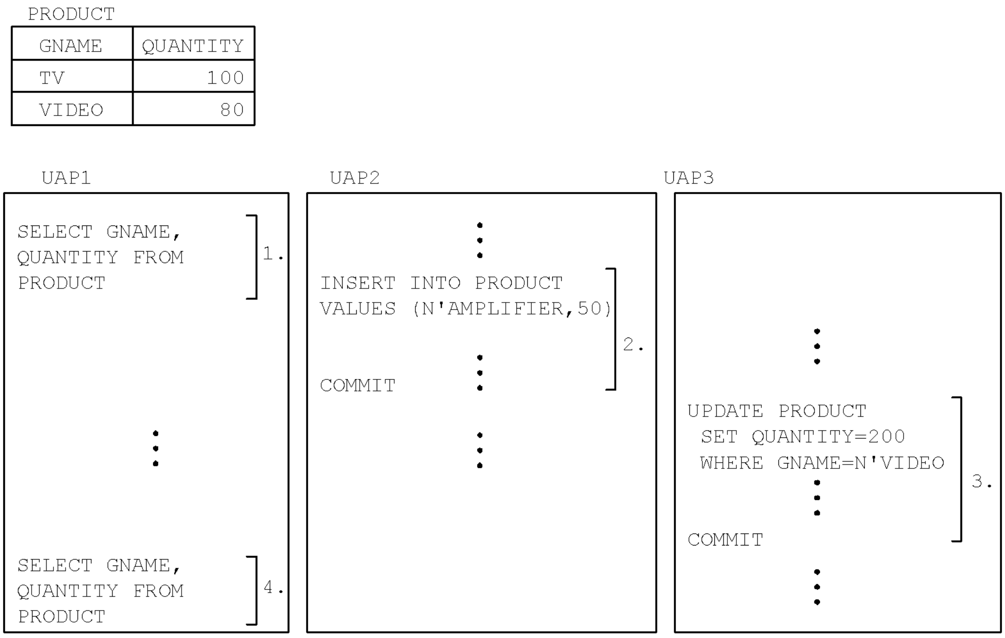
(4) Notes

If data guarantee level 0 is specified for a cursor declaration that accompanies an update, the specification is ignored, and level 1 is assumed.

4.6.3 Example of search results when a data guarantee level is specified

The figure below shows an example of search results when a data guarantee level is specified. `UAP1` is a UAP that searches the `PRODUCT` table, `UAP2` is a UAP that inserts data into the `PRODUCT` table, and `UAP3` is a UAP that updates `PRODUCT` table data. The numbers 1. to 4. show the execution sequence of `UAP1`, `UAP2`, and `UAP3`.

Figure 4-57: Example of search results when a data guarantee level is specified



PRODUCT

GNAME	QUANTITY
TV	100
VIDEO	80

UAP1

```
SELECT GNAME,
QUANTITY FROM
PRODUCT
      :
SELECT GNAME,
QUANTITY FROM
PRODUCT
```

UAP2

```
      :
INSERT INTO PRODUCT
VALUES (N'AMPLIFIER,50)
      :
COMMIT
      :
```

UAP3

```
      :
UPDATE PRODUCT
 SET QUANTITY=200
 WHERE GNAME=N'VIDEO
      :
COMMIT
      :
```

When the UAPs are executed as shown previously, the search results of 1. and 4. of `UAP1` are as shown as follows. In this example, `UAP1`, `UAP2`, and `UAP3` are all executed at the same data guarantee level.

Data guarantee level	UAP1 search results		Explanation
	1.	4.	
0	TV 100 VIDEO 80	TV 100 VIDEO 200 AMPLIFIER 50	• Although the same data is searched twice in the same transaction, the search results of 1. and 4. are different. • Because the data is not guaranteed, the processing for 2. and 3. is reflected in the search results for 4.
1	TV 100 VIDEO 80	TV 100 VIDEO 200 AMPLIFIER 50	• Although the same data is searched twice in the same transaction, the search results of 1. and 4. are different. • Although the data is guaranteed during search processing, the data is no longer guaranteed once a search process ends. Consequently, the processing for 2. and 3. is reflected in the search results for 4.

Data guarantee level	UAP1 search results		Explanation
	1.	4.	

2	`TV 100`
	`VIDEO 80`
		`TV 100`
		`VIDEO 80`
		`AMPLIFIER 50`
			• Although the same data is searched twice in the same transaction, the search results of 1. and 4. are different.
			• When the data is searched in 1., the `TELEVISION` and `VIDEO` rows are guaranteed. However, because the data is not guaranteed in 2., the processing of 2. is reflected. However, the processing for 3. enters wait status because the data is guaranteed, and the processing for 3. is not reflected in the search results for 4.

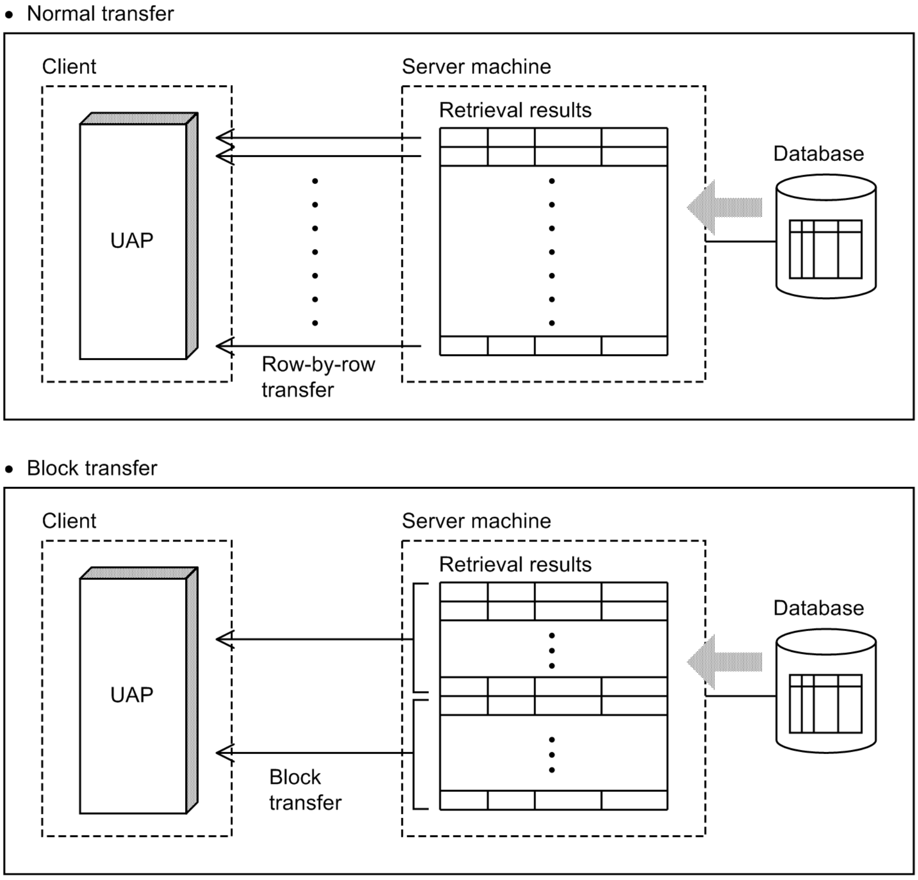
4.7 Block transfer facility

(1) Overview of the block transfer facility

Block transfer means that the HiRDB system sends data to a HiRDB client in units of a specified number of rows. The block transfer facility is useful when a HiRDB client accesses the HiRDB system to retrieve a large amount of data.

The following figure provides an overview of the block transfer facility.

Figure 4-58: Overview of block transfer facility



(2) Usage method

The block transfer facility is executed when both of the following conditions are satisfied:

1. When at least two values are specified in client environment definition PDBLKF or when at least one value is specified in PDBLKBUFFSIZE
2. When the FETCH statement is specified (except when one of the following conditions applies)
   - Update using a cursor
   - Retrieval involving a BLOB-type selection expression
   - Retrieval in which the value of client environment definition PDBINARYBLKF is NO and involving a BINARY-type selection expression with a definition length of 32,001 bytes or more
   - Retrieval that uses a BLOB locator type or BINARY locator type variable to accept results and that uses a holdable cursor

(3) Specification of communication buffer size between server and client

You can use client environment definition PDBLKBUFFSIZE to specify the communication buffer size between the server and the client.

For retrievals in which the number of rows to be extracted (PDBLKF specification value) is large, specifying PDBLKBUFFSIZE suppresses the allocation of a communication buffer memory larger than the value specified in the server. However, a communication buffer memory for transferring one row is allocated.

For details about the calculation equation for the communication buffer size between the server and the client, see *Formula for size of memory required during block transfer or array FETCH* in the *HiRDB Version 9 Installation and Design Guide*.

(4) Number of rows transferred in one transmission

The table below shows the number of rows transferred in one transmission when the block transfer facility is used.

PDBLKF specification value	PDBLKBUFFSIZE specification value	
	0	1 or higher
1	Block transfer facility does not apply.	Number of rows = MIN(X, 4096)[#] X: The number of rows becomes the maximum value (number of rows that can be stored in the specified buffer size) of n that satisfies the following condition expression. However, if (a - b) < ci, then the number of rows becomes 1 (i is 1). $(a - b) \geq \sum_{i=1}^{n} ci$ (unit: bytes) ci: Data length of the i-th row in the search results received with the FETCH statement a: Specified buffer size (PDBLKBUFFSIZE value x 1024) b: Header information and other information (864 + 22 x d + 2 x e) The d and e variables in the calculation expression for b are described below. d: Number of retrieval items specified in the SELECT clause e: Number of BINARY-type selection items in retrieval items specified in the SELECT clause
2 or higher	Number of rows = PDBLKF value	Number of rows = MIN(X, Y)[*] X: Number of rows that can be stored in specified buffer size (same as X shown above) Y: PDBLKF value

#: Certain SQL statements may be able to transfer more than the calculated number of rows.

(5) Notes

1. If one of the following events occurs, HiRDB interrupts retrieval processing and returns the data that was retrieved to that point:
    - A warning error occurs during retrieval processing. (HiRDB returns the warning information and the data that was retrieved to that point.)[#]
    - During a search via a list, a row that was present when the list was created is deleted or an attribute value is deleted or updated. (HiRDB returns return code information (SQLCODE=+110) that indicates the event and the data that was retrieved to that point.)

    #: HiRDB may not interrupt retrieval processing even if a warning error occurs.

If HiRDB does not interrupt processing, it continues retrieval processing until the specified number of rows and returns all warning error information that occurred during the retrieval, and the retrieved data.

2. The block transfer facility can shorten the retrieval time because it decreases the communication overhead by transferring a large number of rows at a time. However, the facility must be used with caution because it increases the amount of required memory. When client environment definition `PDBLKBUFFSIZE` is specified, the memory size used for the communication buffer is held below a fixed value. However, if the value is too small, the block transfer facility becomes ineffective because the number of communications cannot be reduced.
3. When the block transfer facility is being used and the search results of one cursor are received with multiple `FETCH` statements, specify the same embedded variable or embedded variables with the same attribute in each of those `FETCH` statements. If you try to receive the search results with embedded variables having different attributes, an error occurs.

4.8 Facilities using arrays

4.8.1 FETCH facility using arrays

(1) Overview

You can use the FETCH statement to fetch the retrieval results for multiple rows at a time. To do this, specify an array-type embedded variable in the INTO clause or specify the number of retrieval rows in an embedded variable of the BY clause. This method is effective when the HiRDB client accesses the HiRDB system and retrieves a large volume of data. Unlike the block transfer facility, the FETCH facility using arrays clearly specifies in the program that multiple rows of retrieval results are to be fetched.

(2) Usage methods

(a) Static execution

Convert all embedded and indicator variables specified in the INTO clause of the FETCH statement into array-type variables. The number of rows to be retrieved at one time becomes the minimum number of array elements for the specified embedded variables.

(b) Dynamic execution

To execute the FETCH facility using arrays:

1. Use the PREPARE statement to preprocess the SELECT statement.
2. Use the DESCRIBE statement to fetch information about the SQL descriptor area of the preprocessed SELECT statement.
3. In the SQLDATA area indicated in the SQL descriptor area, specify the receiving area for each data item. For variable-length data, specify the size of one element in the SQLSYS area.
4. Specify the SQL descriptor area in the USING DESCRIPTOR clause of the FETCH statement and specify an embedded variable in the BY clause. Use the embedded variable to specify the number of rows to be retrieved at one time.

(3) Notes

1. A cursor specified with the FETCH facility using arrays becomes a dedicated cursor for that facility. When that cursor is used, the block transfer facility becomes ineffective. If that cursor is used to execute the normal FETCH facility, Note 4 applies. When the same module (preprocessing unit) uses both the FETCH facility using arrays and the normal FETCH facility, use a separate cursor for each.
2. Note that, unlike the normal FETCH facility, the FETCH facility using arrays fetches data up to the row before the NOT FOUND occurrence if the rows to be

fetched run out during retrieval processing. Similarly, if an error occurs, the FETCH facility using arrays fetches the data up to the row in which the error occurred.

3. If the FETCH facility using arrays is executed dynamically, the UAP area may be destroyed if the number of rows specified in the embedded variable of the BY clause is larger than the receiving area.
4. The FETCH facility using arrays cannot be used if one of the following conditions applies:
   - A query specification contains a BLOB-type selection expression.
   - A query specification contains a BINARY-type selection expression, and the defined length for one element in the receiving area of the BINARY-type selection expression is not a multiple of 4.
   - The search includes a BINARY-type selection expression having a defined length of 32,001 bytes or more, and the version of either HiRDB Server or HiRDB Client Library is 07-00 or earlier.

(4) Usage examples

Following is a coding example of a FETCH operation using arrays:

Example 1

This example uses FETCH statement format 3. The target table consists of the PCODE (CHAR(4)), PNAME (VARCHAR(17)), COLOR (NCHAR(1)), PRICE (INTEGER), and SQUANTITY (INTEGER) columns.

```
long sel_cnt;
long data_cnt;
short i;
char work[17];

/* Declaration of array-type embedded variables */
EXEC SQL BEGIN DECLARE SECTION;
    char    xpcode[50][5];
    SQL  TYPE  IS  VARCHAR(17)  xpname[50];
    char    xcolor[50][3];
    long    xprice[50];
    long    xsquantity[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE CR3 CURSOR FOR
       SELECT PCODE,PNAME,COLOR,PRICE,SQUANTITY
       FROM STOCK;

EXEC SQL WHENEVER SQLERROR GOTO FIN;
```

```
  EXEC SQL OPEN CR3;

  /*  Heading */

  printf("    *****  Stock Table List  *****\n\n");
  printf("   Product code  Product name  Color  Price     Stock
quantity\n");
  printf("  ----  ----------------  --  --------  --------\n");

  EXEC SQL WHENEVER SQLERROR GOTO FIN;
  EXEC SQL WHENEVER NOT FOUND GOTO FIN;

  /* FETCH */
  sel_cnt = 0;
  for(;;){
      EXEC SQL
          FETCH CR3 INTO
:xpcode,:xpname,:xcolor,:xprice,:xsquantity;
      /* Store total row count retrieved with this  */
      /* cursor to SQLERRD2                         */
      data_cnt = SQLERRD2 - sel_cnt;       /* Calculate number
of retrieved rows */
      for(i=0; i < data_cnt; i++){
          memcpy(work, xpname[i].str, xpname[i].len);
          work[xpname[i].len] = '\0';
          printf("    %4s      %-16s  %2s  %8d  %8d\n",
                  xpcode[i], work, xcolor[i], xprice[i],
xsquantity[i]);
      }
      sel_cnt = SQLERRD2;
  }

FIN:
/*                                                  */
/* Display remaining data because data is read even */
/* if error or NOT FOUND occurs                     */
/*                                                  */
  if(sel_cnt  !=  SQLERRD2){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          memcpy(work, xpname[i].str, xpname[i].len);
          work[xpname[i].len] = '\0';
          printf("    %4s      %-16s  %2s  %8d  %8d\n",
                  xpcode[i], work, xcolor[i], xprice[i],
xsquantity[i]);
      }
  }
```

```
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

Example 2

This example uses FETCH statement format 2. The target table consists of the PCODE (CHAR(4)), PNAME (VARCHAR(17)), COLOR (NCHAR(1)), PRICE (INTEGER), and SQUANTITY (INTEGER) columns.

```
#include <pdbsqlda.h>      /* Include this file to use */
                           /* user-defined SQLDA*/

  long sel_cnt;
  long data_cnt;
  short i;
  char work[17];

  /* Declaration of user-defined SQLDA */
  PDUSRSQLDA(5)  xsqlda;

  /* Declaration of array-type embedded variables */
  EXEC SQL BEGIN DECLARE SECTION;
      char    xpcode[50][5];
      SQL  TYPE  IS  VARCHAR(17)  xpname[50];
      char    xcolor[50][3];
      long    xprice[50];
      long    xsquantity[50];
      short   arry_num;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL WHENEVER SQLERROR GOTO FIN;

  /* Preprocessing of retrieval SQL */
  EXEC SQL PREPARE SEL1 FROM
         'SELECT * FROM STOCK' ;

  /* Acquisition of retrieval SQL output information */
  PDSQLN(xsqlda) = 5 ;      /* Set SQLVAR count */
  EXEC SQL DESCRIBE SEL1 INTO :xsqlda ;

  EXEC SQL
      DECLARE CR3 CURSOR FOR SEL1 ;

  EXEC SQL OPEN CR3;

  /* SQLVAR setting: Normally, it would better if I/O */
  /* area was allocated dynamically from SQLDA.       */
  /* However, the specification is omitted because    */
  /* this is an example.                              */
  /* Values that were set during DESCRIBE processing  */
  /* are used for SQLLEN, SQLXDIM, and SQLSYS.        */
  /* PCODE column information settings */
  PDSQLDATA(xsqlda, 0) = (void *)xpcode ;    /* Set address */
  PDSQLIND(xsqlda, 0) = NULL ;               /* Clear NULL
indicator variable */
  PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;        /* Set data code */
  /* PNAME column information settings */
  PDSQLDATA(xsqlda, 1) = (void *) xpname;    /* Set address */
  PDSQLIND(xsqlda, 1) = NULL ;               /* Clear NULL
indicator variable */
  PDSQLCOD(xsqlda, 1) = PDSQL_VARCHAR ;     /* Set data code */
  PDSQLSYS(xsqlda, 1) = sizeof(xpname[0]) ;  /* Set SQLSYS
because this is */
                                            /* variable-length
data                 */
  /* COLOR column information settings */
  PDSQLDATA(xsqlda, 2) = (void *) xcolor;     /* Set address */
  PDSQLIND(xsqlda, 2) = NULL ;               /* Clear NULL
indicator variable */
  PDSQLCOD(xsqlda, 2) = PDSQL_NCHAR ;       /* Set data code */
  /* PRICE column information settings */
  PDSQLDATA(xsqlda, 3) = (void *) xprice;    /* Set address */
  PDSQLIND(xsqlda, 3) = NULL ;               /* Clear NULL
indicator variable */
  PDSQLCOD(xsqlda, 3) = PDSQL_INTEGER ;     /* Set data code */
  /* SQUANTITY column information settings */
 PDSQLDATA(xsqlda, 4) = (void *) xsquantity;   /* Set address */
  PDSQLIND(xsqlda, 4) = NULL ;               /* Clear NULL
indicator variable */
  PDSQLCOD(xsqlda, 4) = PDSQL_ INTEGER;     /* Set data code */

  /*  Heading */

  printf("    *****  Stock Table List  *****\n\n");
 printf("   Product code Product name         Color  Price
Stock quantity\n");
  printf("    ----        ----------------  --  --------
--------\n");

  EXEC SQL WHENEVER SQLERROR GOTO FIN;
  EXEC SQL WHENEVER NOT FOUND GOTO FIN;

  /* FETCH */
  sel_cnt = 0;
  for(;;){
      arry_num = 50 ;
      EXEC SQL
          FETCH CR3 USING DESCRIPTOR :xsqlda BY :arry_num ROWS ;
      /* Store total row count retrieved with this  */
      /* cursor to SQLERRD2                         */
      data_cnt = SQLERRD2 - sel_cnt;       /* Calculate number
of fetched rows */
      for(i=0; i < data_cnt; i++){
          memcpy(work, xpname[i].str, xpname[i].len);
          work[xpname[i].len] = '\0';
          printf("    %4s      %-16s  %2s  %8d  %8d\n",
               xpcode[i], work, xcolor[i], xprice[i], xsquantity
[i]);
      }
      sel_cnt = SQLERRD2;
  }

FIN:
/*                                                        */
/* Display remaining data because data is read even       */
/* if error or NOT FOUND occurs                           */
/*                                                        */
  if(sel_cnt  !=  SQLERRD2){
      data_cnt = SQLERRD2 - sel_cnt;
      for(i=0; i < data_cnt; i++){
          memcpy(work, xpname[i].str, xpname[i].len);
          work[xpname[i].len] = '\0';
          printf("    %4s      %-16s  %2s  %8d  %8d\n",
               xpcode[i], work, xcolor[i], xprice[i], xsquantity
[i]);
      }
  }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

Example 3

The example uses FETCH statement format 3. The target table consists of the XCODE (INTEGER) and ROW_DATA (BINARY(3002)) columns.

```
  long sel_cnt;
  long data_cnt;
  short i;

  /* Declaration of array-type embedded variables */
  EXEC SQL BEGIN DECLARE SECTION;
      long    xcode[50];
      /* To fetch data using BINARY-type array,   */
      /* define area length with multiple of 4    */
      SQL  TYPE  IS  BINARY(3004)  xrow_data[50];
  EXEC SQL END DECLARE SECTION;

  EXEC SQL
      DECLARE CR3 CURSOR FOR
         SELECT * FROM T_BINARY;

  EXEC SQL WHENEVER SQLERROR GOTO FIN;

  EXEC SQL OPEN CR3;

  /*  Heading */

  printf("    *****  Binary Data Table  *****\n\n");

  EXEC SQL WHENEVER SQLERROR GOTO FIN;
  EXEC SQL WHENEVER NOT FOUND GOTO FIN;

  /* FETCH */
  sel_cnt = 0;
  for(;;){

      EXEC SQL
          FETCH CR3 INTO : xcode,: xrow_data;
      /* Store total row count retrieved with this  */
      /* cursor to SQLERRD2                         */
      data_cnt = SQLERRD2 - sel_cnt;       /* Calculate number
of fetched rows */
      for(i=0; i < data_cnt; i++){
          printf("    CODE=%8d\n",xcode[i]);
          printf("    DATA_LENGTH=%d\n", xrow_data [i].len);
/* Do not display BINARY data section because this is */
/* only an example                                     */
/* Convert xrow_data[i].str to individual format of   */
/* each UAP                                            */
      }
      sel_cnt = SQLERRD2;
  }

FIN:
/*                                                     */
/* Display remaining data because data is read even   */
/* if error or NOT FOUND occurs                        */
/*                                                     */
  if(sel_cnt  !=  SQLERRD2){
      data_cnt = SQLERRD2 - sel_cnt;
```

```
        for(i=0; i < data_cnt; i++){
            printf("    CODE=%8d\n",xcode[i]);
            printf("    DATA_LENGTH=%d\n", xrow_data [i].len);
/* Do not display BINARY data section because this is */
/* only an example                                     */
/* Convert xrow_data[i].str to individual format of    */
/* each UAP                                            */
        }
    }
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL CLOSE CR3;
  EXEC SQL COMMIT;
```

Example 4

This example uses FETCH statement format 2. The target table consists of the XCODE (INTEGER) and ROW_DATA (BINARY(3002)) columns.

```
#include <pdbsqlda.h>     /* Include this file to use */
                          /* user-defined SQLDA       */

  long sel_cnt;
  long data_cnt;
  short i;

  /* Declaration of user-defined SQLDA */
  PDUSRSQLDA(2)  xsqlda;

  /* Declaration of array-type embedded variable */
  EXEC SQL BEGIN DECLARE SECTION;
      long    xcode[50];
      /* To fetch data using BINARY-type array,    */
      /* define area length with multiple of 4     */
      SQL  TYPE  IS  BINARY(3004)  xrow_data[50];
      short    arry_num;
  EXEC SQL END DECLARE SECTION;

  EXEC SQL WHENEVER SQLERROR GOTO FIN;

  /* Preprocessing of retrieval SQL */
  EXEC SQL PREPARE SEL1 FROM
         'SELECT * FROM T_BINARY ;

  /* Acquisition of retrieval SQL output information */
  PDSQLN(xsqlda) = 2 ;       /* Set SQLVAR count */
  EXEC SQL DESCRIBE SEL1 INTO :xsqlda ;

  EXEC SQL
```

```
      DECLARE CR3 CURSOR FOR SEL1 ;

  EXEC SQL OPEN CR3;

  /* SQLVAR setting: Normally, it would better if I/O */
  /* area was allocated dynamically from SQLDA.       */
  /* However, the specification is omitted because    */
  /* this is an example.                              */
  /* Values that were set during DESCRIBE processing  */
  /* are used for SQLLEN, SQLXDIM, and SQLSYS.        */
  /* XCODE column information settings */
  PDSQLDATA(xsqlda, 0) = (void *)xcode ;    /* Set address */
  PDSQLIND(xsqlda, 0) = NULL ;           /* Clear NULL indicator 
variable */
  PDSQLCOD(xsqlda, 0) = PDSQL_INTEGER ;     /* Set data code */
  /* R_DATA column information settings */
  PDSQLDATA(xsqlda, 1) = (void *) xrow_data; /* Set address */
  PDSQLIND(xsqlda, 1) = NULL ;              /* Clear NULL 
indicator variable */
  PDSQLCOD (xsqlda, 1) = PDSQL_BINARY ;      /* Set data code*/
  PDSQLLEN (xsqlda, 1) = 3004 ;              /* Reset because 
defined length */
                                      /* is not multiple of 4 */

  /*  Heading */

  printf("    *****  Binary Data Table  *****\n\n");

  EXEC SQL WHENEVER SQLERROR GOTO FIN;
  EXEC SQL WHENEVER NOT FOUND GOTO FIN;

  /* FETCH */
  sel_cnt = 0;
  for(;;){
      arry_num = 50 ;
      EXEC SQL
         FETCH CR3 USING DESCRIPTOR :xsqlda BY :arry_num ROWS ;
      /* Store total row count retrieved with this  */
      /* cursor to SQLERRD2                         */
      data_cnt = SQLERRD2 - sel_cnt;       /* Calculate number 
of fetched rows */
      for(i=0; i < data_cnt; i++){
          printf("    CODE=%8d\n",xcode[i]);
          printf("    DATA_LENGTH=%d\n", xrow_data [i].len);
/* Do not display BINARY data section because this is an example 
*/
/* Convert xrow_data[i].str to individual format of each UAP */
      }
```

```
        sel_cnt = SQLERRD2;
    }

  FIN:
  /*                                                      */
  /* Display remaining data because data is read even    */
  /* if error or NOT FOUND occurs                         */
  /*                                                      */
    if(sel_cnt  !=  SQLERRD2){
        data_cnt = SQLERRD2 - sel_cnt;
        for(i=0; i < data_cnt; i++){
            printf("    CODE=%8d\n",xcode[i]);
            printf("    DATA_LENGTH=%d\n", xrow_data [i].len);
  /* Do not display BINARY data section because this is */
  /* only an example                                      */
  /* Convert xrow_data[i].str to individual format of    */
  /* each UAP                                             */
        }
    }
  FIN:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL CLOSE CR3;
    EXEC SQL COMMIT;
```

4.8.2 INSERT facility using arrays

(1) Overview

You can insert multiple rows of data with one SQL statement by specifying an array-type variable in which the data for the multiple rows has been set. Using the INSERT facility using arrays reduces the number of communications between the HiRDB client and the HiRDB server. This facility is therefore effective when you want to access the HiRDB server from the HiRDB client and insert a large volume of data at high speed.

(2) Usage methods

(a) Static execution

Specify the embedded variables in the INSERT statement's FOR clause and use an array-type variable to specify all embedded and indicator variables. The embedded variables specified in the FOR clause control the number of rows that can be inserted at one time (*batch insertion*).

(b) Dynamic execution

To execute the INSERT facility using arrays:

1. Use the PREPARE statement to preprocess the INSERT statement (specify one or more ? parameters).

2. In the USING clause of the EXECUTE statement, use an array to specify the values to be assigned to the input ? parameter of the preprocessed INSERT statement, and specify an embedded variable in the BY clause. Use the embedded variable specified in the BY clause to control the number of rows to be inserted by batch insertion.

   If you specify an embedded variable in the USING clause, change all embedded and indicator variables to array-type variables.

   If you specify an SQL descriptor area in the USING clause, use the array format to specify data in all areas indicated by SQLDATA. In the SQLSYS area, specify values that correspond to the data type.

(3) Note

If a row count that exceeds the write area is specified in the embedded variable in the FOR clause of the INSERT statement or the BY clause of the EXECUTE statement, DB destruction or UAP area destruction may occur.

(4) Usage examples

Explained as follows are coding examples for the INSERT facility using arrays.

Example 1

This example uses INSERT statement format 3 to set the data read from the file into an array-format embedded variable and to insert 50 rows at a time into the STOCK table.

The target table is consists of the PCODE (CHAR(4)), PNAME (VARCHAR(17)), COLOR (NCHAR(1)), PRICE (INTEGER), and SQUANTITY (INTEGER) columns.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_pcode[5];
  short in_pname_len;
  char in_pname[17];
  char in_color[3];
  int in_price;
  int i;
  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* Declare array-type embedded variables */
    char   xpcode[50][5];           /* For specifying value to
be inserted */
                                     /* in PCODE (CHAR(4) type
column)      */
    SQL TYPE IS VARCHAR(17) xpname[50];
                                    /* For specifying value to
be inserted */
                                      /* in PNAME (VARCHAR(17)
type column) */
    char   xcolor[50][3];           /* For specifying value to
be inserted */
                                    /* in COLOR (NCHAR(1) type
column)      */
    long   xprice[50];              /* For specifying value to
be inserted */
                                     /* in PRICE (INTEGER type
column)      */
    EXEC SQL END DECLARE SECTION;

   -------(CONNECT processing to HiRDB (omitted))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
  fprintf(stderr, "can't open %s.", INFILE);
  goto FIN;
  }

    EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

    /* Batch insertion row count (up to 50 rows) */
    xinsert_num=50;
    while (!feof(input)) {
    /* Set input data for 50 rows (if last data in file, */
    /* up to that row) to array variables               */
    for (i = 0; i < 50; i++) {
      /* Read data from file */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
       /* If last data in file, set row count up to last data */
       /* in batch insertion row count, and escape for statement
*/
        xinsert_num= i;
        break;
      }
      sscanf(indata, "%4s %hd %16s %2s %8d",
      in_pcode, &in_pname_len, in_pname, in_color, &in_price);
      /* Set input data into array variable elements */
      strncpy(xpcode[i], in_pcode, 5);
      xpname[i].len = in_pname_len;
      strncpy(xpname[i].str, in_pname, 17);
      strncpy(xcolor[i], in_color, 3);
      xprice[i] = in_price;
    }
    /* INSERT execution */
    EXEC SQL FOR :xinsert_num
      INSERT INTO STOCK (PCODE, PNAME, COLOR, PRICE)
      VALUES (:xpcode, :xpname, :xcolor, :xprice);
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
    fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

Example 2

This example uses INSERT statement format 3 to set data read from the data read function to an array-type embedded variable and to insert 50 rows at a time into the STOCK table.

The target table consists of the PCODE (CHAR(4)) and ROW_DATA (BINARY(3002)) columns.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void abnormalend();

main() {
  int i,rc;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* Declaration of array-type embedded variables */
    char  xpcode[50][5];         /* For specifying value to be
inserted */
                            /* in PCODE (CHAR(4) type column) */
    SQL TYPE IS BINARY(3004) xrow_data[50];
                    /* For specifying value to be inserted in
ROW_DATA (BINARY(3002) type column) */
                /* However, set data length to multiple of 4. */
    EXEC SQL END DECLARE SECTION;

   -------(CONNECT processing to HiRDB (omitted))-------

    EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  rc = 0 ;
  /* Batch insertion row count (up to 50 rows) */
  xinsert_num=50;
  while (0==rc) {
    /* Set input data for 50 rows (if last data in file, */
    /* up to that row) to array variables               */
    for (i = 0; i < 50; i++) {
      /* Read BINARY data: Function details omitted */
      rc = get_binarydata(&xpcode[i],&xrow_data[i]);
      if (0 != rc){
      /* If input data runs out, set row count up to last data */
      /* in batch insertion row count, and escape for statement
*/
        xinsert_num= i;
        break;
      }
    }
    /* INSERT execution */
    EXEC SQL FOR :xinsert_num
      INSERT INTO STOCK (PCODE, ROW_DATA)
      VALUES (:xpcode, :xrow_data);
  }
  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

Example 3

This example uses EXECUTE statement format 2 to set data read from a file to array-format embedded variables and insert 50 rows at a time into the STOCK table.

The target table consists of the PCODE (CHAR(4)), PNAME (VARCHAR(17)), COLOR (NCHAR(1)), PRICE (INTEGER), and SQUANTITY (INTEGER) columns.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_pcode[5];
  short in_pname_len;
  char in_pname[17];
  char in_color[3];
  int in_price;
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
```

```
    short  xinsert_num;
    /* Declaration of array-type embedded variables */
    char  xpcode[50][5];     /* For specifying value to be
inserted */
                           /* in PCODE (CHAR(4) type column) */
    SQL TYPE IS VARCHAR(17) xpname[50];
                        /* For specifying value to be inserted
in PNAME (VARCHAR(17) type column) */
   char  xcolor[50][3];  /* For specifying value to be inserted
in COLOR (NCHAR(1) type column) */
    long  xprice[50];   /* For specifying value to be inserted
in PRICE (INTEGER type column) */
  EXEC SQL END DECLARE SECTION;

   -------(CONNECT processing to HiRDB (omitted))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* SQL preprocessing execution */
  EXEC SQL PREPARE INS1 FROM
    'INSERT INTO STOCK(PCODE, PNAME, COLOR, PRICE)
VALUES(?,?,?,?)';

  /* Batch insertion row count (up to 50 rows) */
  xinsert_num=50;
  while (!feof(input)) {
    /* Set input data for 50 rows (if last data in file, */
    /* up to that row) to array variables               */
    for (i = 0; i < 50; i++) {
      /* Read data from file */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
      /* If input data runs out, set row count up to last data */
      /* in batch insertion row count, and escape for statement
*/
        xinsert_num= i;
        break;
      }
      sscanf(indata, "%4s %hd %16s %2s %8d",
      in_pcode, &in_pname_len, in_pname, in_color, &in_price);
      /* Set input data to array variable elements */
      strncpy(xpcode[i], in_pcode, 5);
      xpname[i].len = in_pname_len;
      strncpy(xpname[i].str, in_pname, 17);
      strncpy(xcolor[i], in_color, 3);
      xprice[i] = in_price;
    }
    /* EXECUTE execution */
    EXEC  SQL  EXECUTE  INS1
      USING  :xpcode, :xpname, :xcolor, :xprice
      BY :xinsert_num ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
  fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

Example 4

This example uses EXECUTE statement format 2 to set data read from a file to array-format embedded variables and uses a user-defined SQLDA to insert 50 rows at a time into the STOCK file.

The target table consists of the PCODE (CHAR(4)), PNAME (VARCHAR(17)), COLOR (NCHAR(1)), PRICE (INTEGER), and SQUANTITY (INTEGER) columns.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pdbsqlda.h>         /* Include file to use */
```

```
                                         /* user-defined SQLDA  */

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_pcode[5];
  short in_pname_len;
  char in_pname[17];
  char in_color[3];
  int in_price;
  int i;

  /* Declaration of user-defined SQLDA */
  PDUSRSQLDA(4)  xsqlda;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* Declaration of array-type embedded variables */
    char  xpcode[50][5];       /* For specifying value to be
inserted */
                            /* in PCODE (CHAR(4) type column) */
    SQL TYPE IS VARCHAR(17) xpname[50];
                      /* For specifying value to be inserted to
PNAME (VARCHAR(17) type column) */
    char  xcolor[50][3];      /* For specifying value to be
inserted to COLOR (NCHAR(1) type column) */
   long  xprice[50];     /* For specifying value to be inserted
to PRICE (INTEGER type column) */
  EXEC SQL END DECLARE SECTION;

   -------(CONNECT processing to HiRDB (omitted))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* SQL preprocessing execution */
  EXEC SQL PREPARE INS1 FROM
```

```
    'INSERT INTO STOCK(PCODE, PNAME, COLOR, PRICE)
VALUES(?,?,?,?)';

  /* SQLVAR settings */
  PDSQLN(xsqlda) = 4 ;       /* Set SQLVAR count */
  PDSQLD(xsqlda) = 4 ;       /* Set ? parameter count */
  /* Set PCODE column information */
  PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;  /* Set data code */
  PDSQLXDIM(xsqlda, 0) = 1 ;        /* Set number of repeated */
                                      /* structure elements */
  PDSQLSYS(xsqlda, 0) = 0 ;          /* Length of one element */
                                     /* (fixed to 0 except for
variable-length character strings)        */
  PDSQLLEN(xsqlda, 0) = 4 ;        /* Set data defined length */
  PDSQLDATA(xsqlda, 0) = (void *)xpcode ; /* Set data area
address */
  PDSQLIND(xsqlda, 0) = NULL ;        /* Clear NULL indicator
variable */
  /* Set PNAME column information */
  PDSQLCOD(xsqlda, 1) = PDSQL_VARCHAR ;  /* Set data code */
  PDSQLXDIM(xsqlda, 1) = 1 ;        /* Set number of repeated */
                                      /* structure elements */
  PDSQLLEN(xsqlda, 1) = 17 ;       /* Set data defined length */
  PDSQLSYS(xsqlda, 1) = sizeof(xpname[0]) ;  /* Length of one
element */
  PDSQLDATA(xsqlda, 1) = (void *) xpname;  /* Set data area
address */
  PDSQLIND(xsqlda, 1) = NULL ;        /* Clear NULL indicator
variable */
  /* Set COLOR column information */
  PDSQLCOD(xsqlda, 2) = PDSQL_NCHAR ;  /* Set data code */
  PDSQLXDIM(xsqlda, 2) = 1 ;        /* Set number of repeated */
                                      /* structure elements */
  PDSQLSYS(xsqlda, 2) = 0 ;          /* Length of one element */
                                     /* (fixed to 0 except for
variable-length character strings)      */
  PDSQLLEN(xsqlda, 2) = 1 ;        /* Set data defined length */
  PDSQLDATA(xsqlda, 2) = (void *) xcolor;  /* Set data area
address */
  PDSQLIND(xsqlda, 2) = NULL ;         /* Clear NULL indicator
variable */
  /* Set PRICE column information */
  PDSQLCOD(xsqlda, 3) = PDSQL_INTEGER ;  /* Set data code */
  PDSQLXDIM(xsqlda, 3) = 1 ;        /* Set number of repeated */
                                      /* structure elements */
  PDSQLSYS(xsqlda, 3) = 0 ;          /* Length of one element */
                                     /* (fixed to 0 except for
variable-length character strings)        */
```

```
  PDSQLLEN(xsqlda, 3) = 4 ;          /* Set data defined length */
  PDSQLDATA(xsqlda, 3) = (void *) xprice;  /* Set data area
address */
  PDSQLIND(xsqlda, 3) = NULL ;         /* Clear NULL indicator
variable */

  /* Batch insertion row count (up to 50 rows) */
  xinsert_num=50;
  while (!feof(input)) {
    /* Set input data for 50 rows (if last data in file, */
    /* up to that row) to array variables               */
    for (i = 0; i < 50; i++) {
      /* Read data from file */
        fgets(indata, MAXCOLUMN, input);
        if (feof(input)){
       /* If last data in file, set row count up to last data */
      /* in batch insertion row count, and escape for statement
*/
          xinsert_num= i;
          break;
        }
        sscanf(indata, "%4s %hd %16s %2s %8d",
       in_pcode, &in_pname_len, in_pname, in_color, &in_price);
        /* Set input data to array variable elements */
        strncpy(xpcode[i], in_pcode, 5);
        xpname[i].len = in_pname_len;
        strncpy(xpname[i].str, in_pname, 17);
        strncpy(xcolor[i], in_color, 3);
        xprice[i] = in_price;
      }
      /* EXECUTE execution */
      EXEC SQL EXECUTE INS1
        USING DESCRIPTOR :xsqlda
        BY :xinsert_num ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  if (input != NULL) {
    fclose(input);
  }
    EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
```

```
{
  int  wsqlcode;

  if (input != NULL) {
  fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

Example 5

This example uses EXECUTE statement format 2 to set data read by a data read function to array-type embedded variables and uses a user-defined SQLDA to insert 50 rows at a time into the STOCK file.

The target table consists of the PCODE (CHAR(4)) and ROW_DATA (BINARY(3002)) columns.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pdbsqlda.h>         /* Include file for using */
                              /* user-defined SQLDA */

void abnormalend();

main() {
  int i,rc;

  /* Declaration of user-defined SQLDA */
  PDUSRSQLDA(4)  xsqlda;

  EXEC SQL BEGIN DECLARE SECTION;
    short  xinsert_num;
    /* Declaration of array-type embedded variables */
   char  xpcode[50][5];   /* For specifying value to be inserted
*/
                           /* to PCODE (CHAR(4) type column) */
    SQL TYPE IS BINARY(3004) xrow_data[50];
                  /* For specifying value to be inserted to
ROW_DATA (BINARY(3002) type column) */
                 /* However, set data length to multiple of 4 */
  EXEC SQL END DECLARE SECTION;
```

```
   -------(CONNECT processing to HiRDB (omitted))-------

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* SQL preprocessing execution */
  EXEC SQL PREPARE INS1 FROM
    'INSERT INTO STOCK(PCODE, ROW_DATA) VALUES(?,?)';

  /* SQLVAR settings */
  PDSQLN(xsqlda) = 2 ;                /* Set SQLVAR count */
  PDSQLD(xsqlda) = 2 ;              /* Set ? parameter count */
  /* Set PCODE column information */
  PDSQLCOD(xsqlda, 0) = PDSQL_CHAR ;  /* Set data code */
  PDSQLXDIM(xsqlda, 0) = 1 ;        /* Set number of repeated */
                                    /* structure elements */
  PDSQLSYS(xsqlda, 0) = 0 ;         /* Length of one element */
                                    /* (fixed to 0 except for
variable-length character strings)            */
  PDSQLLEN(xsqlda, 0) = 4 ;        /* Set data defined length */
  PDSQLDATA(xsqlda, 0) = (void *)xpcode ;  /* Set data area
address */
  PDSQLIND(xsqlda, 0) = NULL ;      /* Clear NULL indicator
variable */
  /* Set ROW_DATA column information */
  PDSQLCOD(xsqlda, 1) = PDSQL_BINARY ;  /* Set data code */
  PDSQLXDIM(xsqlda, 1) = 1 ;        /* Set number of repeated */
                                    /* structure elements */
  PDSQLLOBLEN(xsqlda, 1) = 3004 ;  /* Set data defined length */
  PDSQLDATA(xsqlda, 1) = (void *) xrow_data;  /* Set data   */
                                              /* area address */
  PDSQLIND(xsqlda, 1) = NULL ;      /* Clear NULL indicator
variable */

  rc = 0 ;
  /* Batch insertion row count (up to 50 rows) */
  xinsert_num=50;
  while (0==rc) {
    /* Set input data for 50 rows (if last data in file, */
    /* up to that row) to array variables               */
    for (i = 0; i < 50; i++) {
      /* Read BINARY data: Function details omitted */
      rc = get_binarydata(&xpcode[i],&xrow_data[i]);
      if (0 != rc){
       /* If input data runs out, set row count up to last data */
       /* in batch insertion row count, and escape for statement
*/
        xinsert_num= i;
        break;
      }
    }
    /* EXECUTE execution */
    EXEC  SQL  EXECUTE  INS1
      USING DESCRIPTOR :xsqlda
      BY :xinsert_num ROWS ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.8.3 UPDATE facility using arrays

(1) Overview

You can update multiple table columns with one SQL statement by specifying an array-type variable in which the data for multiple columns has been set.

Using the UPDATE facility using arrays reduces the number of communications between the HiRDB client and the HiRDB server. This facility is therefore effective when you want to access the HiRDB server from the HiRDB client and update a large volume of data at high speed.

(2) Usage methods

(a) Static execution

In the UPDATE statement, specify an embedded variable in the FOR clause and change all embedded and indicator variables specified in the search condition to array-type variables. Use the embedded variable specified in the FOR clause to control the number of updates to be performed by batch processing.

(b) Dynamic execution

To execute the UPDATE facility using arrays:

1. Use the PREPARE statement to preprocess the UPDATE statement. (Specify the ? parameter for the update values and in the search condition.)
2. In the USING clause of the EXECUTE statement, use an array to specify the values to be assigned to the input ? parameter of the preprocessed UPDATE statement, and specify an embedded variable in the BY clause. Use the embedded variable specified in the BY clause to control the number of updates to be performed by batch processing.

Notes about dynamic execution are described below.

- If you specify an embedded variable in the USING clause, change all embedded and indicator variables to array-type variables.
- If you specify an SQL Descriptor Area in the USING clause, use the array format to specify data in all areas indicated by SQLDATA. In the SQLSYS area, specify values that correspond to the data type.

(3) Note

1. If a count that exceeds the write area is specified in the embedded variable in the FOR clause of the UPDATE statement or the BY clause of the EXECUTE statement, database destruction or UAP area destruction may occur.

(4) Usage example

Example

This example sets the data read from a file into an array-format embedded variable and performs several updates to the STOCK table by batch processing.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_pcode[5];
  int in_quantity;
  int i;
  EXEC SQL BEGIN DECLARE SECTION;
    short   xupdate_num;
    /* Declare array-type embedded variables */
    char    xpcode[50][5]; /* For search condition to PCODE (CHAR(4) type
column) */
    long    squantity[50];      /* For specifying update value to SQUANTITY
(INTEGER type column) */
  EXEC SQL END DECLARE SECTION;

      -------(CONNECT processing to HiRDB (omitted))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* Batch update count (up to 50 updates) */
  xupdate_num=50;
  while (!feof(input)) {
    /* Set update/search condition data for 50 updates (if last */
    /* data in file, up to that row) to array variables */
    for (i = 0; i < 50; i++) {
      /* Read data from file */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* If last data in file, set array elements up to last data */
        /* in batch update count, and escape for statement */
        xupdate_num= i;
        break;
      }
      sscanf(indata, "%4s %8d", in_pcode, &in_quantity);
      /* Set update/search condition data into array variable elements */
      strncpy(xpcode[i], in_pcode, 5);
      xquantity[i] = in_quantity;
    }
    /* UPDATE execution */
    EXEC SQL FOR :xupdate_num
      UPDATE STOCK SET ZQUANTITY = :xquantity WHERE PCODE =
:xpcode ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  if (input != NULL) {
  fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;

  if (input != NULL) {
    fclose(input);
  }
  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.8.4 DELETE facility using arrays

(1) Overview

You can delete multiple rows with one SQL statement by specifying an array-type variable in which the data for multiple deletions has been set.

Using the DELETE facility using arrays reduces the number of communications between the HiRDB client and the HiRDB server. This facility is therefore effective when you want to access the HiRDB server from the HiRDB client and delete a large volume of data at high speed.

(2) Usage methods

(a) Static execution

In the DELETE statement, specify an embedded variable in the FOR clause and change all embedded and indicator variables specified in the search condition to array-type variables. Use the embedded variable specified in the FOR clause to control the number of deletions to be performed by batch processing.

(b) Dynamic execution

To execute the DELETE facility using arrays:

1. Use the PREPARE statement to preprocess the DELETE statement (specify the ? parameter in the search condition).

2. In the USING clause of the EXECUTE statement, use an array to specify the values to be assigned to the input ? parameter of the preprocessed DELETE statement, and specify an embedded variable in the BY clause. Use the embedded variable specified in the BY clause to control the number of deletions to be performed by batch processing.

Notes about dynamic execution are described below.

- If you specify an embedded variable in the USING clause, change all embedded and indicator variables to array-type variables.
- If you specify an SQL Descriptor Area in the USING clause, use the array format to specify data in all areas indicated by SQLDATA. In the SQLSYS area, specify values that correspond to the data type.

(3) Note

1. If a count that exceeds the write area is specified in the embedded variable in the BY clause of the EXECUTE statement, database destruction or UAP area destruction may occur.

(4) Usage example

Example

This example sets the data read from a file into an array-format embedded variable and performs several deletions from the STOCK table by batch processing.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80
#define INFILE   "inputf1"

void abnormalend();

FILE *input ;

main() {
  char indata[MAXCOLUMN];
  char in_pcode[5];
  int i;

  EXEC SQL BEGIN DECLARE SECTION;
    short   xdelete_num;
    /* Declare array-type embedded variables */
    char     xpcode[50][5]; /* For search condition to PCODE (CHAR(4) type
column) */
```

```
  EXEC SQL END DECLARE SECTION;

      -------(CONNECT processing to HiRDB (omitted))-------

  input = fopen(INFILE, "r");
  if (input == NULL) {
    fprintf(stderr, "can't open %s.", INFILE);
    goto FIN;
  }

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;

  /* Batch deletion count (up to 50 deletions) */
  xdelete_num=50;
  while (!feof(input)) {
    /* Set search condition data for 50 deletions (if last */
    /* data in file, up to that row) to array variables */
    for (i = 0; i < 50; i++) {
      /* Read data from file */
      fgets(indata, MAXCOLUMN, input);
      if (feof(input)){
        /* If last data in file, set array element count up to last data */
        /* in batch deletion count and escape for statement */
        xdelete_num= i;
        break;
      }
      sscanf(indata, "%4s", in_pcode);
      /* Set search condition data into array variable elements */
      strncpy(xpcode[i], in_pcode, 5);
    }
    /* DELETE execution */
    EXEC SQL FOR :xdelete_num
      DELETE FROM STOCK WHERE PCODE = :xpcode ;
  }

  EXEC SQL COMMIT;
  printf(" *** normal ended ***\n");
FIN:
  if (input != NULL) {
    fclose(input);
  }
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;
  if (input != NULL) {
    fclose(input);
   }
  wsqlcode = -SQLCODE;
  printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

4.9 Rapid grouping facility

4.9.1 Overview

When the GROUP BY clause of the SQL is specified for grouping, grouping is performed after sorting. Rapid grouping is accomplished by combining hashing with grouping. The rapid grouping facility reduces the time required for grouping as the number of groups to be grouped gets smaller and as the number of rows gets larger.

In HiRDB/Parallel Server, you must also consider the grouping processing method, because how the floatable servers are used affects performance. For details about grouping processing methods, see *4.5.5 Grouping processing methods (HiRDB/Parallel Server only)*.

4.9.2 Application criteria

The rapid grouping facility can be used when an SQL that satisfies all the following conditions is executed:

■ **HiRDB/Parallel Server**

- The GROUP BY clause is specified.
- Use of the rapid grouping facility is defined in the system common definitions, front-end server definitions, client environment definitions, or routine definitions.
- The selection expression column length is 4,096 or less.
- The grouping process is not for an inquiry specification that becomes the input for a set operation (UNION, EXCEPT).
- DISTINCT is not specified within the set function.
- A character string-type column with a defined length of 256 bytes or more, a BINARY-type column, or a BLOB-type column is not specified in the set function.
- When a HAVING clause is specified in the specification of a query in which the GROUP BY clause is specified, no subquery is specified in the HAVING clause.
- No subquery is specified in the selection expression.

In the following case, rapid grouping is performed regardless of whether or not the SQL optimization option is specified:

- Grouping can be executed without sorting, using a grouping column index.

The following facilities cannot be used when the rapid grouping facility is used:

- Facility for creating multiple objects
- AND multi-index use (however, this function is used with structured repetition predicates and functions dedicated to index type plug-ins).

■ **HiRDB/Single Server**

- The GROUP BY clause is specified.
- Use of the rapid grouping facility is defined in the system common definitions, client environment definitions, or routine definitions.
- The grouping process is not for an inquiry specification that becomes the input for a set operation (UNION, EXCEPT).
- DISTINCT is not specified within the set function.
- A character string-type column with a defined length of 256 bytes or more, a BINARY-type column, or a BLOB-type column is not specified in the set function.

In the following case, rapid grouping is performed regardless of whether or not the SQL optimization option is specified:

- Grouping can be executed without sorting, using a grouping column index or by sorting join columns for sort/merge join.

The following facility is not used when the rapid grouping facility is used:

- AND multi-index use (however, this function is used with structured repetition predicates and functions dedicated to index type plug-ins).

4.9.3 Specification method

To use the rapid grouping facility, specify in the SQL optimization options either RAPID_GROUPING or a value to which 1,024 has been added. For details about how to specify the SQL optimization options, see the following locations:

- The pd_optimize_level description in the manual *HiRDB Version 9 System Definition*
- The SQL optimization options description in the manual *HiRDB Version 9 SQL Reference*
- The PDSQLOPTLVL description in *6.6.4 Environment definition information*

4.9.4 Tuning method

If the number of groups to be grouped is high, the rapid grouping facility may not be effective. If this is the case, specify a value of the required size (number of groups or higher) in the PDAGGR operand of the client environment definitions. Note, however, that a large amount of process-specific memory may be used. If the amount of memory used is large and a value of the required size cannot be specified, specify the maximum specifiable value that is less than the number of groups. Specifying a value larger than the number of groups is no more effective than specifying a value equal to the number of groups. For details about the `PDAGGR` operand, see *6.6.4 Environment definition information*.

4.10 Multi-connection facility

(1) Overview

(a) What is the multi-connection facility?

The multi-connection facility establishes multiple connections to the HiRDB server from one UAP process in a HiRDB client.

The multi-connection facility establishes independent individual connections. A separate server process is allocated to each connection, and the connections are processed as separate transactions. The UAP can, therefore, execute multiple SQL statements simultaneously. Because multiple connections can be established from one UAP, the number of UAPs to be executed can be reduced, and the overall memory requirement for UAPs can be reduced.

Because each connection is counted as a separate user, the maximum number of server connections becomes the maximum number of simultaneous connections rather than the maximum number of users.

The multi-connection facility has the following characteristics:

- A different authorization server and password can be used for each connection.
- One UAP can connect to HiRDB servers on multiple server computers and execute SQL statements, because each connection can be connected to a server in a different server computer.
- The multi-connection facility can be used for all servers that can connect to the client library.

(b) Multi-connection facility in the X/Open XA interface environment

When the multi-connection facility is used in the X/Open XA interface environment, a UAP operating under a single transaction manager (such as OpenTP1) can use the XA interface to access multiple HiRDB systems. Because the UAP is using the XA interface, the UAP can synchronize and control processing among transactions that access multiple HiRDB systems.

For the open character string to be specified in the `xa_open()` function, specify the name of the file in which the environment variables (client environment definitions) were set. The `xa_open()` function establishes a connection to HiRDB according to those environment variables. You can select the destination to which an SQL statement is issued from among the connection destinations connected by the `xa_open()` function.

The multi-connection facility in the X/Open XA interface environment can be used only with the following client platforms:

- HP-UX 11.0
- Solaris
- AIX
- Linux (single thread)
- Windows

(2) Processing overview

Figures 4-59 through *4-63* show an overview of multi-connection facility processing.

Figure 4-59: Overview of multi-connection facility processing (when multithreading is not used)

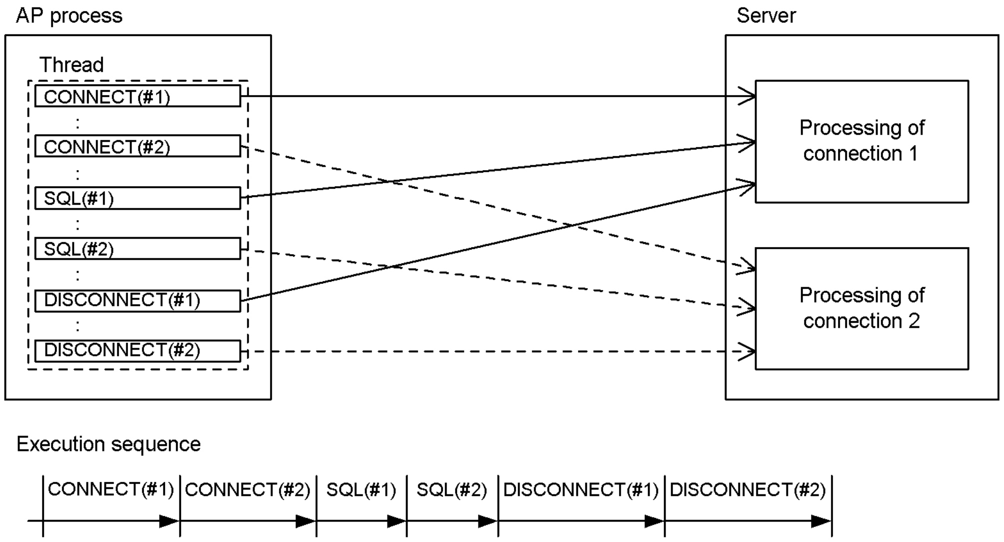
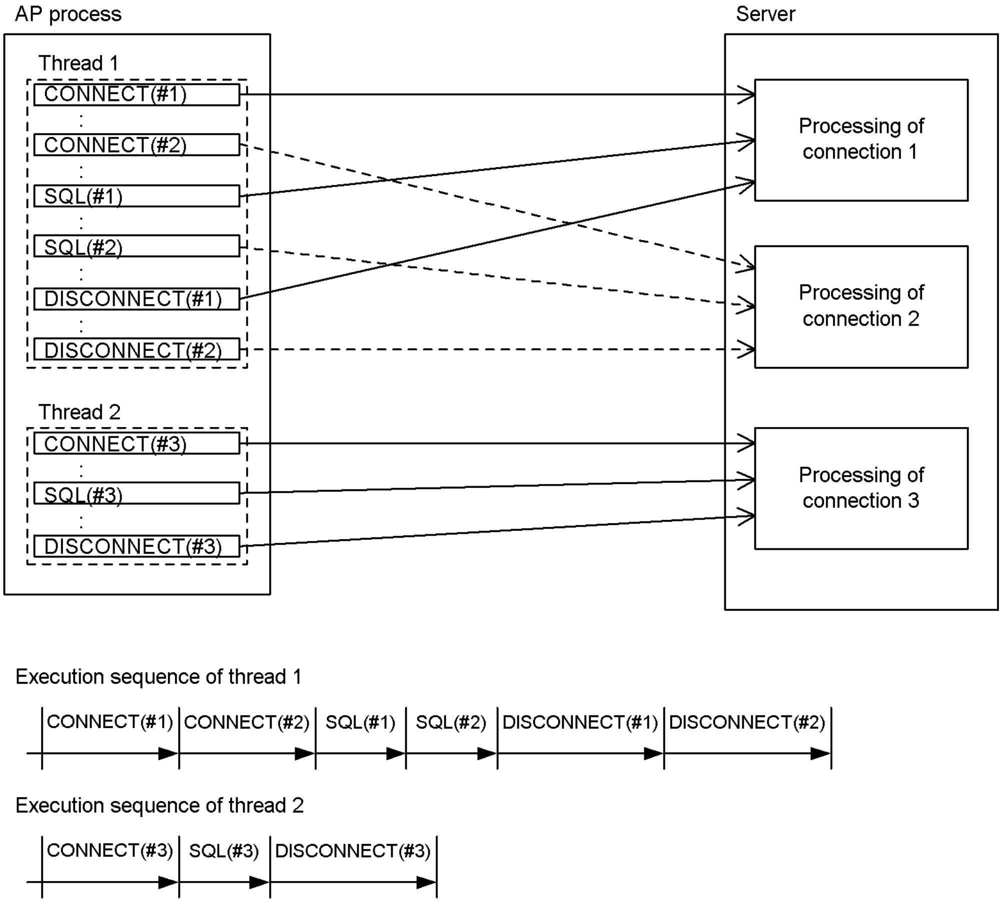
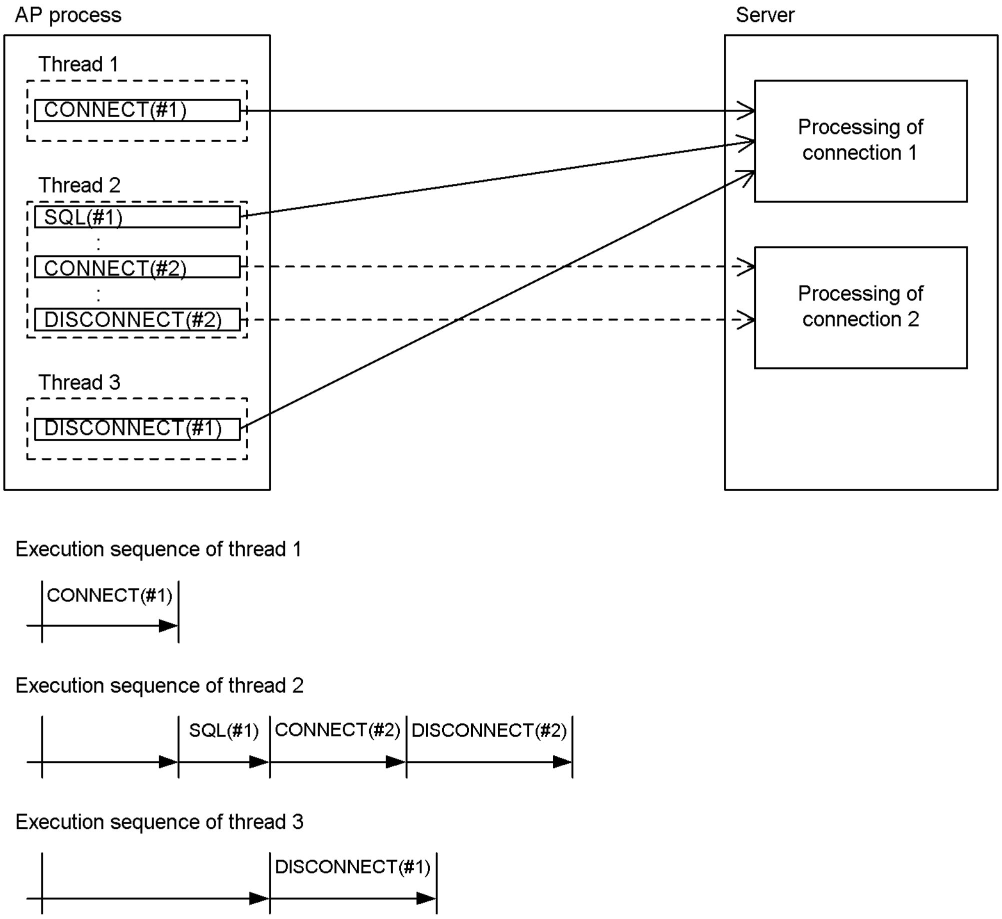


Figure 4-60: Overview of multi-connection facility processing (when multithreading is used)



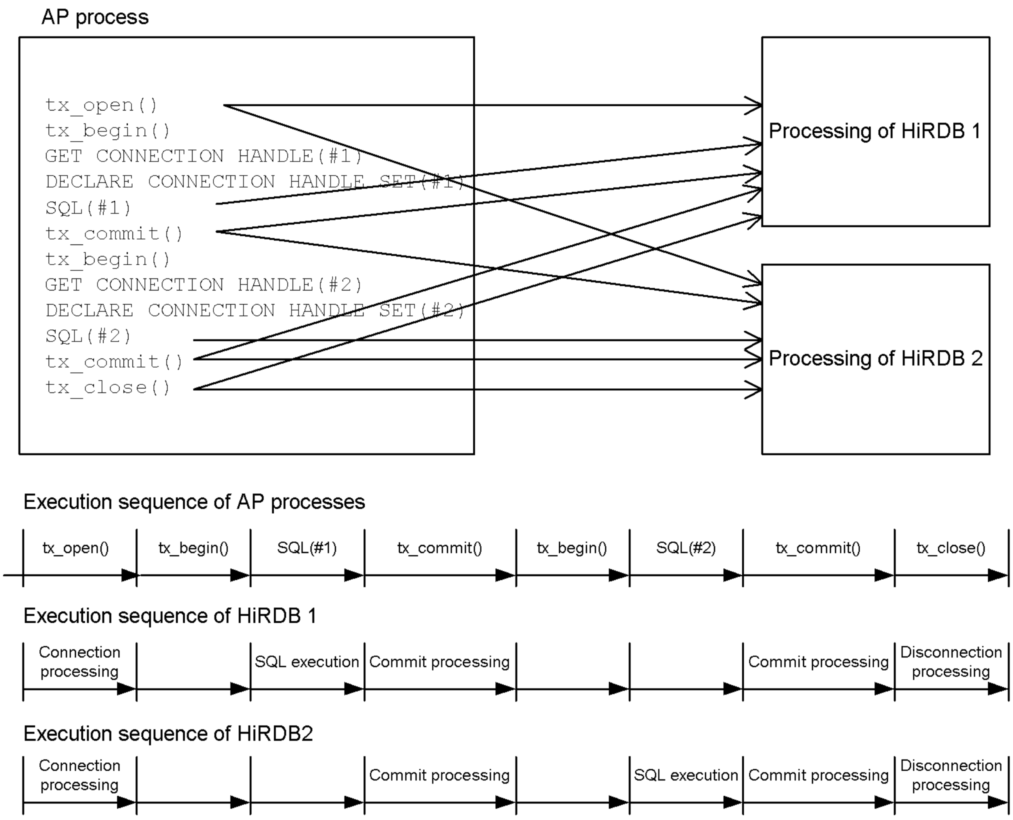
Note: The threads can execute SQL statements simultaneously because each connection is independent.

Figure 4-61: Overview of multi-connection facility processing (when a connection is shared by multiple threads)



Note: You must synchronize the processing among the threads so that, for example, thread 2 does not execute SQL (#1) or thread 3 does not execute DISCONNECT (#1) before thread 1 executes CONNECT (#1).

Figure 4-62: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a single-thread OLTP system)



```
tx_open()
tx_begin()
GET CONNECTION HANDLE(#1)
DECLARE CONNECTION HANDLE SET(#1)
SQL(#1)
tx_commit()
tx_begin()
GET CONNECTION HANDLE(#2)
DECLARE CONNECTION HANDLE SET(#2)
SQL(#2)
tx_commit()
tx_close()
```

Execution sequence of AP processes

tx_open()	tx_begin()	SQL(#1)	tx_commit()	tx_begin()	SQL(#2)	tx_commit()	tx_close()

Execution sequence of HiRDB 1

Connection processing		SQL execution	Commit processing			Commit processing	Disconnection processing

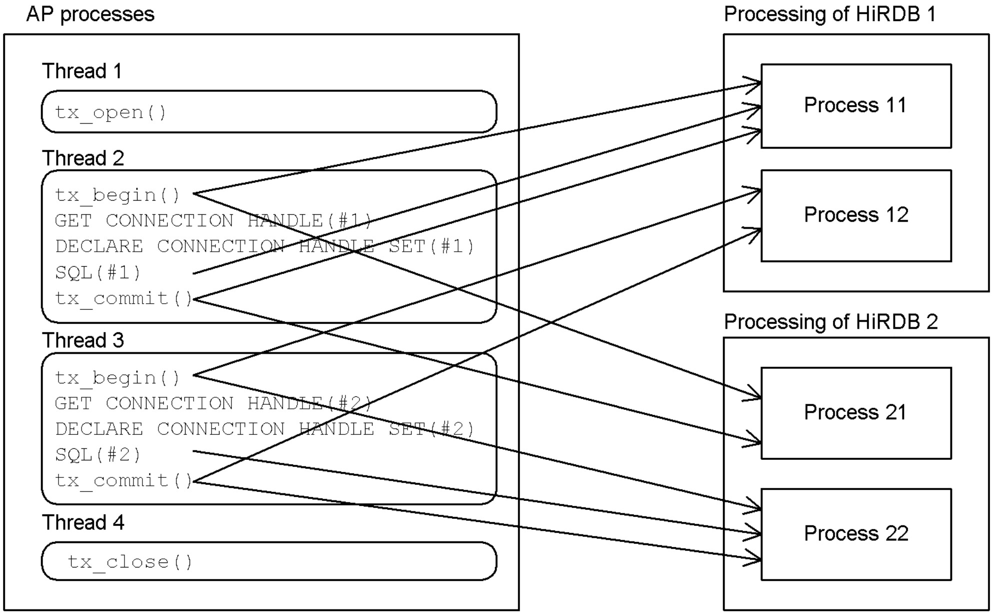
Execution sequence of HiRDB2

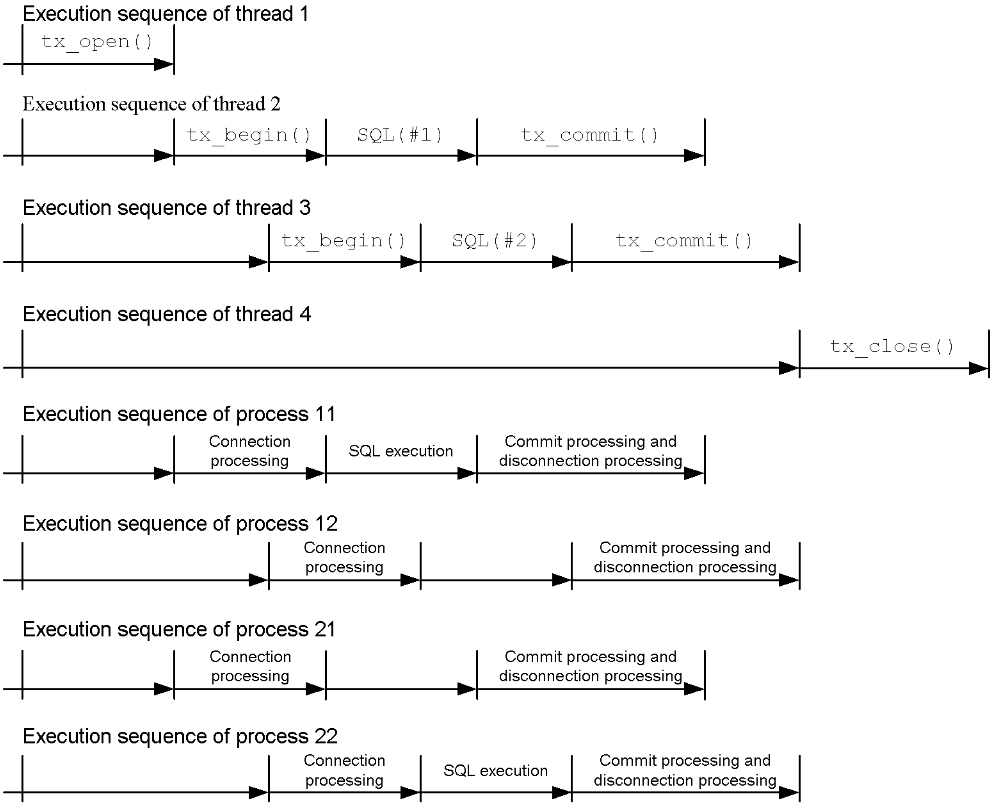
Connection processing			Commit processing		SQL execution	Commit processing	Disconnection processing

Explanation

Register HiRDB 1 and HiRDB 2 in the OLTP system beforehand. When `tx_open()` is executed, the OLTP system connects to all registered HiRDB systems. When an SQL statement is executed, select the connection destination for that SQL statement.

Figure 4-63: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a multi-thread OLTP system)




Execution sequence of thread 1
tx_open()
Execution sequence of thread 2
tx_begin()
SQL(#1)
tx_commit()
Execution sequence of thread 3
tx_begin()
SQL(#2)
tx_commit()
Execution sequence of thread 4
tx_close()
Execution sequence of process 11
Connection processing
SQL execution
Commit processing and disconnection processing
Execution sequence of process 12
Connection processing
Commit processing and disconnection processing
Execution sequence of process 21
Connection processing
Commit processing and disconnection processing
Execution sequence of process 22
Connection processing
SQL execution
Commit processing and disconnection processing

Explanation

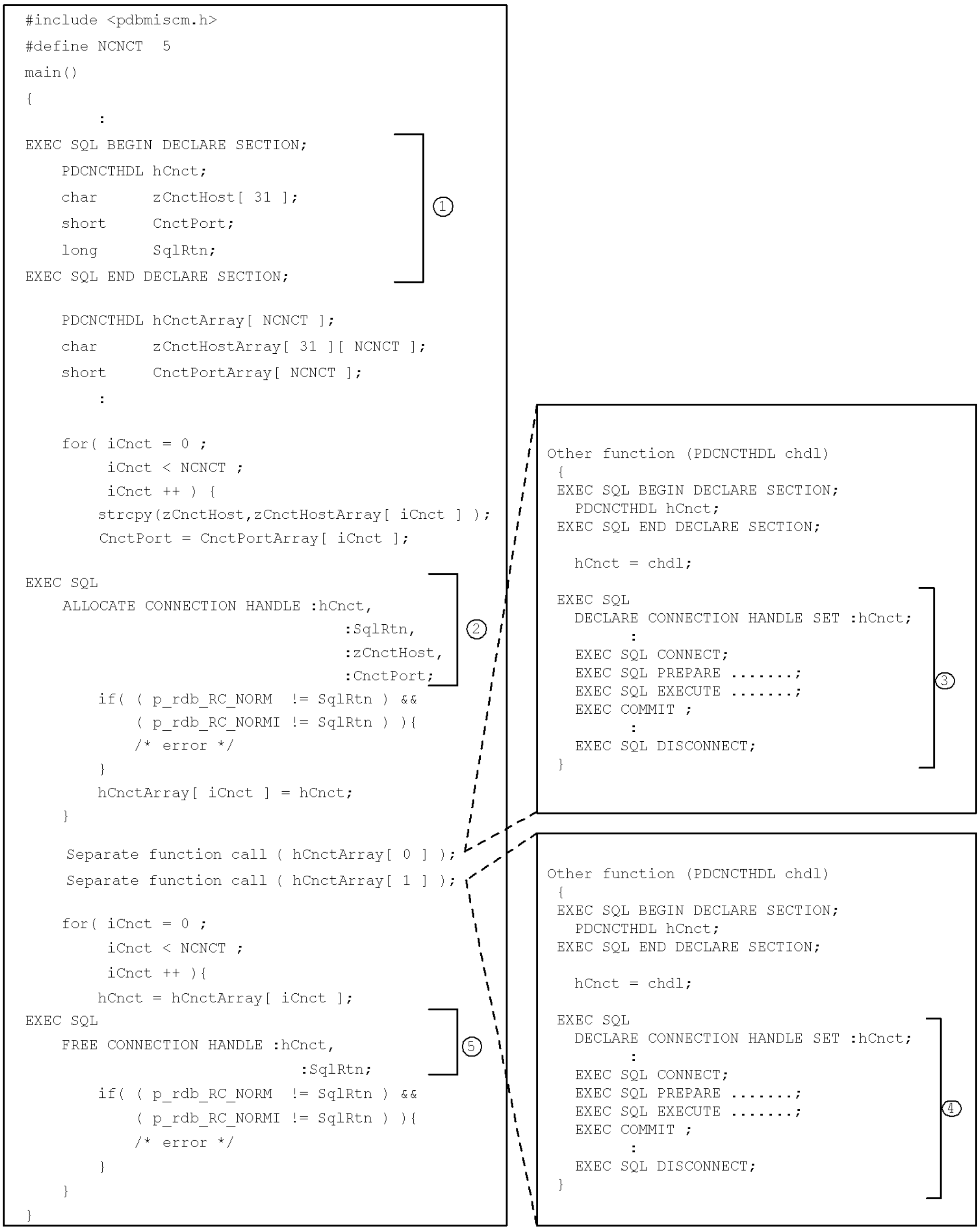
Register HiRDB 1 and HiRDB 2 in the OLTP system beforehand. When `tx_begin()` is executed, the OLTP system connects to all registered HiRDB systems. When an SQL statement is executed, select the connection destination for that SQL statement. Because the individual transactions are independent, SQL statements for different threads can be executed simultaneously.

(3) Coding example

(a) Normal UAPs

Figures 4-64 and *4-65* show coding examples of UAPs that use the multi-connection facility.

Figure 4-64: Coding example (C) of a UAP that uses the multi-connection facility



```
#include <pdbmiscm.h>
#define NCNCT  5
main()
{
        :
EXEC SQL BEGIN DECLARE SECTION;
    PDCNCTHDL hCnct;
    char      zCnctHost[ 31 ];
    short     CnctPort;
    long      SqlRtn;
EXEC SQL END DECLARE SECTION;

    PDCNCTHDL hCnctArray[ NCNCT ];
    char      zCnctHostArray[ 31 ][ NCNCT ];
    short     CnctPortArray[ NCNCT ];
        :

    for( iCnct = 0 ;
         iCnct < NCNCT ;
         iCnct ++ ) {
        strcpy(zCnctHost,zCnctHostArray[ iCnct ] );
        CnctPort = CnctPortArray[ iCnct ];

EXEC SQL
    ALLOCATE CONNECTION HANDLE :hCnct,
                               :SqlRtn,
                               :zCnctHost,
                               :CnctPort;
        if( ( p_rdb_RC_NORM  != SqlRtn ) &&
            ( p_rdb_RC_NORMI != SqlRtn ) ){
            /* error */
        }
        hCnctArray[ iCnct ] = hCnct;
    }

    Separate function call ( hCnctArray[ 0 ] );
    Separate function call ( hCnctArray[ 1 ] );

    for( iCnct = 0 ;
         iCnct < NCNCT ;
         iCnct ++ ){
        hCnct = hCnctArray[ iCnct ];
EXEC SQL
    FREE CONNECTION HANDLE :hCnct,
                           :SqlRtn;
        if( ( p_rdb_RC_NORM  != SqlRtn ) &&
            ( p_rdb_RC_NORMI != SqlRtn ) ){
            /* error */
        }
    }
}
```

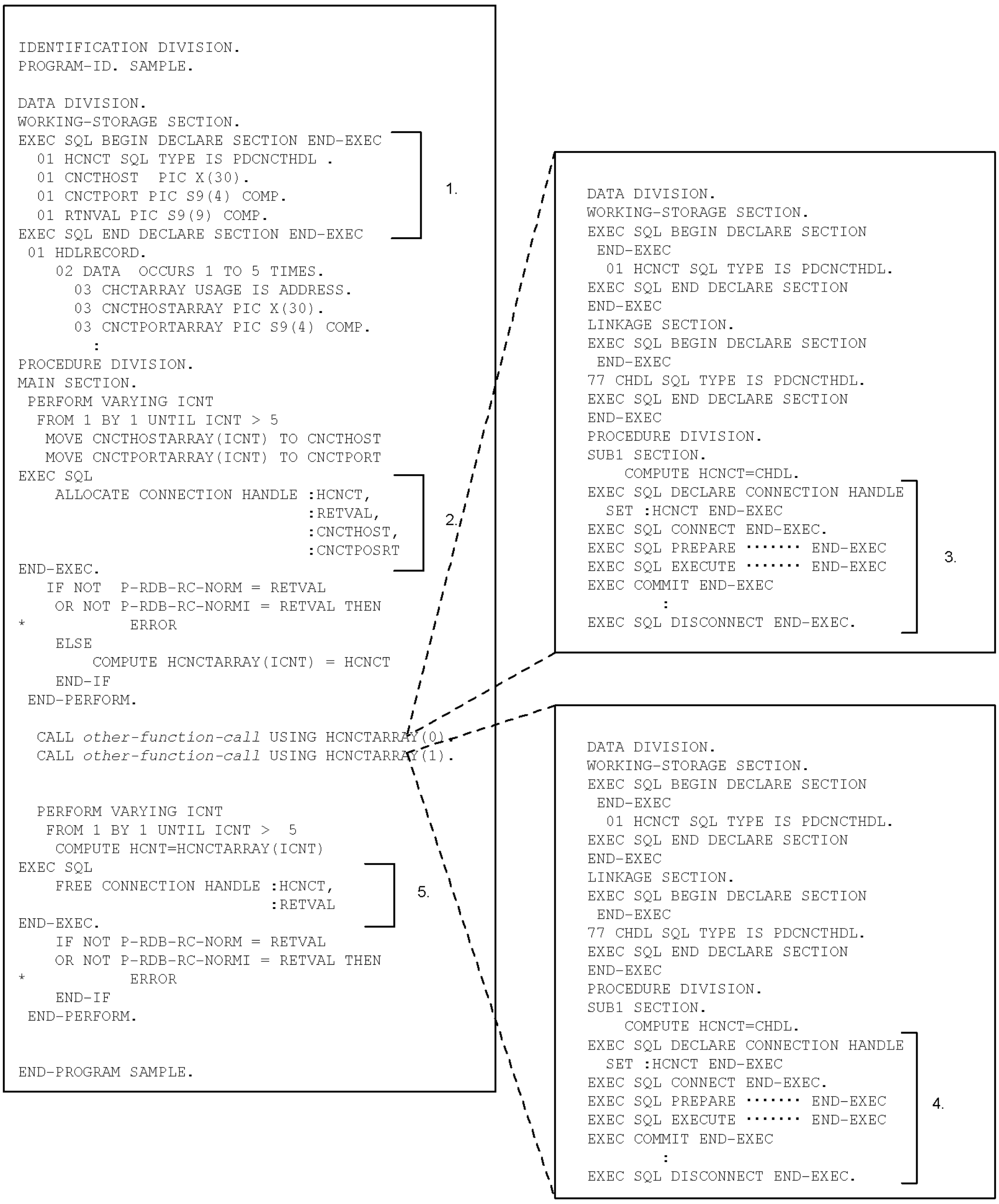
```
Other function (PDCNCTHDL chdl)
 {
 EXEC SQL BEGIN DECLARE SECTION;
   PDCNCTHDL hCnct;
 EXEC SQL END DECLARE SECTION;

   hCnct = chdl;

 EXEC SQL
   DECLARE CONNECTION HANDLE SET :hCnct;
          :
   EXEC SQL CONNECT;
   EXEC SQL PREPARE .......;
   EXEC SQL EXECUTE .......;
   EXEC COMMIT ;
          :
   EXEC SQL DISCONNECT;
 }
```

```
Other function (PDCNCTHDL chdl)
 {
 EXEC SQL BEGIN DECLARE SECTION;
   PDCNCTHDL hCnct;
 EXEC SQL END DECLARE SECTION;

   hCnct = chdl;

 EXEC SQL
   DECLARE CONNECTION HANDLE SET :hCnct;
          :
   EXEC SQL CONNECT;
   EXEC SQL PREPARE .......;
   EXEC SQL EXECUTE .......;
   EXEC COMMIT ;
          :
   EXEC SQL DISCONNECT;
 }
```

Explanation

1. Defines the connection handle.
2. Allocates the connection handle.
3. Specifies HiRDB processing for connection 1.
4. Specifies HiRDB processing for connection 2.
5. Releases the connection handle.

Figure 4-65: Coding example (COBOL) of a UAP that uses the multi-connection facility



```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.

DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL .
  01 CNCTHOST  PIC X(30).
  01 CNCTPORT PIC S9(4) COMP.
  01 RTNVAL PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC
 01 HDLRECORD.
    02 DATA  OCCURS 1 TO 5 TIMES.
      03 CHCTARRAY USAGE IS ADDRESS.
      03 CNCTHOSTARRAY PIC X(30).
      03 CNCTPORTARRAY PIC S9(4) COMP.
        :
PROCEDURE DIVISION.
MAIN SECTION.
 PERFORM VARYING ICNT
  FROM 1 BY 1 UNTIL ICNT > 5
   MOVE CNCTHOSTARRAY(ICNT) TO CNCTHOST
   MOVE CNCTPORTARRAY(ICNT) TO CNCTPORT
EXEC SQL
    ALLOCATE CONNECTION HANDLE :HCNCT,
                               :RETVAL,
                               :CNCTHOST,
                               :CNCTPOSRT
END-EXEC.
   IF NOT  P-RDB-RC-NORM = RETVAL
    OR NOT P-RDB-RC-NORMI = RETVAL THEN
*           ERROR
    ELSE
        COMPUTE HCNCTARRAY(ICNT) = HCNCT
    END-IF
 END-PERFORM.

  CALL other-function-call USING HCNCTARRAY(0)
  CALL other-function-call USING HCNCTARRAY(1).


  PERFORM VARYING ICNT
   FROM 1 BY 1 UNTIL ICNT >  5
    COMPUTE HCNT=HCNCTARRAY(ICNT)
EXEC SQL
    FREE CONNECTION HANDLE :HCNCT,
                           :RETVAL
END-EXEC.
    IF NOT P-RDB-RC-NORM = RETVAL
    OR NOT P-RDB-RC-NORMI = RETVAL THEN
*           ERROR
    END-IF
 END-PERFORM.


END-PROGRAM SAMPLE.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
LINKAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
77 CHDL SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
PROCEDURE DIVISION.
SUB1 SECTION.
    COMPUTE HCNCT=CHDL.
EXEC SQL DECLARE CONNECTION HANDLE
  SET :HCNCT END-EXEC
EXEC SQL CONNECT END-EXEC.
EXEC SQL PREPARE ······· END-EXEC
EXEC SQL EXECUTE ······· END-EXEC
EXEC COMMIT END-EXEC
        :
EXEC SQL DISCONNECT END-EXEC.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
  01 HCNCT SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
LINKAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
 END-EXEC
77 CHDL SQL TYPE IS PDCNCTHDL.
EXEC SQL END DECLARE SECTION
END-EXEC
PROCEDURE DIVISION.
SUB1 SECTION.
    COMPUTE HCNCT=CHDL.
EXEC SQL DECLARE CONNECTION HANDLE
  SET :HCNCT END-EXEC
EXEC SQL CONNECT END-EXEC.
EXEC SQL PREPARE ······· END-EXEC
EXEC SQL EXECUTE ······· END-EXEC
EXEC COMMIT END-EXEC
        :
EXEC SQL DISCONNECT END-EXEC.
```

Note

Specify the entire SQL, including the SQL prefix and terminator, in the B area (columns 12 to 72).

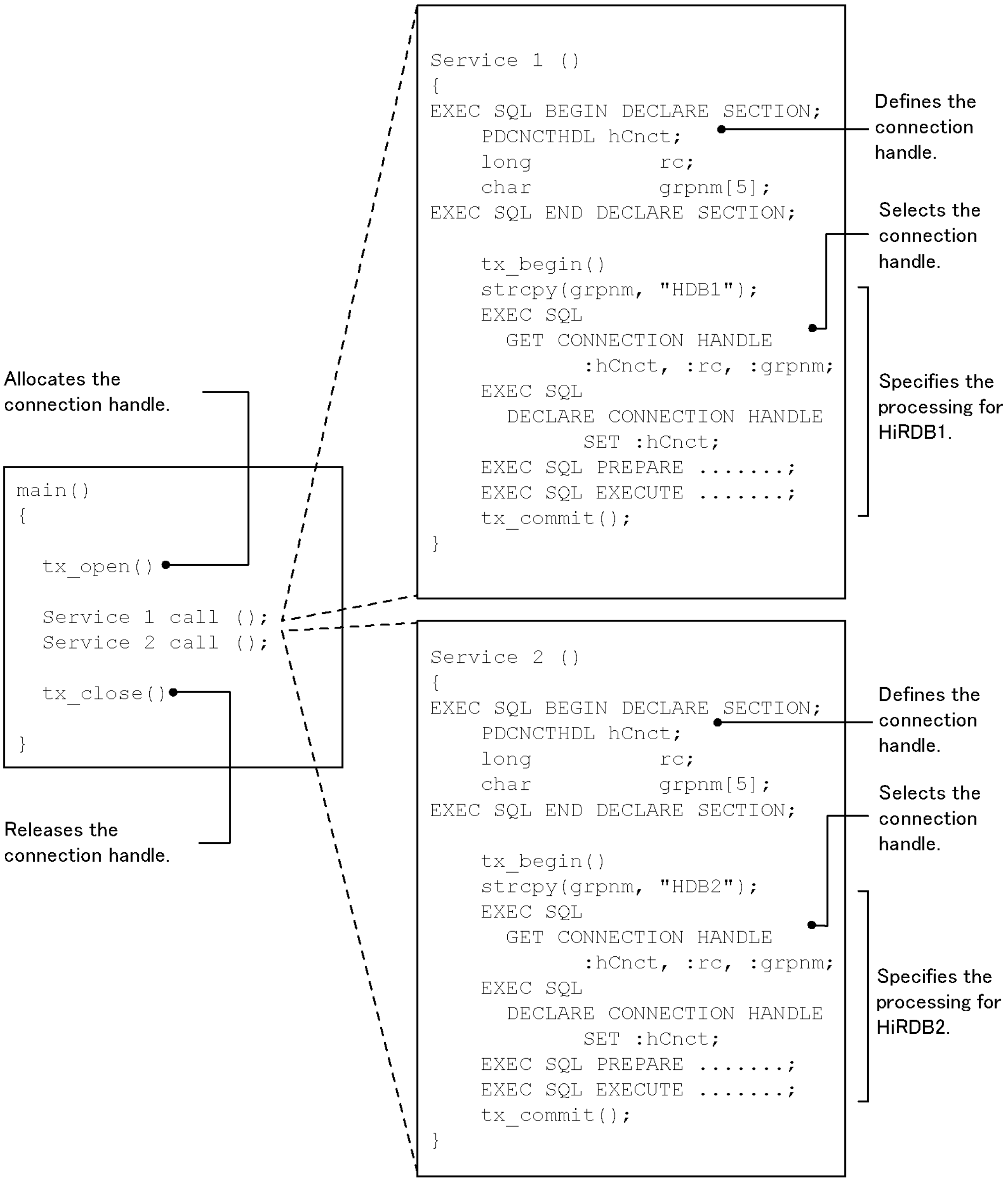
Explanation

1. Defines the connection handle.
2. Allocates the connection handle.
3. Specifies HiRDB processing for connection 1.
4. Specifies HiRDB processing for connection 2.
5. Releases the connection handle.

(b) UAPs that use an X/Open-compliant API under OLTP

Figures 4-66 and *4-67* show coding examples in which the multi-connection facility is used by UAPs that use an X/Open-compliant API under OLTP.

Figure 4-66: Coding example (C) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP



```
main()
{

  tx_open()

  Service 1 call ();
  Service 2 call ();

  tx_close()

}
```

```
Service 1 ()
{
EXEC SQL BEGIN DECLARE SECTION;
    PDCNCTHDL hCnct;
    long          rc;
    char          grpnm[5];
EXEC SQL END DECLARE SECTION;

    tx_begin()
    strcpy(grpnm, "HDB1");
    EXEC SQL
      GET CONNECTION HANDLE
            :hCnct, :rc, :grpnm;
    EXEC SQL
      DECLARE CONNECTION HANDLE
            SET :hCnct;
    EXEC SQL PREPARE .......;
    EXEC SQL EXECUTE .......;
    tx_commit();
}
```

```
Service 2 ()
{
EXEC SQL BEGIN DECLARE SECTION;
    PDCNCTHDL hCnct;
    long          rc;
    char          grpnm[5];
EXEC SQL END DECLARE SECTION;

    tx_begin()
    strcpy(grpnm, "HDB2");
    EXEC SQL
      GET CONNECTION HANDLE
            :hCnct, :rc, :grpnm;
    EXEC SQL
      DECLARE CONNECTION HANDLE
            SET :hCnct;
    EXEC SQL PREPARE .......;
    EXEC SQL EXECUTE .......;
    tx_commit();
}
```

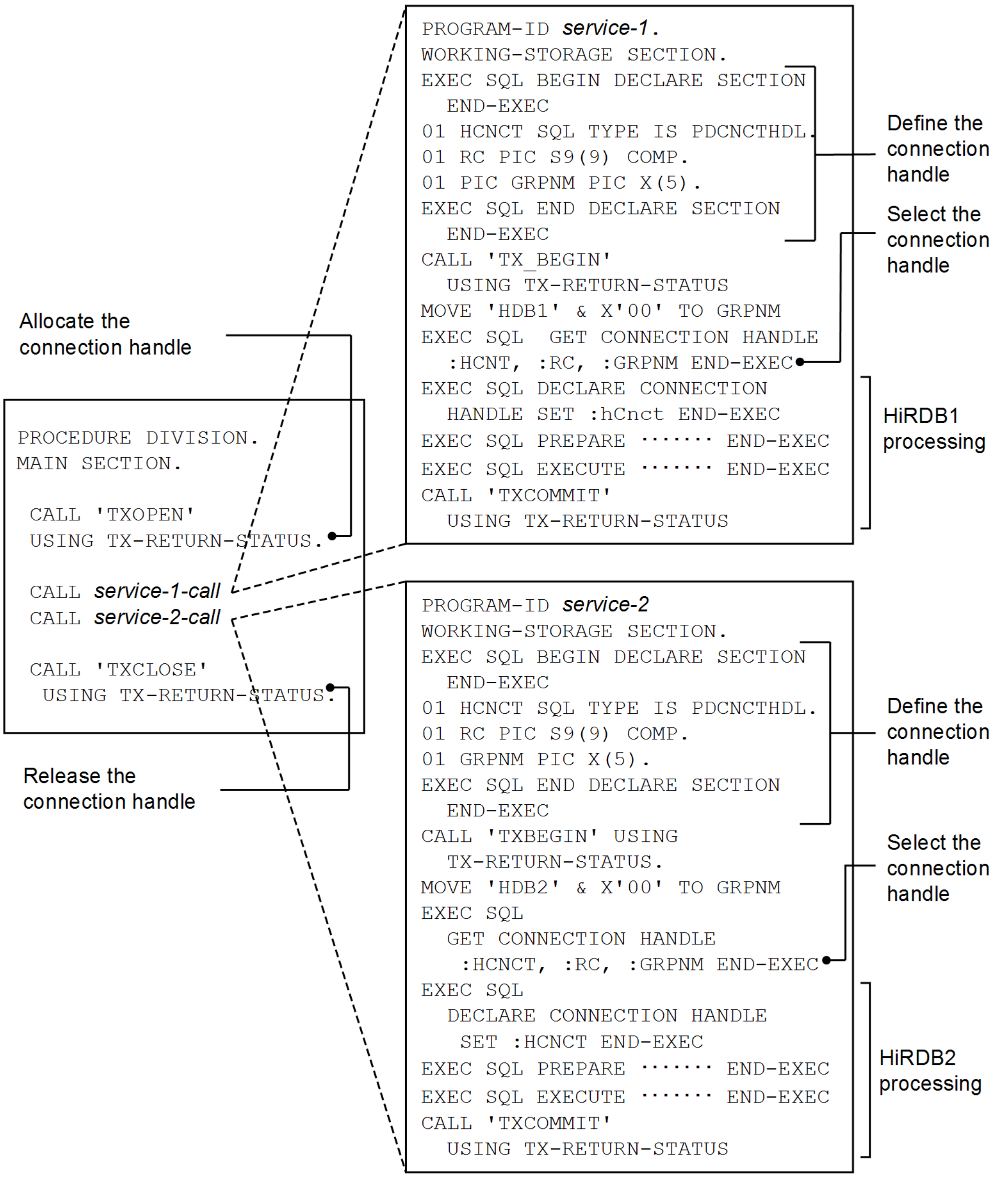
Allocates the connection handle.
Releases the connection handle.
Defines the connection handle.
Selects the connection handle.
Specifies the processing for HiRDB1.
Defines the connection handle.
Selects the connection handle.
Specifies the processing for HiRDB2.

Explanation

Register HiRDB 1 (environment variable group identifier HDB1) and HiRDB 2 (environment variable group identifier HDB2) in the OLTP system beforehand.

For details about how to register a HiRDB system to a transaction manager, see the *HiRDB Version 9 Installation and Design Guide*.

Figure 4-67: Coding example (COBOL) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP



```
PROCEDURE DIVISION.
MAIN SECTION.

 CALL 'TXOPEN'
 USING TX-RETURN-STATUS.

 CALL service-1-call
 CALL service-2-call

 CALL 'TXCLOSE'
  USING TX-RETURN-STATUS.
```

```
PROGRAM-ID service-1.
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
  END-EXEC
01 HCNCT SQL TYPE IS PDCNCTHDL.
01 RC PIC S9(9) COMP.
01 PIC GRPNM PIC X(5).
EXEC SQL END DECLARE SECTION
  END-EXEC
CALL 'TX_BEGIN'
  USING TX-RETURN-STATUS
MOVE 'HDB1' & X'00' TO GRPNM
EXEC SQL  GET CONNECTION HANDLE
  :HCNT, :RC, :GRPNM END-EXEC
EXEC SQL DECLARE CONNECTION
  HANDLE SET :hCnct END-EXEC
EXEC SQL PREPARE ······· END-EXEC
EXEC SQL EXECUTE ······· END-EXEC
CALL 'TXCOMMIT'
  USING TX-RETURN-STATUS
```

```
PROGRAM-ID service-2
WORKING-STORAGE SECTION.
EXEC SQL BEGIN DECLARE SECTION
  END-EXEC
01 HCNCT SQL TYPE IS PDCNCTHDL.
01 RC PIC S9(9) COMP.
01 GRPNM PIC X(5).
EXEC SQL END DECLARE SECTION
  END-EXEC
CALL 'TXBEGIN' USING
  TX-RETURN-STATUS.
MOVE 'HDB2' & X'00' TO GRPNM
EXEC SQL
  GET CONNECTION HANDLE
   :HCNCT, :RC, :GRPNM END-EXEC
EXEC SQL
  DECLARE CONNECTION HANDLE
   SET :HCNCT END-EXEC
EXEC SQL PREPARE ······· END-EXEC
EXEC SQL EXECUTE ······· END-EXEC
CALL 'TXCOMMIT'
  USING TX-RETURN-STATUS
```

Allocate the connection handle
Release the connection handle
Define the connection handle
Select the connection handle
HiRDB1 processing
Define the connection handle
Select the connection handle
HiRDB2 processing

Note

Specify the entire SQL, including the SQL prefix and terminator, in the B area (columns 12 to 72).

Explanation

Register HiRDB 1 (environment variable group identifier HDB1) and HiRDB 2 (environment variable group identifier HDB2) in the OLTP system beforehand. For details about how to register a HiRDB system to a transaction manager, see the *HiRDB Version 9 Installation and Design Guide*.

(4) Rules

1. If a UAP is to use the multi-connection facility, a special library must be linked to that UAP. For details, see *8.3.4 Compiling and linking when the multi-connection facility is used*.
2. If a UAP that uses the multi-connection facility library branches a thread while maintaining a single connection, and that thread executes SQL statements, serialize the processing between that thread and the other threads that issue SQL statements. SQL statements for the same connection cannot be issued simultaneously. However, SQL statements for different connections can be issued simultaneously.
3. To obtain error information for `ALLOCATE CONNECTION HANDLE` or `FREE CONNECTION HANDLE`, reference the value of the return code receiving variable instead of `SQLCODE` and `SQLERRM`. For details about the return code receiving variable, see the *HiRDB Version 9 SQL Reference* manual.
4. To reference an SQL Communications Area, the UAP must use the `DECLARE CONNECTION HANDLE SET` statement to declare a connection handle to the SQL Communications Area to be referenced.
5. When the programming language is COBOL, UAPs containing SQL statements that use the multi-connection facility cannot specify SQL statements except those for connection handle allocation and fetching before the `DECLARE CONNECTION HANDLE SET` specification (outside the effective scope).
6. When the programming language is COBOL, `DECLARE CONNECTION HANDLE UNSET` cannot be used.
7. The multi-connection facility can be used by UAPs that support multiple threads (DCE threads or real threads) or a single thread. To create a UAP that supports multi-threads and uses the multi-connection facility, you need to know how to develop a UAP in HiRDB and how to develop a UAP that uses DCE threads or real threads.
8. If an X/Open XA interface is not used, the multi-connection function in Windows can be used only by UAPs that support multi-threads. Therefore, when specifying the C runtime library to be used in UAP compilation using Visual Studio, select the multi-thread DLL (specify **Multithread DLL** in **Compile option: Code**

generation).

9. If an X/Open XA interface is used, the multi-connection function in Windows can also be used by UAPs that support only single threads. For UAPs that support single threads, when specifying the C runtime library to be used in UAP compilation using Visual Studio, also select the multi-thread DLL (specify **Multithread DLL** in **Compile option: Code generation**).
10. When using C or C++ to reference the SQL Communications Area, use and reference macro names that begin with SQL. Do not reference the SQLCA structures directly. For details about the macro names to be used, see *B.2(1)(a) C*.

4.11 Narrowed search

4.11.1 What is a narrowed search?

A *narrowed search* refers to a search that limits the target records in stages.

When a narrowed search is executed, lists are created with the `ASSIGN LIST` statement of the data manipulation SQL. The lists are used in information searches that specify conditions and limit the data items in stages until the appropriate number of data items is reached. These lists are intermediate-stage data sets that are temporarily saved with a name (list name) or data sets that are saved.

If a list is created for a certain condition, using that list can increase the processing speed. When several conditions are specified, a search that combines several lists can be executed.

4.11.2 Preparations for executing a narrowed search

Before executing a narrowed search, perform the following preparations:

- Specify the system definition
- Create an RDAREA for lists

You can execute a narrowed search (create lists) after you specify the system definitions and create an RDAREA for lists.

(1) Specifying the system definition

Before executing a narrowed search, specify the operands for the narrowed search in the system definition. The following operands must be specified before a narrowed search can be executed:

- `pd_max_list_users` (maximum number of users who can create lists)
- `pd_max_list_count` (maximum number of lists that each user can create)

In addition, the following operands can be specified if necessary:

- `pd_max_list_users_wrn_pnt` (output timing of the usage rate warning message for the specified `pd_max_list_users` value)
- `pd_max_list_count_wrn_pnt` (output timing of the usage rate warning message for the specified `pd_max_list_count` value)
- `pd_rdarea_list_no_wrn_pnt` (output timing of the usage rate warning message for the maximum number of lists that can be created in the server)

For details about these system definition operands, see the *HiRDB Version 9 System Definition* manual.

(2) Creating an RDAREA for lists

To create an RDAREA for lists, use the database initialization utility (`pdinit`) or the database structure modification utility (`pdmod`). For details about the database initialization utility and the database structure modification utility, see the *HiRDB Version 9 Command Reference* manual.

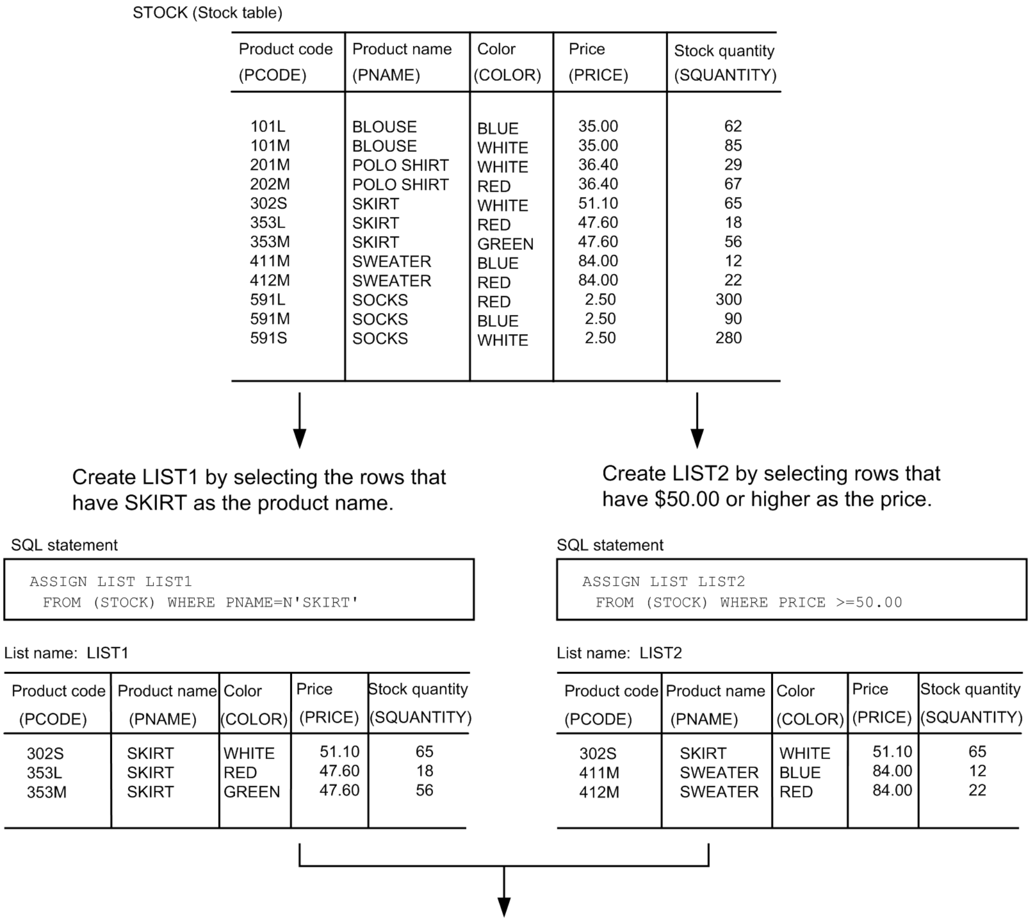
For the HiRDB file system area to be specified in the RDAREA for lists, specify `WORK` as the usage purpose. For details about how to design the RDAREA for lists, see the *HiRDB Version 9 Installation and Design Guide*.

4.11.3 Search using lists

This section explains the method of searching using lists.

The following figure shows an example of a search that uses a list.

Figure 4-68: Example of a search that uses lists



STOCK (Stock table)

Product code (PCODE)	Product name (PNAME)	Color (COLOR)	Price (PRICE)	Stock quantity (SQUANTITY)
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLO SHIRT	WHITE	36.40	29
202M	POLO SHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22
591L	SOCKS	RED	2.50	300
591M	SOCKS	BLUE	2.50	90
591S	SOCKS	WHITE	2.50	280

Create LIST1 by selecting the rows that have SKIRT as the product name.

SQL statement

```
ASSIGN LIST LIST1
  FROM (STOCK) WHERE PNAME=N'SKIRT'
```

List name: LIST1

Product code (PCODE)	Product name (PNAME)	Color (COLOR)	Price (PRICE)	Stock quantity (SQUANTITY)
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

Create LIST2 by selecting rows that have $50.00 or higher as the price.

SQL statement

```
ASSIGN LIST LIST2
  FROM (STOCK) WHERE PRICE >=50.00
```

List name: LIST2

Product code (PCODE)	Product name (PNAME)	Color (COLOR)	Price (PRICE)	Stock quantity (SQUANTITY)
302S	SKIRT	WHITE	51.10	65
411M	SWEATER	BLUE	84.00	12
412M	SWEATER	RED	84.00	22

Create LIST3 by determining the set product of LIST1 and LIST2. (Find the rows that have SKIRT as the product name and a value of $50.00 or higher as the price.)

SQL statement

```
ASSIGN LIST LIST3 FROM LIST1 AND LIST2
```

List name: LIST3

Product code (PCODE)	Product name (PNAME)	Color (COLOR)	Price (PRICE)	Stock quantity (SQUANTITY)
302S	SKIRT	WHITE	51.10	65

To search for data that has SKIRT as the product name and a value of $50.00 or higher as the price from the stock table (STOCK), search the LIST3 list. Processing is faster by searching LIST3 rather than specifying conditions and searching the stock table.

SQL statement

```
SELECT * FROM LIST LIST3
```

Product code (PCODE)	Product name (PNAME)	Color (COLOR)	Price (PRICE)	Stock quantity (SQUANTITY)
302S	SKIRT	WHITE	51.10	65

4.11.4 Action if a rollback occurs for a transaction that uses a list

If a transaction is cancelled by the `ROLLBACK` statement of SQL or an error, you many need to re-create a list that was created or deleted by that transaction. The following table describes the user action to be taken depending on the status of a list that had been created or deleted when a transaction was cancelled.

List operation in cancelled transaction		List status	User action
List created with `ASSIGN LIST` statement in the transaction	If the list was created with a list name that did not exist before the transaction was started	The list that was created cannot be found.	Reexecute the transaction process.
	If the list was created with a list name that existed before the transaction was started	The list that had the same list name before the transaction was started cannot be used. (An error occurs if the list is searched.)#	To use the list that had the same list name before the transaction was started in the transaction, re-create the list. Then reexecute the transaction.

List operation in cancelled transaction		List status	User action
List to be deleted by DROP LIST statement in the transaction	If the deletion-target list did not exist before the transaction was started	The list that was deleted cannot be found.	Reexecute the transaction process.
	If the deletion-target list existed before the transaction was started	The list that did not exist before the transaction was started cannot be found. The deletion-target list that existed before the transaction was started cannot be used. (An error occurs if the list is searched.)#	To use the deletion-target list that existed before the transaction was started in the transaction, re-create that list. Then reexecute the transaction.

#: Depending on when the transaction was cancelled, you still may be able to use the list normally.

4.11.5 Automatic list deletion at HiRDB startup and termination

When HiRDB is terminated or started, all lists that have been created are deleted regardless of the start mode.

If a HiRDB/Parallel Server is being used, and a single unit is terminated or started, all lists in the RDAREA for lists in that unit are deleted. If a single server is terminated or started, all lists in the RDAREA for lists in that server are deleted. When a deleted list is searched, an error occurs.

If HiRDB terminates abnormally in a unit or if all the units that configure the HiRDB system are stopped, all created lists are deleted when HiRDB is started. If some of the units terminate abnormally and those units are restarted, all lists in the RDAREA for lists in those units are deleted. When a deleted list is searched, an error occurs.

If such an error occurs, use one of the following methods to delete or re-create the list.

If you want to use the list that was affected by the search error

Use the ASSIGN LIST statement to create a list with the same list name that was used previously.

If you do not want to use the list that was affected by the search error

Use the DROP LIST statement to delete the list that resulted in the search error, or terminate and restart HiRDB to delete all created lists.

4.11.6 Notes about using lists

(1) List after disconnection from HiRDB

A list is not deleted even after the UAP is disconnected from the HiRDB system. To delete a list, either use the DROP LIST statement or stop the HiRDB system to delete all lists.

(2) List status after row insertion or deletion

In a search that uses a list, rows that were present in the list when the list was created but then later deleted are not searched. If a row is updated after the list is created, the updated data is fetched.

(3) Row insertion and deletion after list creation

In a search that uses a list, rows that were inserted after the list was created and after rows in the base table were deleted are sometimes searched.

(4) Execution of the ASSIGN LIST statement for a row partitioned table

If the `ASSIGN LIST` statement is executed for a row partitioned table and the table cannot be searched because of shutdown of some of the RDAREAs in the base table, an error occurs even if the data of an RDAREA that can be searched is specified in a search condition for a partitioned column.

(5) List operation by the same user

The same user cannot connect to multiple HiRDB systems simultaneously and operate a list.

(6) Stopping of the dictionary server or a unit found in the dictionary server

With a HiRDB/Parallel Server, if the dictionary server or the unit that contains the dictionary server is stopped, the list management information is lost. As a result, operations (search, deletion, and update) become disabled for all lists that were created up to that point. (An error occurs if a list is operated.) To use a list that triggered an error when operated, use the `ASSIGN LIST` statement to create a new list that has the same list name as the previous list.

If the dictionary server is restarted, the `KFPA11998-E` error (list operation while transaction is undetermined) may be displayed for processes that use a list. This error may be displayed until recovery is completed for all list-using transactions of other users that were started before the server was stopped.

(7) Recovery of a list base table with the database recovery utility

If a log is used to recover a list base table to its latest status, the lists that were created can be used without modification. However, for a recovery that uses only a backup, a time-specification recovery that uses a log, or a recovery that does not use the latest log, use one of the following methods to delete or re-create all lists that were based on the recovered table:

If you want to use the lists:

Use the `ASSIGN LIST` statement to create lists that have the same list names as the previous lists.

If you do not want to use the lists:

Use the DROP LIST statement to delete the lists, or terminate and restart HiRDB to delete all created lists.

(8) Reinitialization of an RDAREA where a list base table is stored

Use one of the following methods to delete or re-create all lists that are based on a list stored in a reinitialized RDAREA:

If you want to use the lists:

Use the ASSIGN LIST statement to create lists that have the same list names as the previous lists.

If you do not want to use the lists:

Use the DROP LIST statement to delete the lists, or terminate and restart HiRDB to delete all created lists.

(9) Execution of reorganization, creation mode download, or the PURGE TABLE statement on a list base table

Executing reorganization, creation mode download, or the PURGE TABLE statement on a list base table invalidates previously obtained search results for lists that were created based on that table. To use the lists, you must use the ASSIGN LIST statement to re-create the lists.

(10) Narrowed search when the inner replica facility is used

When you use the inner replica facility and also use the pddbchg command, PDDBACCS in the UAP environment definitions, or PDDBACCS in the client environment definitions to switch the RDAREA to be accessed, the search results become invalid unless one of the following conditions is satisfied:

- The RDAREA to be accessed during list retrieval matches the RDAREA to be accessed during list creation.
- The RDAREA to be accessed during list retrieval contains data that was copied from the RDAREA to be accessed during list creation.

To use a list, perform one of the following:

- Use the RDAREA to be accessed during list creation.
- Use the access-target RDAREA to which data was copied from the RDAREA to be accessed during list creation.
- Re-create the list in the RDAREA that is currently being accessed.

4.12 File output facility for BLOB data

4.12.1 What is the file output facility for BLOB data?

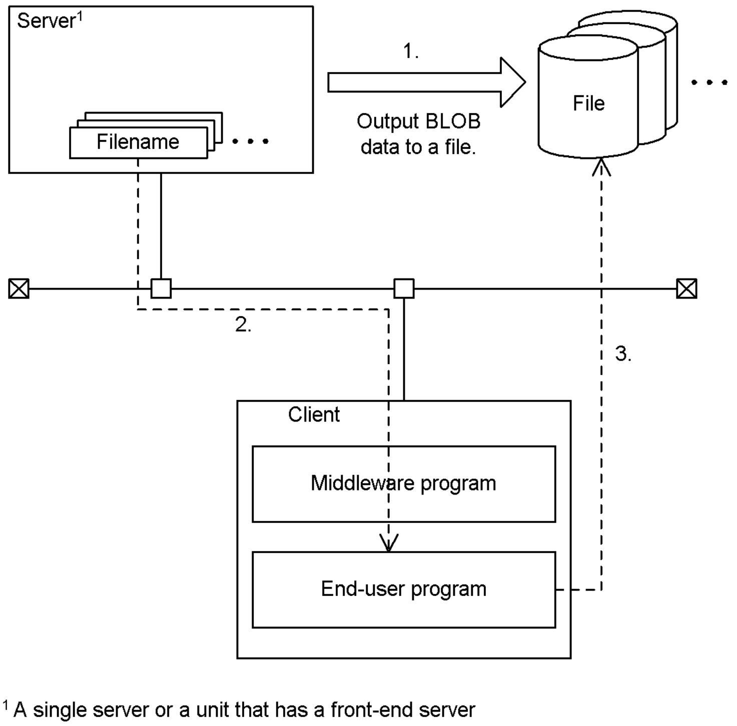
Before you can search BLOB data, you must prepare a memory area for storing BLOB data in the client. You will also need a send buffer for BLOB data returns in the server and memory for a receive buffer that accepts BLOB data in the client library. Consequently, a large amount of memory must be allocated according to the BLOB data size, and the memory resources will be strained.

More and more systems are configured so that a middleware program that operates as a HiRDB client is placed between the end user program and HiRDB. This configuration design has further increased the amount of memory used as BLOB data is transferred between these programs.

The file output facility for BLOB data prevents increased memory usage during BLOB data searches by outputting retrieved BLOB data directly to a file in a single server or a unit with a front-end server, instead of returning the BLOB data to the client. The facility then returns the name of the file to the client.

The following figure provides an overview of the file output facility for BLOB data.

Figure 4-69: Overview of the file output facility for BLOB data


Server[1]
Filename
• • •
1.
Output BLOB data to a file.
File
• • •
2.
3.
Client
Middleware program
End-user program
[1] A single server or a unit that has a front-end server

Explanation

1. When the client searches BLOB data, the server outputs that BLOB data in single rows and single columns to a file.
2. The server returns the file name of the BLOB data that was output in (1) to the client.
3. Based on the file name that was returned, the client accesses the BLOB data file located in the server.

4.12.2 Application criteria

Apply the file output facility for BLOB data to reduce the amount of memory required during BLOB data search.

This facility is effective in reducing the memory size required for client programs and the memory size required for the communication buffer used in server-client communication. However, applying this facility also increases the disk input/output operations that take place during file output. Therefore, be sure to consider both the required memory size effects and the disk input/output effects before you use the file output facility for BLOB data.

4.12.3 Specification method

Specify the file output facility for BLOB data in a WRITE specification of SQL. The WRITE specification can be specified in a cursor specification or a query specification.

For details about the WRITE specification, see the *HiRDB Version 9 SQL Reference* manual.

4.12.4 Notes about using the file output facility for BLOB data

1. When a BLOB data file that was created becomes unnecessary, the user must delete that file. Note the following point about deleting BLOB data files. Also, BLOB data files can be deleted unconditionally after cursor close or transaction resolution.
    - When deleting a BLOB data file immediately after FETCH processing, and the FETCH result prior to the same cursor search and the BLOB value are the same, there are cases in which the file is not re-created with the same file name. In this case, control the processing by storing the prior file name and then deleting it when the file name changes.
2. Created BLOB data files are not deleted if an error or a rollback occurs. Note that if the BLOB data files are not deleted, they use up disk space and operating system resources.
3. Check that there is enough disk space available before using the following facilities:
    - FETCH facility using arrays

      Each time FETCH is executed, a file is created for each array element.
    - Block transfer facility

      During the first FETCH is executed, a file is created for each block transfer row. Subsequently, each time the FETCH for all block transfer rows is completed and the next FETCH is executed, the file creation for each block transfer row is repeated.
4. If a file name is the same as the file name of another transaction or cursor search, the files may destroy one another. To avoid this problem, for each transaction or cursor, change the directory or file name in the file prefix so that file names are not duplicated.

4.12.5 Examples of using the file output facility for BLOB data

This section shows search examples in which the file output facility for BLOB data is used.

(1) Retrieving BLOB columns

In the following example, columns C1 and C2 are searched from table T1. The BLOB data in column C1 is output to files, and the names of those files are obtained.

Table T1

C1	C2
BLOB value 1	10
BLOB value 2	20
BLOB value 3	30
BLOB value 4	40

SQL statement

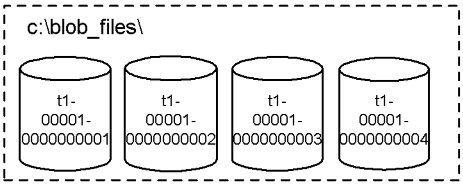
```
SELECT WRITE(C1,'c:\blob_files\t1',0),C2 FROM T1
```

Retrieval results

C1	C2
172.16.202.5:c:\blob_files\t1-00001-0000000001	10
172.16.202.5:c:\blob_files\t1-00001-0000000002	20
172.16.202.5:c:\blob_files\t1-00001-0000000003	30
172.16.202.5:c:\blob_files\t1-00001-0000000004	40

BLOB data that is output to the server

- IP address: 172.16.202.5



(2) Retrieving an abstract data type that has the BLOB attribute

In the following example, the ADT1 column in which CONTAINS() is true is searched from table T2. At this time, the BLOB values of the results for passing the columns values to the EXTRACTS() argument are output to a file, and the file name is obtained. This example shows the case when all values are hit.

Table T2

ADT1
Abstract data-type value 1
Abstract data-type value 2
Abstract data-type value 3
Abstract data-type value 4

SQL statement

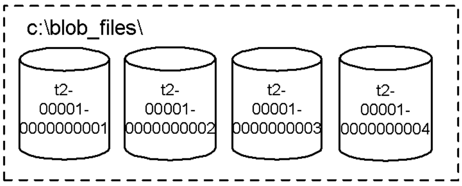
```
SELECT WRITE(EXTRACTS(ADT1,···),'c:\blob_files\t2',0) FROM T2
   WHERE CONTAINS(ADT1,···) IS TRUE
```

Retrieval results

ADT1
172.16.202.5:c:\blob_files\t2-00001-0000000001
172.16.202.5:c:\blob_files\t2-00001-0000000002
172.16.202.5:c:\blob_files\t2-00001-0000000003
172.16.202.5:c:\blob_files\t2-00001-0000000004

BLOB data that is output to the server

- IP address: 172.16.202.5



4.13 Partial update and retrieval of BLOB and BINARY data

4.13.1 About partial update and retrieval of BLOB and BINARY data

If all registered BLOB or BINARY[#] data must be updated when new data is added, or if all BLOB or BINARY[#] data must be fetched when data is retrieved, both the server and client must secure large amounts of memory that match the enormous data size. Consequently, the memory resources become used up. Partial update and retrieval of BLOB and BINARY data provides the following functions to solve this problem:

- Addition update of BLOB and BINARY data
- Partial extraction of BLOB and BINARY data
- Rear deletion update of BLOB and BINARY data

#: This refers to BINARY data that has a minimum defined length of 32,001 bytes.

(1) Addition update of BLOB or BINARY data

To add new data to registered BLOB or BINARY data, specify a concatenation operation in the SET clause of the UPDATE statement. The amount of memory used is suppressed to the amount of data to be added.

(2) Partial extraction of BLOB or BINARY data

To extract only the specified portion from BLOB or BINARY data, specify the SUBSTR scalar function. The amount of memory used is suppressed to the amount of data to be extracted.

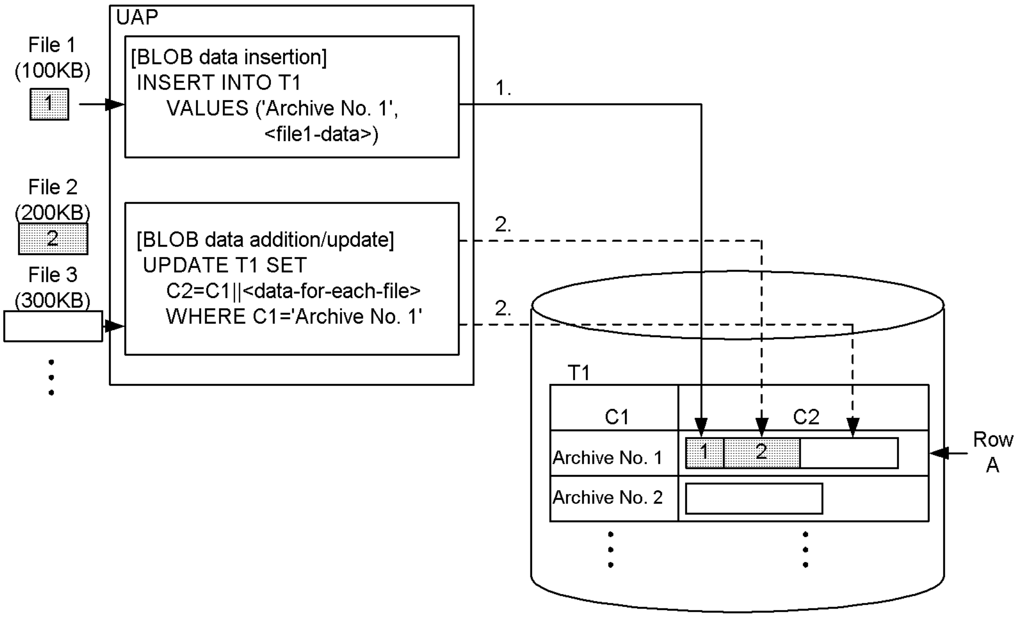
(3) Rear deletion update of BLOB and BINARY data

You can delete only the rear part of BLOB or BINARY data by using the SUBSTR scalar function that specifies the columns to be processed and the literal 1 as the start position in the SET clause in the UPDATE statement. This facility enables you to limit the amount of required memory and log space because data can be updated without having to acquire as much memory as would be needed for all data that is to be updated.

4.13.2 Examples of using the addition update and partial extraction facility for BLOB data

(1) Addition update of BLOB data

Multiple files are stored as one BLOB data element.

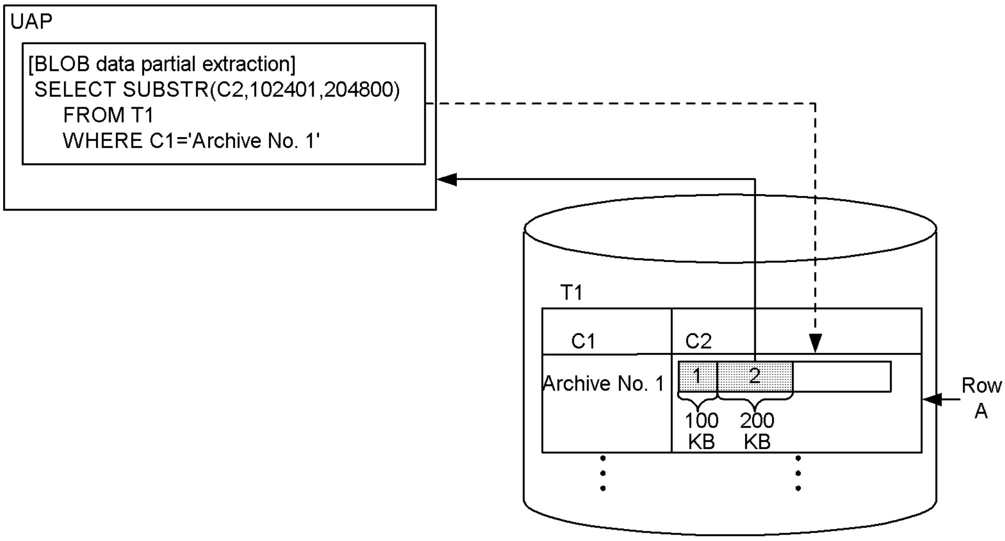


Explanation

1. The BLOB data of file 1 is inserted in column C2 of row A in the target table (T1).
2. The BLOB data of file 2 is added by concatenating the data to column C2 of row A. The same applies when subsequent data is added.

(2) Partial extraction of BLOB data

The BLOB data of file 2 is extracted from the BLOB data (column C2) in row A that was stored in *Addition update of BLOB data*.

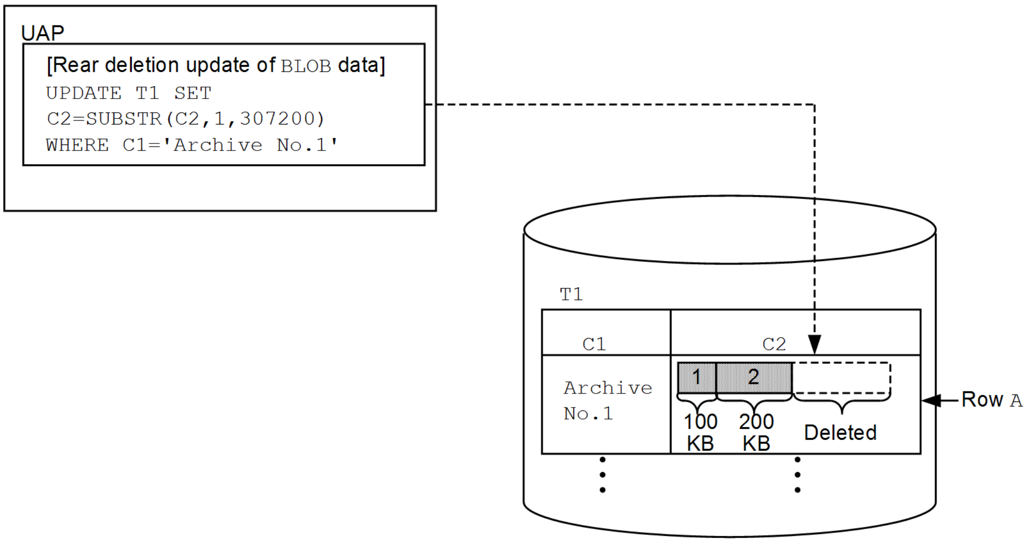


Explanation

The SUBSTR scalar function is used to extract data from the starting position (byte (100 x 1024 + 1) = byte 102401) of the data column for file 2. Only the amount of data equivalent to the length of the data column in file 2 (200 x 1024 = 204800 bytes) is extracted.

(3) Rear deletion update of BLOB data

The rear part of the BLOB data (column C2) in row A that was stored in *Addition update of BLOB data* is deleted and only files 1 and 2 are retained.



```
UPDATE T1 SET
C2=SUBSTR(C2,1,307200)
WHERE C1='Archive No.1'
```

Explanation:

The SUBSTR scalar function is used to replace data from the start position (byte 1) of the data column for file 1 with as much data as there is in files 1 and 2 (100 x 1024 + 200 x 1024 = 307200 bytes). As a result, only files 1 and 2 remain, and the rear part of the data is deleted.

4.13.3 Notes about performing partial updating and retrieval of BLOB and BINARY data

You should note the following about performing partial updating and retrieval of BLOB and BINARY data:

1. A concatenation operation for BLOB or BINARY data can be specified only with an update value in the SET clause of the UPDATE statement. The item specified for the first term in the concatenation operation must be a column specification, and the item specified for the second term must be an embedded variable, the ? parameter, an SQL variable, or an SQL parameter.

   For details about the rules for using the concatenation operation to update a BLOB-type or BINARY-type column, see the manual *HiRDB Version 9 SQL Reference*.

2. To execute an addition update, create a column that stores unique key values, and specify that column in the search conditions to identify the update row. To accelerate the row identification process, create an index in that column.

3. The minimum input/output unit for BLOB data is the page length of the RDAREA, and HiRDB performs batch input/output of up to 128 kilobytes. Therefore, to

improve the performance of BLOB data insertion, addition update, or partial extraction, you should set the data length to units of 128 x 1024 x *n* bytes (*n* is a nonzero positive integer).

4. If BACKWARD_CUTOFF_UPDATE is not specified in the pd_rpl_func_control operand in the system common definitions, rear deletion update is disabled. In such a case, the data specified in SUBSTR is extracted into memory and then is updated.

4.14 Retrieve first n records facility

4.14.1 Overview

Sometimes the SQL retrieval performance can be improved by obtaining the retrieval results of only the first *n* rows. The performance improvement can be expected to increase as the number of retrieval result rows decreases.

When the retrieve first *n* records facility is used, only the first *n* rows from the beginning of the SQL retrieval results (or after the specified offset of the first row to return has been skipped) are accepted. In this case, the access path selected by the SQL optimization method changes. Consequently, the SQL retrieval performance may improve as described as follows:

- Fewer rows may need to be sorted because sort processing that targets all rows that satisfy the search conditions becomes unnecessary.
- The work tables that HiRDB creates exclusively for the `ORDER BY` clause may become unnecessary.
- The amount of communication between server processes can sometimes be reduced by having the server processes not read the rows that do not fall in the first *n* rows of the retrieval results.

To use the retrieve first *n* records facility, specify `LIMIT`. For details about `LIMIT`, see the manual *HiRDB Version 9 SQL Reference*.

4.14.2 Notes

In the following cases, the retrieval performance may not improve, or conversely, may become worse, even if the retrieve first *n* records facility is used.

1. If the sum of the offset of the first row to return and the maximum number of rows to return is the same or extremely close to the value when the `LIMIT` clause is not specified.
2. If the `LIMIT` clause is specified but the `ORDER BY` clause is not, HiRDB cannot uniquely determine which rows are to be retrieved. The `ORDER BY` clause should therefore be specified whenever the `LIMIT` clause is specified. However, when the `ORDER BY` clause is specified, the SQL optimization method may select a different access path and the retrieval performance may worsen. To check the access path selected by the SQL optimization method, use the access path display utility (`pdvwopt`).
3. If both the `ORDER BY` and `LIMIT` clauses are specified and there are several rows that have the same sort key value as the last row that was skipped based on the offset of the first row to return or the last row that was obtained based on the maximum number of rows to return, HiRDB cannot uniquely determine which of

the rows with the same sort key value are to be retrieved. To retrieve a specific row that has the same sort key value as the row that satisfies this condition, add more columns to the sort key. However, when more sort key columns are added, the SQL optimization method may select a different access path, and the retrieval performance may worsen. To check the access path selected by the SQL optimization method, use the access path display utility (`pdvwopt`).

In cases like those described, do not use the retrieve first *n* records facility.

If the maximum number of rows to return is 1 or more and the sum of the offset of the first row to return and the maximum number of rows to return is 32,767 or less, HiRDB stores the rows that fall within that sum in memory instead of creating a work table. Therefore, the required memory size increases compared to when the facility is not used. For details about the required memory size, see *Calculating the required memory size for execution of the retrieve first n records facility* in the *HiRDB Version 9 Installation and Design Guide*.

4.14.3 Checking the access path

Depending on whether or not the retrieve first *n* records facility is used to accelerate retrieval processing, the SQL optimization method may select an access path that differs from the `ORDER BY` processing method. For details about the `ORDER BY` processing method, see the `pdvwopt` description in the manual *HiRDB Version 9 Command Reference*.

4.15 Automatic reconnect facility

The automatic reconnect facility automatically reconnects the HiRDB client to the HiRDB server if the connection with the HiRDB server is disconnected because of a service process failure, system switchover, network failure, or other cause. By using the automatic reconnect facility, you can continue UAP execution without worrying about disconnections with the HiRDB server.

To use the automatic reconnect facility, specify `YES` in the `PDAUTORECONNECT` client environment definition.

4.15.1 Application criteria

If the HiRDB server is executing any of the processes listed below, the HiRDB client waits until that process terminates.

- Changing the system definitions (executing the `pdchgconf` command)
- Updating the current HiRDB to the HiRDB update version (executing the `pdprgcopy` and `pdprgrenew` commands)
- Performing planned system switchover by using the transaction queuing facility (`pdtrnqing` command)

While the HiRDB client is waiting, the wait time is monitored based on the `PDCWAITTIME` time. If the `PDCWAITTIME` time is exceeded, the wait status is released and a `PDCWAITTIME` over error is returned to the UAP.

Depending on the execution timing, the HiRDB client might not be able to detect that the above process is underway, resulting in a communication error. If you know ahead of time that the above process is to be performed, evaluate the use of the automatic reconnect facility. If you use this facility, the HiRDB client can continue processing without returning an error to the UAP even if the above process is underway.

4.15.2 Reconnect timings

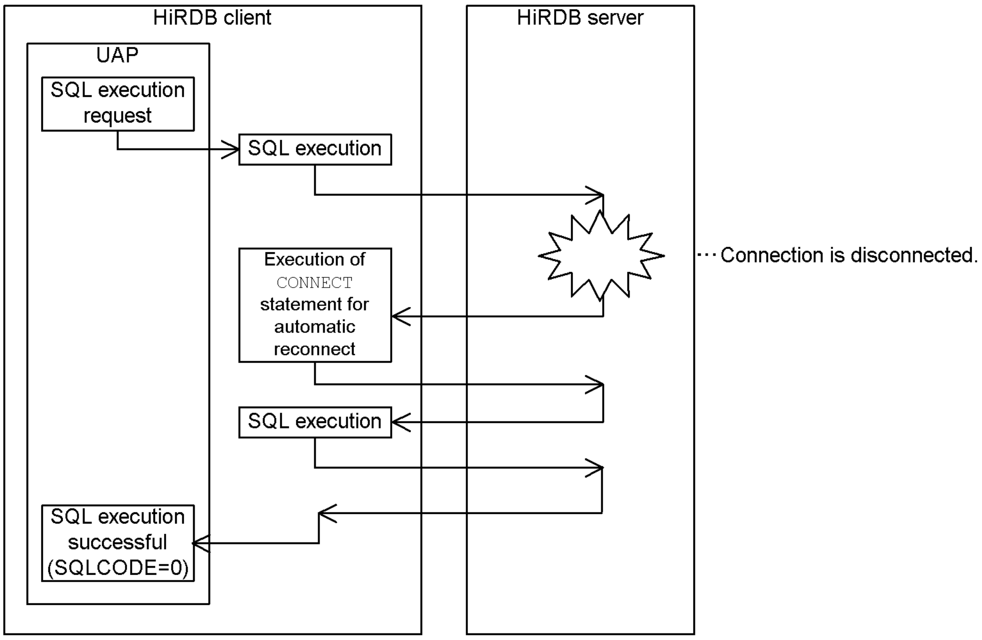
Reconnection is performed at the following times:

- When the HiRDB client executes an SQL statement immediately after executing the `CONNECT` statement, or when the transaction for the previous SQL statement is completed
- When the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement
- When the HiRDB client executes the `CONNECT` statement

(1) If the HiRDB client executes an SQL statement immediately after executing the CONNECT statement, or when the transaction for the previous SQL statement is completed

When the HiRDB client executes an SQL statement, the automatic reconnect facility detects whether the connection with the HiRDB server has been disconnected. If the facility detects a disconnection, it reconnects the client to the server, and re-executes the SQL statement after the connection is re-established. If the automatic reconnect facility detects a connection failure when the HiRDB client executes an SQL statement after automatic reconnection, it returns an error to the UAP. The following figure shows the reconnect timing (when the HiRDB client executes an SQL statement immediately after executing the `CONNECT` statement, or when the transaction for the previous SQL statement is completed).

Figure 4-70: Reconnect timing (when the HiRDB client executes an SQL statement immediately after executing the CONNECT statement, or when the transaction for the previous SQL statement is completed)

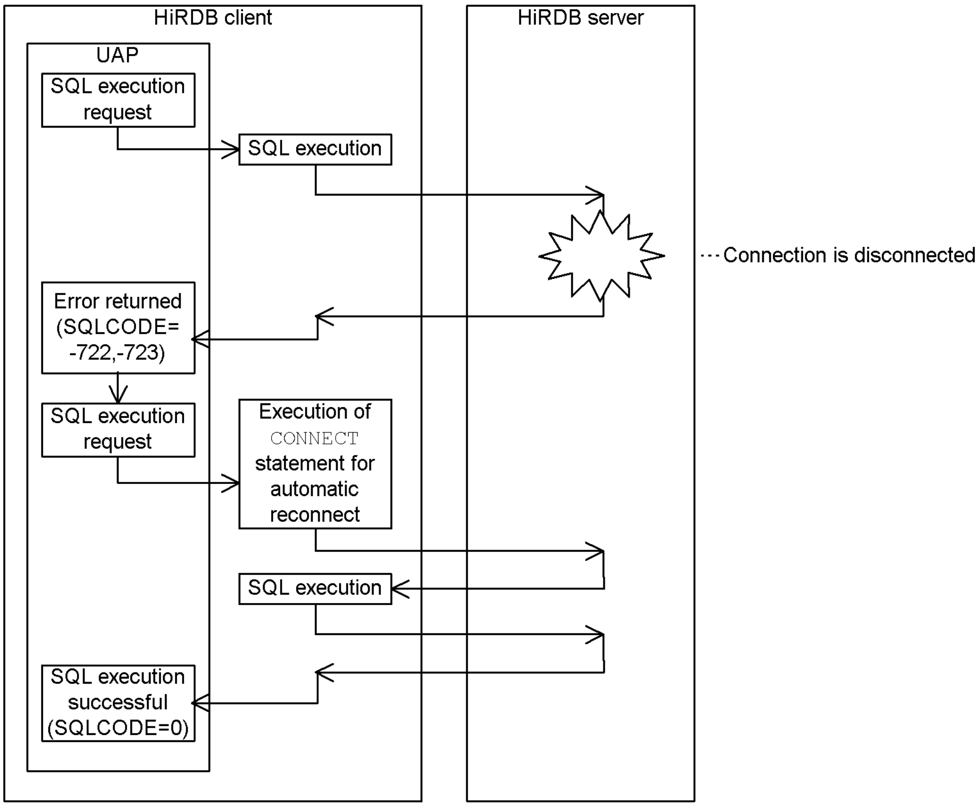

HiRDB client
HiRDB server
UAP
SQL execution request
SQL execution
Connection is disconnected.
Execution of CONNECT statement for automatic reconnect
SQL execution
SQL execution successful (SQLCODE=0)

(2) When the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement

When the HiRDB client executes an SQL statement, the automatic reconnect facility detects whether the connection with the HiRDB server has been disconnected. If the facility detects a disconnection, it returns a connection error (`SQLCODE` = `-722` or `-723`) to the UAP. When the client executes the next SQL statement, the facility reconnects with the server and re-executes the previous SQL statement.

If the automatic reconnect facility detects a connection failure when the HiRDB client executes an SQL statement after automatic reconnection, it returns an error to the UAP. The figure below shows the reconnect timing (when the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement). Any uncompleted transactions that were being executed when the SQL statement with the returned error was executed are rolled back.

Figure 4-71: Reconnect timing (when the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement)


HiRDB client
HiRDB server
UAP
SQL execution request
SQL execution
···Connection is disconnected
Error returned (SQLCODE= -722,-723)
SQL execution request
Execution of CONNECT statement for automatic reconnect
SQL execution
SQL execution successful (SQLCODE=0)

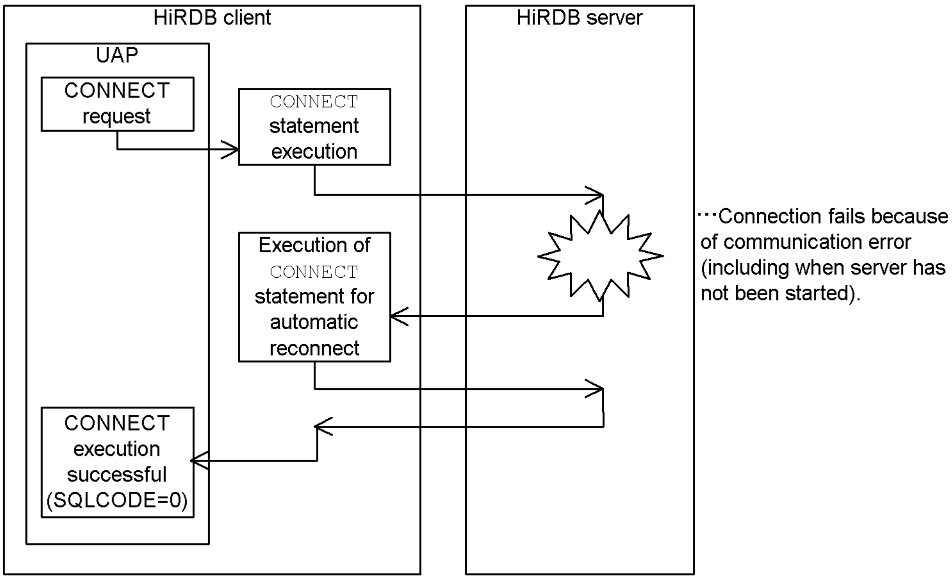
(3) When the HiRDB client executes the CONNECT statement

If the HiRDB client executes the `CONNECT` statement and the connection fails because of a communication error, the automatic reconnect facility executes reconnect processing.

The following figure shows the reconnect timing (when the HiRDB client executes the

CONNECT statement).

Figure 4-72: Reconnect timing (when the HiRDB client executes the CONNECT statement)


HiRDB client
UAP
CONNECT request
CONNECT statement execution
Execution of CONNECT statement for automatic reconnect
CONNECT execution successful (SQLCODE=0)
HiRDB server
...Connection fails because of communication error (including when server has not been started).

4.15.3 CONNECT processing during automatic reconnect

The automatic reconnect facility executes the CONNECT statement five times internally at 5-second intervals. You can use PDRCCOUNT and PDRCINTERVAL to change the number of times the CONNECT statement is executed, and the execution interval, respectively.

If the reconnect timing is the CONNECT statement, HiRDB uses the PDCONNECTWAITTIME value to monitor the processing time. If it is not the CONNECT statement, HiRDB uses the PDCWAITTIME value. If the processing time for automatic reconnect exceeds the PDCONNECTWAITTIME or PDCWAITTIME value, HiRDB aborts the automatic reconnect process and returns an error to the UAP. If you wish to retry the automatic reconnect facility as many times as specified in PDRCCOUNT, specify at least the value of PDRCCOUNT x PDRCINTERVAL in PDCWAITTIME.

4.15.4 Notes about using the automatic reconnect facility

1. The automatic reconnect facility cannot be used if the UAP contains a LOCK statement that specifies UNTIL DISCONNECT.
2. If the UAP that uses a holdable cursor is being used, the automatic reconnect facility returns an error to the UAP, even if a transaction is not being processed.

3. If the JDBC driver[#1] or DABroker for JAVA[#2] accesses the system and a statement that applies over several transactions is effective, the JDBC statement becomes ineffective after the automatic reconnect facility reconnects the HiRDB client. In this case, the `prepareStatement()` method must be executed again.

4. If the time specified in `PDCWAITTIME` is reached by the first SQL statement of the transaction, HiRDB re-executes the `CONNECT` statement and SQL statement without returning an error to the UAP. Therefore, HiRDB might return an error when the amount of time equivalent to about twice the `PDCWAITTIME` value is reached.

5. If Cosminexus is connected and DB Connector's statement pooling facility is enabled, an `SQLException` exception including the `KFPA11901-E` message might occur when an SQL statement is executed after connection has been re-established by HiRDB's automatic reconnect facility. If you use the statement pooling facility, do not use the automatic reconnect facility. For details about the statement pooling facility, see the *Cosminexus Application Setup Guide*.

   #1: When the JDBC driver is used, a statement that applies over several transactions becomes effective when `"CLOSE"` or `"RESERVE"` is set to `COMMIT_BEHAVIOR. COMMIT_BEHAVIOR` can be set with the `Properties info` argument of the `connect` method in the `Driver` class, the `Properties info` argument of the `DriverManager.getConnection` method, or the `COMMIT_BEHAVIOR` key in a URL connection.

   #2: When DABroker for JAVA is used, a statement that applies over several transactions becomes effective when the DABroker version is 03-06 or later and the DABroker for JAVA version is 02-10 or later.

4.16 Locator facility

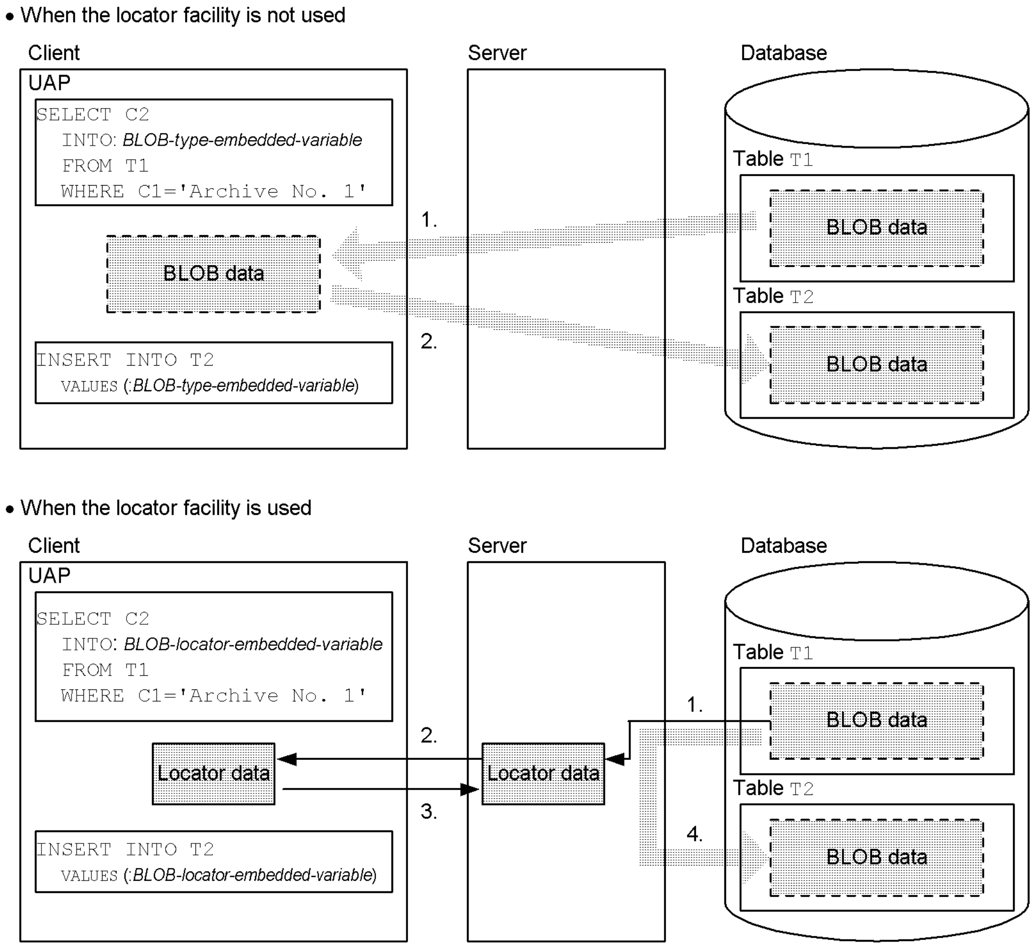
4.16.1 What is the locator facility?

For the client UAP to accept retrieved BLOB or BINARY data in embedded variables of that data type, the client must have a memory area available for storing the data. Therefore, the memory resources of the client become overburdened when large object data is retrieved. Furthermore, the amount of data transferred from the server to the client becomes large. However, if only a portion of that data is required or if the accepted data is simply specified unchanged into another SQL statement and returned to the server, transferring all the data to the client makes processing ineffective.

HiRDB provides a locator facility to resolve this problem. A locator is a 4-byte value that identifies data on the server. When a locator embedded variable is specified in the INTO clause of a FETCH or single-row SELECT statement, a locator value that identifies that data is received as the search result instead of the actual data entity. Also, by specifying the locator embedded variable identifying the data into another SQL statement, you can execute a process that handles the data identified by the locator.

The following figure provides an overview of the locator facility.

Figure 4-73: Overview of the locator facility



```
SELECT C2
  INTO: BLOB-type-embedded-variable
  FROM T1
  WHERE C1='Archive No. 1'
```

```
INSERT INTO T2
  VALUES (:BLOB-type-embedded-variable)
```

```
SELECT C2
  INTO: BLOB-locator-embedded-variable
  FROM T1
  WHERE C1='Archive No. 1'
```

```
INSERT INTO T2
  VALUES (:BLOB-locator-embedded-variable)
```

Explanation

When the locator facility is not used:

1. The server transfers the BLOB data retrieved from the database to the client.
2. The client transfers the BLOB data to the server for storage in the database.

When the locator facility is used:

1. The server creates locator data that identifies the data retrieved from the database.
2. The server transfers the locator data to the client.
3. The client transfers the locator data to the server.

4. The server stores the BLOB data identified by the locator data in the database.

4.16.2 Application standard

Apply the locator facility when you are retrieving BLOB or BINARY data and you want to reduce the amount of memory required in the client or decrease the amount of data transferred between the server and the client.

When the locator facility is used, memory for the actual data size does not need to be allocated in the client. In addition, the amount of transferred data can be decreased because a locator can be used for transferring data between the server and the client.

4.16.3 Usage method

To accept a locator value, specify a locator embedded variable of the corresponding data type at the location where the embedded variable for accepting the BLOB-type or BINARY-type data is specified in the SQL statement. To process the data assigned to the locator, specify a locator embedded variable of the corresponding data type instead of specifying a BLOB-type or BINARY-type embedded variable in the SQL statement.

4.16.4 Usage example

This example replaces only the first 400 kilobytes starting from a certain binary data column (search_data) of the data in column C2 of row C1=1 in table T1 with other data (change_data). The result is inserted into table T1 in column C2 of a new row (C1=2).

The data types of the columns in table T1 are shown below:

- C1: INTEGER NOT NULL (INDEX)
- C2: BLOB (100M) NOT NULL

```
void abnormalend(void);

main()
{
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB AS LOCATOR alldata_loc; /* Locator representing
all data */
    long change_pos;                          /* Change start position */
    SQL TYPE IS BLOB(10) search_data;          /* Binary data column to
be searched */
    SQL TYPE IS BLOB(400K) change_data;        /* Binary data column to
be changed */
    SQL TYPE IS BLOB AS LOCATOR enddata_loc; /* Locator representing
data */
                                   /* that follows section to be changed */
    long pos;
  EXEC SQL END DECLARE SECTION;
```

```
            -------(CONNECT process to HiRDB (omitted))          -------
            -------(Settings for binary data column to be searched (omitted))-------
            -------(Settings for binary data column to be changed (omitted))-------

  EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;
  /* Use locator to get column data */
  EXEC SQL SELECT C2 INTO :alldata_loc FROM T1 WHERE C1 = 1;
  /* Get start position that includes binary data to be searched */
  EXEC SQL SET :change_pos = POSITION(:search_data AS BLOB(10)
    IN :alldata_loc AS BLOB(100M));
  pos = change_pos + 409600;
  /* Use locator to get data that follows changed portion */
 EXEC SQL SET :enddata_loc = SUBSTR(:alldata_loc AS BLOB(100M),
:pos);
  pos = change_pos -1;
  /* Use locator to insert data in front of changed section */
  EXEC SQL INSERT INTO T1 VALUES(2, SUBSTR
    (:alldata_loc AS BLOB(100M), 1, :pos));
  /* Locator representing all data is nullified because it is no longer necessary */
  EXEC SQL FREE LOCATOR :alldata_loc;
  /* Link data of changed section and update */
  EXEC SQL UPDATE T1 SET C2 = C2 || :change_data WHERE C1 = 2;
  /* Use locator to link data that follows changed section and update */
  EXEC SQL UPDATE T1 SET C2 = C2 || :enddata_loc WHERE C1 = 2;
  EXEC SQL COMMIT;
  printf(" *** normally ended ***\n");
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;
  return(0);
}
void abnormalend()
{
  int  wsqlcode;
  wsqlcode = -SQLCODE;printf("\n*** HiRDB SQL ERROR SQLCODE
    = %d \n", wsqlcode);
  printf("SQLERRMC = %s\n", SQLERRMC);
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  exit(1);
}
```

(1) Note

1. For server data to be assigned to a locator, the server may need memory to store the data assigned to the locator. Therefore, if a single transaction assigns many data items to locators and keeps the locators valid, the server memory will be

overburdened. To prevent this problem, use the `FREE LOCATOR` statement to invalidate locators that are no longer necessary.

4.17 Facility for returning the total number of hits

4.17.1 Overview

Normally, you would have to execute two SQL statements to obtain the total number of hits and the values of the rows that were hit. However, by using the facility for returning the total number of hits, you can combine the SQL statement for obtaining the total number of hits and the SQL statement for obtaining the values of the hit rows into a single SQL statement. As a result, the retrieval time for executing the two SQL statements essentially becomes the same as the retrieval time for executing one SQL statement.

To use the facility for returning the total number of hits, specify the `COUNT(*) OVER()` window function in a selection expression. For details about window functions, see the manual *HiRDB Version 9 SQL Reference*.

4.17.2 Usage examples

Shown below are examples in which a count of the total number of products whose price (`PRICE`) is $50.00 or more is obtained from the stock table (`STOCK`), along with the product names of these products (`PNAME`), and then the resulting list of products is sorted by quantity (`SQUANTITY`).

- When the facility for returning the total number of hits is not used

```
SELECT COUNT(*) FROM STOCK WHERE PRICE>=5000
SELECT PNAME    FROM STOCK WHERE PRICE>=5000 ORDER BY
SQUANTITY
```

Explanation

When the facility for returning the total number of hits is not used, two SQL statements are required.

- When the facility for returning the total number of hits is used

```
SELECT COUNT(*) OVER(), PNAME
             FROM STOCK WHERE PRICE>=5000 ORDER BY SQUANTITY
```

Explanation

Because the underlined sections of the two SQL statements are the same, you can use the facility for returning the total number of hits in order to combine the two SQL statements into a single SQL statement, and so obtain the total number of hits in the first fetch operation.

4.17.3 Note

When you use the facility for returning the total number of hits in the cases listed below, improvement of retrieval performance cannot be expected, and in fact performance may even drop. If performance drops, do not use the facility for returning the total number of hits.

1. If `DISTINCT`, the `ORDER BY` clause, or the `FOR READ ONLY` clause is not specified
2. If the `ORDER BY` clause is specified and an access path that can cancel the sorting for `ORDER BY` is selected

   To check whether the sorting for `ORDER BY` can be canceled, use the access path display utility (`pdvwopt`). For details about the access path display utility, see the manual *HiRDB Version 9 Command Reference*.
3. If the projection length is short, and the communication volume increase attributed to the 4-byte string length of `COUNT(*) OVER()` cannot be ignored
4. If the retrieval processing cost is small

4.18 Retrieval, updating, or deletion with an RDAREA name specified

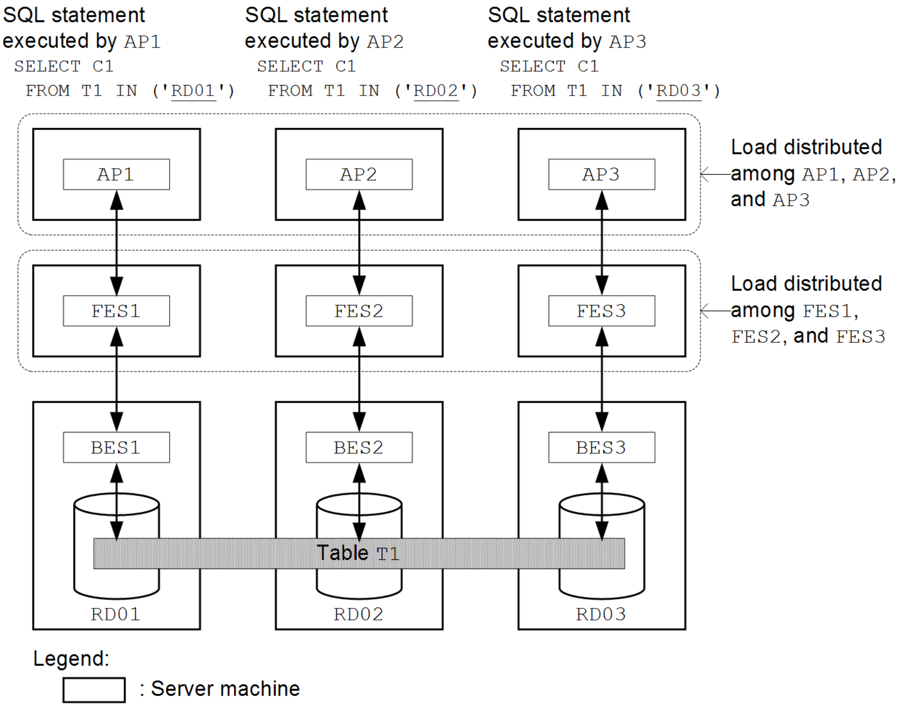
4.18.1 Overview of function

When you use row-partitioned tables on multiple front-end servers and perform retrieval, update, or deletion processing with RDAREA names specified, you can limit the RDAREAs to be accessed. This enables you to access multiple RDAREAs in parallel, thereby distributing the load among the server machines.

4.18.2 Example

This example performs retrieval processing with RDAREA names specified.

Figure 4-74: Example of retrieval processing with RDAREA names specified


SQL statement executed by AP1
SELECT C1
FROM T1 IN ('RD01')
SQL statement executed by AP2
SELECT C1
FROM T1 IN ('RD02')
SQL statement executed by AP3
SELECT C1
FROM T1 IN ('RD03')
AP1
AP2
AP3
Load distributed among AP1, AP2, and AP3
FES1
FES2
FES3
Load distributed among FES1, FES2, and FES3
BES1
BES2
BES3
Table T1
RD01
RD02
RD03
Legend:
: Server machine

Explanation:

This example performs retrieval processing on the row-partitioned table T1 that is stored in RDAREAs RD01, RD02, and RD03. Each of the three UAPs (AP1, AP2, and AP3) issues the SELECT statement specifying an RDAREA to be accessed. This method enables the RDAREAs to be accessed in parallel via the front-end servers FES1, FES2, and FES3, thereby distributing the load among the server machines where the UAP and front-end servers are located.

4.18.3 Notes

- This function is not applicable in the following cases:
    - Retrieval, update, or deletion processing is performed on a table for which WITHOUT ROLLBACK is specified in CREATE TABLE
    - The UAP is running in the no-log mode
    - The -i option in the pdlbuffer operand is used for local buffer allocation for indexes
- When this function is applied, only those indexes that have the same number of partitions as the table partitions are used. If you perform only retrieval, update, or deletion processing with an RDAREA name specified, and have defined an index that has a different number of partitions than the table partitions, that index is not used. If you perform retrieval, update, or deletion processing with an RDAREA name specified, define an index that has the same number of partitions as the table partitions.

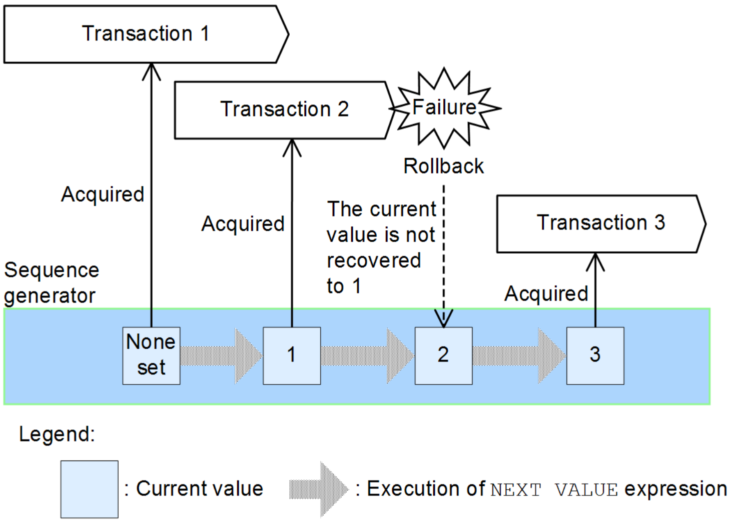
4.19 Automatic numbering facility

The automatic numbering facility returns a series of integer values whenever data in the database is called. You can use this facility by defining a sequence generator. The automatic numbering facility helps you improve the efficiency of developing UAPs that perform numbering. It also helps improve portability from UAPs created using other DBMSs that support sequence generators. We recommend that you use the automatic numbering facility for numbering.

4.19.1 About sequence generators

A sequence generator generates one consecutive number (sequential number) at a time regardless of user or transaction status. The following figure provides an overview of a sequence generator.

Figure 4-75: Overview of a sequence generator



Explanation:

The NEXT VALUE expression is used to acquire a sequential number generated by the sequence generator. This expression acquires the value that immediately follows the most recent value (current value) generated by the sequence generator, and then updates the current value to that value.

If the NEXT VALUE expression has not yet been used since the sequence generator was defined, no current value is set. When the NEXT VALUE expression is used while no current value is set, the sequence generator's start value is returned and that value is stored as the current value.

Note

The current value is not recovered during a rollback. The next number in sequence is always generated regardless of the transaction's status.

4.19.2 Defining a sequence generator

You use `CREATE SEQUENCE` to define a sequence generator. For details about `CREATE SEQUENCE`, see the manual *HiRDB Version 9 SQL Reference*.

The following shows an example of a definition of a sequence generator:

```
CREATE SEQUENCE SEQ1..........1
  AS INTEGER..................2
  START WITH 10...............3
  INCREMENT BY 10.............4
  MAXVALUE 999................5
  MINVALUE 10.................6
  CYCLE.......................7
  LOG INTERVAL 3..............8
IN RD01.......................9
```

Explanation:

1. Defines sequence generator `SEQ1` consisting of conditions 2 through 9.
2. Data type
3. Start value
4. Increment value
5. Maximum value
6. Minimum value
7. Cycle specification

   Cycles the numbers so that the maximum value (999) is followed immediately by the minimum value (10) with an increment value of 10.
8. Log output interval
9. Storage RDAREA name

Note

When the cycle specification is used, the sequence generator is capable of generating duplicate sequence numbers.

(1) Specifying the sequence generator storage RDAREA

When you define a sequence generator, you can specify any RDAREA for the storage for the sequence generator that satisfies the following conditions:

- RDAREA in which fewer than 500 tables and sequence generators combined are defined
- RDAREA that is not shut down
- RDAREA to which the inner replica facility is not being applied

Notes

- In a HiRDB/Parallel Server, if the sequence generator and the table that uses the sequence generator (and which is not partitioned among multiple servers) are stored in separate servers, communication will occur each time the sequence generator is used, which will have an adverse effect on performance. For a table that is partitioned among multiple servers, you might be able to reduce the overhead for communication by storing the sequence generator and the table that uses the sequence generator in an RDAREA on the same server. Therefore, we recommend that you store a sequence generator and the table that uses the sequence generator in an RDAREA on the same server.
- The inner replica facility cannot be used on an RDAREA that stores a sequence generator.

(2) Specifying the log output interval

You can improve performance by how you specify the log output interval when you define the sequence generator.

Sequence generator logs are output at the following times:

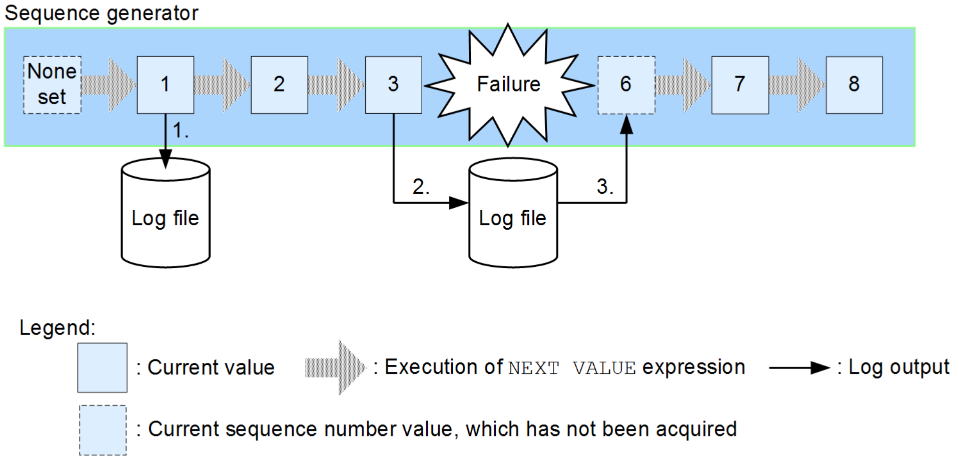
- The first time the `NEXT VALUE` expression is used after the sequence generator has been defined
- When the number of times a sequence number is collected equals the log output interval that has been specified

The following figure shows an example of a sequence generator for which a log output interval is specified.

Sequence generator definition

```
CREATE SEQUENCE SEQ1
  AS INTEGER
  START WITH 1
  INCREMENT BY 1
  MAXVALUE 999
  MINVALUE 1
  CYCLE
  LOG INTERVAL 3
IN RD01
```

Sequence generator


None set
1
2
3
Failure
6
7
8
1.
2.
3.
Log file
Log file
Legend:
: Current value
: Execution of NEXT VALUE expression
: Log output
: Current sequence number value, which has not been acquired

Explanation:

1. A log is output the first time a sequence number is acquired after the sequence generator is defined.
2. A log is output when the number of times a sequence number has been collected reaches the log output interval (3 times in this case).
3. If a HiRDB system failure occurs, the log output value at the time of the restart (6 in this example) is loaded as the current value. The sequence numbers generated since the last log output prior to the failure and the time of the failure (4, 5, and 6) are lost.

Notes

- Setting a small value for the sequence generator's log output interval will reduce the number of skipped numbers in the event of a system failure, but performance will be affected adversely because the number of log output operations is increased. On the other hand, if you set a large value for the log output interval, more numbers will be skipped in the event of a system failure, but performance will be improved because you reduce the number of log output operations.
- The maximum number of skipped numbers will be the same as the value specified for the log output interval.

4.19.3 Deleting a sequence generator

You use `DROP SEQUENCE` to delete a sequence generator. For details about `DROP SEQUENCE`, see the manual *HiRDB Version 9 SQL Reference*.

4.19.4 Acquiring the sequence numbers generated by a sequence generator

There are two ways to acquire the sequence numbers generated by a sequence generator:

- Data loading using the automatic numbering facility
- Using the `NEXT VALUE` expression

For details about data loading using the automatic numbering facility, see the manual *HiRDB Version 9 Command Reference*. For details about the `NEXT VALUE` expression, see the manual *HiRDB Version 9 SQL Reference*.

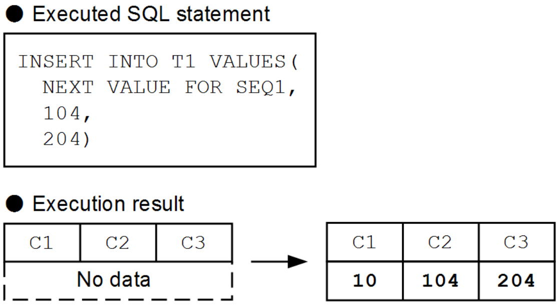
4.19.5 Examples

For these examples, you specify the `NEXT VALUE` expression in a selection expression in a query expression in the `INSERT` statement, an insertion value in the `INSERT` statement, or an update value in the `UPDATE` statement. If you specify the `NEXT VALUE` expression more than once for the same row using the same sequence generator, the `NEXT VALUE` expressions all return the same value.

Examples of the `NEXT VALUE` expression are presented below. These examples use sequence generator `SEQ1` that was defined in *4.19.2 Defining a sequence generator*.

Example:

If the `NEXT VALUE` expression is executed before any sequence number has been acquired from sequence generator `SEQ1`, the sequence generator's start value (10) is returned.



● Executed SQL statement

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  104,
  204)
```

● Execution result

C1	C2	C3
No data		

→

C1	C2	C3
10	**104**	**204**

Example:

If two or more `NEXT VALUE` expressions with the same sequence generator specified are specified for the same row, they all return the same value (20).

● Executed SQL statement

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  NEXT VALUE FOR SEQ1 + 1,
  202)
```

● Execution result

C1	C2	C3
10	101	201

→

C1	C2	C3
10	101	201
20	**21**	**202**

Example:

If two or more NEXT VALUE expressions with the same sequence generator specified are specified for the same row for a table that contains repetition columns, they all return the same value (30).

● Executed SQL statement

```
INSERT INTO T2 VALUES(
 NEXT VALUE FOR SEQ1,
  ARRAY [
   NEXT VALUE FOR SEQ1 + 1,
   NEXT VALUE FOR SEQ1 + 2
  ],
 203)
```

● Execution result

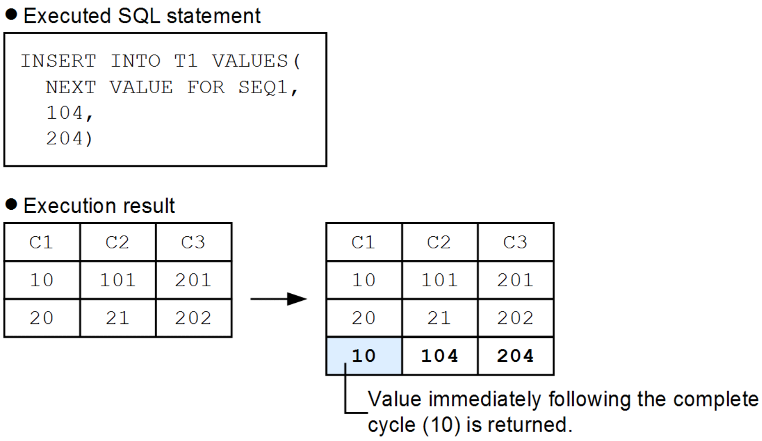
C1	C2	C3
No data		

→

C1	C2	C3
30	**31**	**203**
	32	

Example:

If the NEXT VALUE expression is specified when the current value of sequence generator SEQ1 is the maximum value (990), the value immediately following the complete cycle (10) is returned.



- Executed SQL statement

```
INSERT INTO T1 VALUES(
  NEXT VALUE FOR SEQ1,
  104,
  204)
```

- Execution result

C1	C2	C3
10	101	201
20	21	202

→

C1	C2	C3
10	101	201
20	21	202
10	**104**	**204**

Value immediately following the complete cycle (10) is returned.

4.19.6 Notes

Depending on the order in which the sequence generator storage RDAREA and the RDAREA storing the table that uses the sequence generator are backed up, there might be duplicate or missing sequence numbers after the database has been recovered.

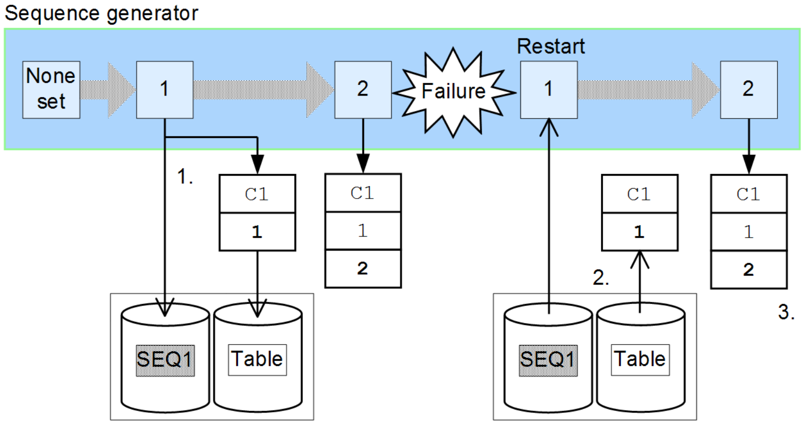
(1) Case in which there are no duplicate or missing sequence numbers

If you back up the sequence generator storage RDAREA and the RDAREA storing the table that uses the sequence generator at the same time, there will be no duplicated or missing sequence numbers.

Figure 4-76: Backing up the sequence generator storage RDAREA and the RDAREA storing the table that uses the sequence generator at the same time

Sequence generator definition

```
CREATE SEQUENCE SEQ1
  START WITH    1
  INCREMENT BY  1
  MAXVALUE      999
  MINVALUE      1
  LOG INTERVAL  1
  CYCLE
```



Legend:

□ : Current value

⇨ : Execution of `NEXT VALUE` expression

→ : Input to or output from backup

➔ : Execution of `INSERT` statement

Explanation:

1. Inserts (`INSERT`) into the table the sequence number obtained from sequence generator `SEQ1` and then backs up the sequence generator and the table at the same time.
2. Restores the sequence generator and the table from the backup files when the system restarts after recovering from the system failure.
3. Inserts (`INSERT`) into the table the sequence number. This sequence number is 2 and there is no duplicate or missing number because the sequence generator has been restored to the point of the backup.

(2) Case in which there are duplicated or missing sequence numbers

(a) When the sequence generator is backed up before the table that uses the sequence generator

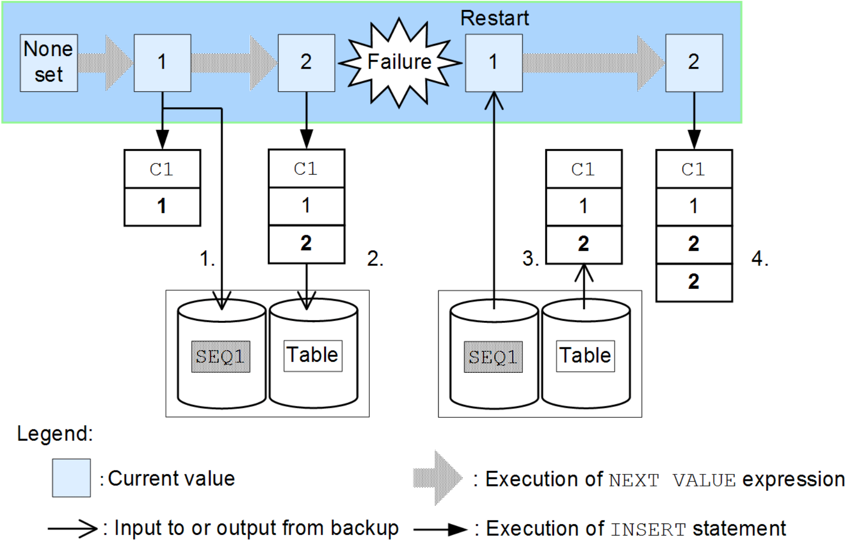
There might be duplicated sequence numbers. The following figure provides an example.

Figure 4-77: When the sequence generator is backed up before the table that uses the sequence generator

Sequence generator definition

```
CREATE SEQUENCE SEQ1
  START WITH    1
  INCREMENT BY  1
  MAXVALUE      999
  MINVALUE      1
  LOG INTERVAL  1
  CYCLE
```

Sequence generator


Restart
None set
1
2
Failure
1
2
C1
1
C1
1
2
C1
1
2
C1
1
2
2
1.
2.
3.
4.
SEQ1
Table
SEQ1
Table
Legend:
: Current value
: Execution of NEXT VALUE expression
: Input to or output from backup
: Execution of INSERT statement

Explanation:

1. Inserts (INSERT) sequence number 1 obtained from sequence generator SEQ1 into the table, and then backs up the sequence generator.
2. Inserts (INSERT) sequence number 2 into the table, and then backs up the table.
3. Restores the sequence generator and the table from the backup files when the system restarts after recovering from the system failure.
4. Inserts (INSERT) sequence number 2 into the table. This sequence number is 2 because the sequence generator has been restored to the point of the backup.
   As a result, sequence number 2 obtained from the sequence generator is duplicated because it was obtained after the sequence generator was backed up but before the table was backed up.

(b) When the replication facility is used

Sequence numbers might be duplicated.

Sequence generators cannot be used with the replication facility (HiRDB Dataextractor and HiRDB Datareplicator). If sequence generators with the same name are used at the source and the target, sequence numbers might be duplicated because the source sequence generator's current value cannot be inherited to the target.

(c) When the table that uses the sequence generator is backed up before the sequence generator

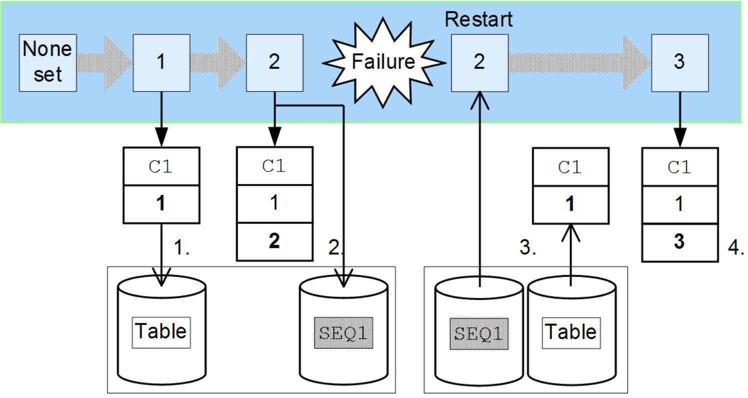
There might be missing sequence numbers. The following figure provides an example.

Figure 4-78: When the table that uses the sequence generator is backed up before the sequence generator

Sequence generator definition

```
CREATE SEQUENCE SEQ1
  START WITH   1
  INCREMENT BY 1
  MAXVALUE     999
  MINVALUE     1
  LOG INTERVAL 1
  CYCLE
```

Sequence generator



Legend:

□ : Current value

⇨ : Execution of NEXT VALUE expression

→ : Input to or output from backup

➞ : Execution of INSERT statement

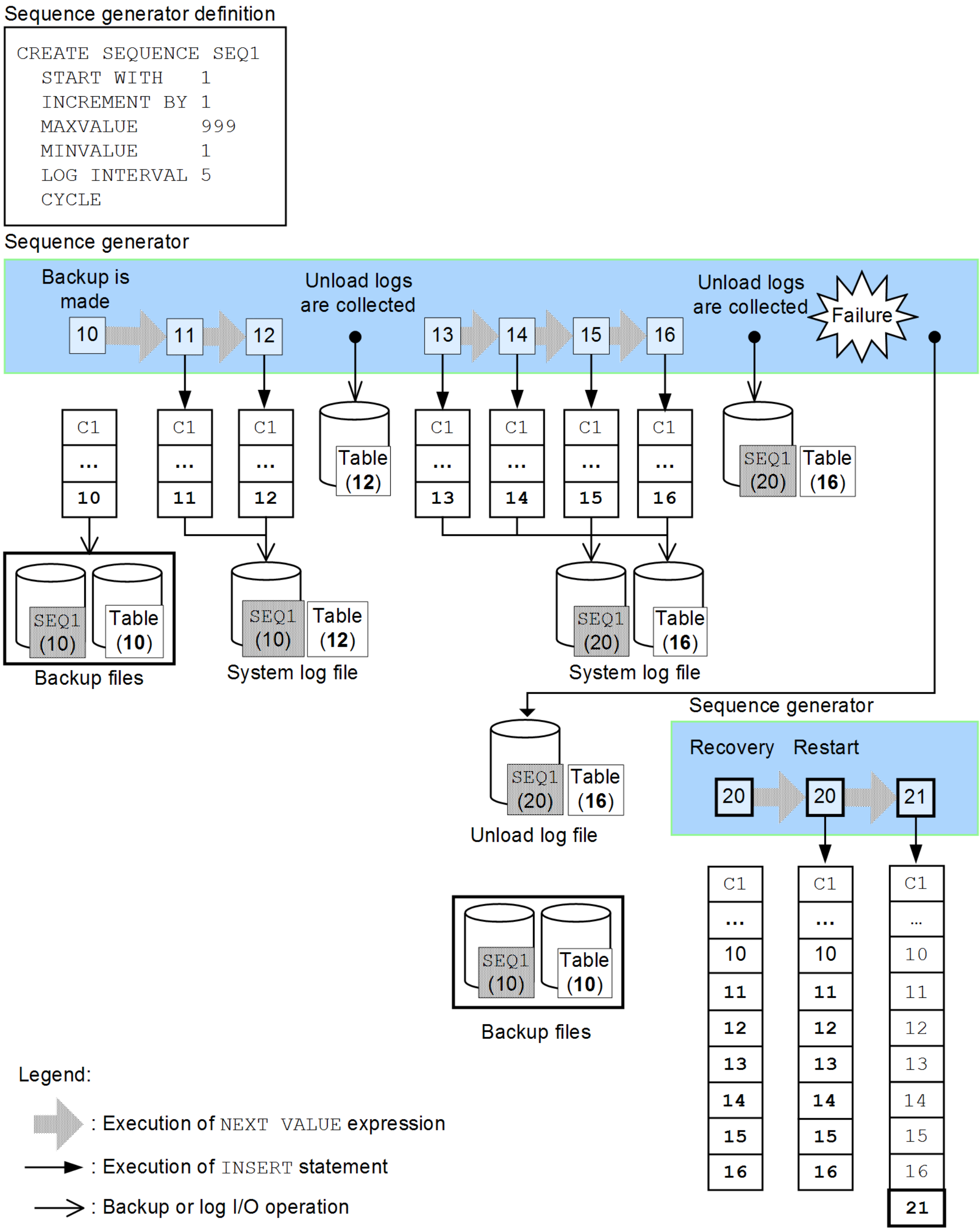
Explanation:

1. Inserts (INSERT) sequence number 1 obtained from sequence generator SEQ1 into the table, and then backs up the table.
2. Inserts (INSERT) sequence number 2 into the table, and then backs up the sequence generator.
3. Restores the sequence generator and the table from the backup files when the system restarts after recovering from the system failure.
4. Inserts (INSERT) sequence number 3 into the table. This sequence number is 3 because the sequence generator has been restored to the point of the backup.
   As a result, sequence number 2 obtained from the sequence generator is missing because it was obtained after the table was backed up but before the sequence generator was backed up.

(d) When logs are used for recovery

If you wish to restore the database to a desired synchronization point after a backup was made or to the most recent synchronization point before a failure, you must restore the backup files and system log files or unload log file. In this case, as many sequence numbers might be missing as the value you set for the log output interval when you defined the sequence generator. The following figure provides an example.

Figure 4-79: When logs are used for recovery



Sequence generator definition

```
CREATE SEQUENCE SEQ1
  START WITH    1
  INCREMENT BY 1
  MAXVALUE      999
  MINVALUE      1
  LOG INTERVAL 5
  CYCLE
```

Explanation:

If the unload log file is used to recover sequence generator SEQ1 and the table, and then the NEXT VALUE expression is executed after recovery, the value 21 is obtained from sequence generator SEQ1. The values 17 through 20 are missing because they had not been acquired between the time the backup was made and time the failure occurred.

Chapter

5. Notes about Creating UAPs that Access Object Relational Databases

This chapter gives notes about creating UAPs that access object relational databases.

This chapter contains the following sections:

5.1 Using abstract data types and user-defined functions
5.2 Restrictions on functions provided by plug-ins

5.1 Using abstract data types and user-defined functions

This section describes writing UAPs that access tables with abstract data types and UAPs that have user-defined functions.

(1) Embedded variable data types

- User-defined data types cannot be specified in embedded variable declarations (that is, in embedded SQL declaration clauses).
- If the table column to be retrieved contains an abstract data type, no column can be specified in the selection expression of the `SELECT` statement.
- When embedded variables are specified in function arguments, the specifications must match the actual argument data types of the function used. If the embedded variable specifications and the argument data types in the function do not match, that function cannot be used.

  To determine the argument data types of a function, retrieve the `SQL_ROUTINE_PARAMS` dictionary table. For details about how to retrieve data dictionary tables, see Appendix *G.1 Examples of SQL statements for retrieval*.

Note

An embedded variable cannot be used if the `ROW` data type is specified in a function argument.

(2) Literal data types

When literals are used in function arguments, the literals must match the actual argument data types of the function used. For example, if the data type of a function argument is `SMALLINT`, and an integer literal is specified, the literal does not match the data type. If another function has the same name, the same number of arguments, and an argument with the integer type, that function may be used instead.

Literals cannot be used in function arguments that have one of the following data types specified:

- `SMALLINT`
- `SMALLFLT`
- `CHAR`
- `NCHAR`
- `MCHAR`
- `DATE`
- `TIME`

- `TIMESTAMP`
- `INTERVAL YEAR TO DAY`
- `INTERVAL HOUR TO SECOND`
- `ROW`
- `BLOB`
- `BINARY`

5.2 Restrictions on functions provided by plug-ins

Functions provided by plug-ins are called *plug-in distribution functions*.

(1) Restrictions on passing values between plug-in distribution functions

(a) Types of plug-in distribution functions

The following table lists the types of plug-in distribution functions.

Table 5-1: Types of plug-in distribution functions

Function type	**Plug-in process**	
	Process that generates inter-function values to be passed and sends them to other plug-in distribution functions	**Process that receives inter-function values sent from other plug-in distribution functions**
Function that does not have inter-function values	Not supported	Not supported
Function that sends inter-function values[#1]	Not supported	Supported
Function that receives passed inter-function values[#2]	Supported	Not supported
Function that sends and receives inter-function values	Supported	Supported

#1: An example of this function type that the HiRDB Text Search Plug-in provides is the `score` function.

#2: An example of this function type that the HiRDB Text Search Plug-in provides is the `contains_with_score` function.

The HiRDB system allows values to be passed between plug-in distribution functions. Because HiRDB automatically passes inter-function values between plug-in distribution functions, the inter-function value does not have to be specified as an argument of the plug-in distribution function.

Note

> Where the plug-in distribution function can be specified in an SQL differs depending on the function type. For details about the plug-in distribution function types, see the plug-in manuals.

The following explanations use these terms:

- Function that does not have passing inter-function values → Function without inter-function values
- Function that receives passing inter-function values → Receive function for passing inter-function values
- Function that sends passing inter-function values → Send function for passing inter-function values
- Function that sends and receives passing inter-function values → Send/receive function for passing inter-function values

(b) Correspondences between send and receive functions for passing inter-function values

The following are rules for the correspondences between send and receive functions for passing inter-function values.

- Some combinations of send and receive functions for passing inter-function values do not allow inter-function values to be passed. For details about the correspondences between send and receive functions for passing inter-function values, see the plug-in manuals.
- The first arguments in both the send and receive functions for passing inter-function values must be the same and must be a column specification for a base table, an SQL parameter, or an SQL variable. The first argument cannot be a component specification.
- Use one query specification to close the send and receive functions for passing inter-function values. However, when you specify a send function for passing inter-function values during list creation to store the passing inter-function values to a list, and then specify a receive function for passing inter-function values during search via a list to get the passing inter-function values from the list, you can specify the send and receive functions for passing inter-function values across multiple queries. (For details, see *(3)(c) Methods of executing set operations between lists*.)

The following table shows the combinations of receive and send functions for passing inter-function values and the action by HiRDB.

Table 5-2: Correspondences between receive and send functions for passing inter-function values

Receive function for passing inter-function values	Send function for passing inter-function values	HiRDB action
Not specified	Not specified	Can be executed
	Specified	

Receive function for passing inter-function values	Send function for passing inter-function values	HiRDB action
Specified	Not specified	Cannot be executed#
	Specified (one)	Can be executed
	Specified (two or more)	Cannot be executed

#: When passing inter-function values are to be obtained from a list, the send and receive functions for passing inter-function values can be specified across multiple queries.

(c) Restrictions on plug-in distribution functions

- **Functions without inter-function values**

  Any of these functions can be specified in locations where a function can be specified.

- **Receive functions for passing inter-function values**

  - These functions can be specified only in the selection expression of a SELECT statement, the selection expression of an INSERT statement that has a query specification, or the update value of a SET clause in an UPDATE statement.
  - These functions cannot be specified in CASE expressions or the VALUE scalar function.
  - When a GROUP BY clause, a HAVING clause, or a set function is specified, receive functions for passing inter-function values that have an SQL variable or an SQL parameter as the first argument can be specified only in a set function argument.

- **Send functions for passing inter-function values**

  If a receive function for passing inter-function values is not specified

  Any send function for passing inter-function values can be specified in locations where a function can be specified.

  If a receive function for passing inter-function values is specified

  - The receive function can be specified only in a WHERE clause or an ON search condition.
  - If a send function for passing inter-function values is specified in the ON search condition of a joined table that specifies an outer join, a column of the outer table cannot be specified in the first argument of the function.
  - If a send function for passing inter-function values is specified in the search

conditions of the OR operand, all of the following conditions must be satisfied:

- A plug-in instance is defined in the first argument of the send function for passing inter-function values.

- The first argument of the send function for passing inter-function values specifies a base table column that is not a reference column to the outside.

- The second and subsequent arguments of the send function for passing inter-function values do not specify a column (except a reference column to the outside) or an argument that includes a value expression for a component specification of a column.

- A predicate that includes `IS FALSE`, `IS UNKNOWN`, or `NOT` is not specified for the send function for passing inter-function values.

- The send function for passing inter-function values is not specified in a `CAST` specification.

- If the `FROM` clause contains specifications for two or more tables, a table column that is different from the column specified in the first argument of the send function for passing inter-function values cannot be specified in the search conditions of the `OR` operand. (When the `WHERE` clause or the `ON` search condition contains the `NOT` Boolean operator, the same applies, even if the previous condition is satisfied after the `NOT` Boolean operator is eliminated by De Morgan's theorem.)

- Send functions for passing inter-function values cannot be specified in `CASE` expressions or the `VALUE scalar` function.
- A restriction applies if a named derived table defined by specifying the `GROUP BY` clause, `HAVING` clause, or a set function is specified in the `FROM` clause, and the named derived table does not create an internal derived table. In this case, a send function for passing inter-function values in which the first argument becomes an SQL value or SQL parameter cannot be specified in the search conditions of the query specification that specifies the named derived table.

■ **Send/receive functions for passing inter-function values**

These functions cannot be specified in SQL statements.

(2) Restrictions on executing plug-in distribution functions

(a) Execution methods for plug-in distribution functions

You can execute plug-in distribution functions in two ways:

- Use an index-type plug-in to execute a plug-in distribution function
- Execute a plug-in distribution function without using an index-type plug-in

Some plug-in distribution functions can be executed only if an index-type plug-in is used (index-type plug-in-dependent function).

When HiRDB executes an index-type plug-in-dependent function, an error occurs if HiRDB determines that the index-type plug-in cannot be used. The table below shows the combinations that trigger an error. To find out whether a plug-in distribution function requires an index-type plug-in, see the plug-in manuals.

Table 5-3: Combinations that trigger an error when a plug-in distribution function is executed

Method that uses index-type plug-in to execute function	Method that executes function without using index-type plug-in	Retrieval method selected by HiRDB	
		Retrieval with index-type plug-in	Retrieval without index-type plug-in
Provided	Provided	E	E
Provided#	Not provided#	E	--
Not provided	Provided	NA	E

E: Can be executed

--: Error occurs when executed

NA: Not applicable

#: Index-type plug-in-dependent functions fall into this category. Examples for the HiRDB Text Search Plug-in are `contains` and `contains_with_score`.

(b) Restrictions on execution methods for index-type plug-in-dependent functions

The following restrictions apply when index-type plug-in-dependent functions are used:

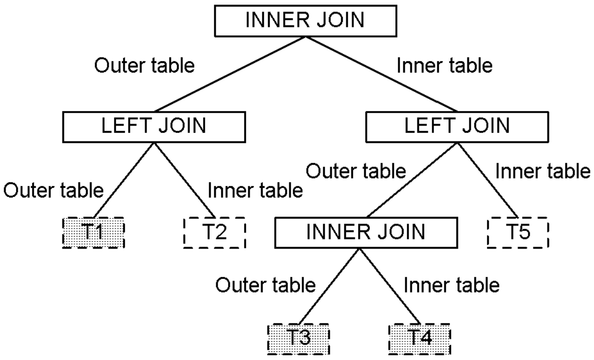
1. Only a base table column specification can be specified in the first argument. Also, the column cannot be an external-referencing column.
2. Arguments that include the following value expressions cannot be specified in any argument except the first:
   - Column specifications, except external-referencing columns
   - Component specifications for columns
3. Index-type plug-in-dependent functions can be specified in `WHERE` clauses and `ON` search conditions.
4. If an index-type plug-in-dependent function is specified in the `WHERE` clause of a query specification that specifies an outer join, a column that becomes the inner

table of the outer join cannot be specified in the first argument. An example is shown as follows:

SQL statements

```
SELECT * FROM
 (T1 LEFT JOIN T2 ON search-condition-1 )
  INNER JOIN
   ((T3 INNER JOIN T4 ON search-condition-2 )
    LEFT JOIN T5 ON search-condition-3 )
  ON search-condition-4
  WHERE search-condition-5
```

... When an index-type plug-in-dependent function is specified in *search-condition-5* , a column from table T1, T3, or T4 can be specified in argument 1.



5. If a index-type plug-in dependent function is specified in the `ON` search condition of a joined table that specifies an outer join, the following columns cannot be specified in the first argument:
   - Columns of the outer table
   - Columns of the outer-joined inner table included in the inner table, if the inner table is a joined table containing an outer join
6. If the `FROM` clause contains specifications for two or more tables, a table column that is different from the column specified in the first argument of an index-type plug-in-dependent function cannot be specified in the search condition of the `OR` operand. However, when the `WHERE` clause or the `ON` search condition contains the `NOT` logical operator, a different table column can be specified, if the previous condition is not satisfied after the `NOT` logical operator is eliminated by De Morgan's theorem.# An example is shown as follows:

   Example:

   ```
   SELECT T1.C1,T2.C2 FROM T1,T2
     WHERE T1.C1=10 AND ((CONTAINS(T2.ADT,'ABC') IS TRUE)
                      OR CONTAINS(T2.ADT,'DEF') IS TRUE))
   ```

The UNION representation of this SQL is as follows:

```
(SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE T1.C1=10 AND (CONTAINS(T1.ADT,'ABC') IS TRUE)
UNION ALL
  SELECT T1.C1,T2.C2 FROM T1,T2
      WHERE T1.C1=10 AND (CONTAINS(T2.ADT,'DEF') IS TRUE))
EXCEPT ALL
  SELECT T1.C1,T2.C2 FROM T1,T2
      WHERE T1.C1=10 AND (CONTAINS(T2.ADT,'DEF') IS TRUE)
            AND (CONTAINS(T1.ADT,'ABC') IS TRUE)
```

#: Assume that the following SQL statements have been specified:

```
SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE NOT(CONTAINS(T1.ADT, ...)IS NOT TRUE AND T1.C1=10)
  AND T1.C1=T2.C1
```

If the NOT logical operator is eliminated according to De Morgan's theorem, the result is as follows:

```
SELECT T1.C1,T2.C2 FROM T1,T2
  WHERE NOT(CONTAINS(T1.ADT, ...)IS TRUE OR T1.C1<>10)
  AND T1.C1=T2.C1
```

7. Index-type plug-in dependent functions cannot be specified in CASE expressions and CAST specifications.
8. Predicates that include IS FALSE, IS UNKNOWN, or negation (NOT) cannot be specified for index-type plug-in-dependent functions.

Examples related to these restrictions are as follows.

Example 1

If the WHERE clause specifies a send function for passing inter-function values and that function is dependent on an index-type plug-in, the first argument cannot contain one query specification that specifies that send function, together with an index-type plug-in-dependent function that has a column from the same table.

```
SELECT C1,C2, score(SENTENCES) FROM T1
   WHERE contains(SENTENCES,...)IS TRUE
      AND contains_with_score(SENTENCES, ...) IS TRUE
```

Example 2

This example outer-joins tables T1 and T2 and retrieves data by specifying an index-type plug-in-dependent function in the WHERE clause.

```
SELECT T1.C1,T2.C2 FROM T1 LEFT OUTER JOIN T2
   ON T1.C1=T2.C1 WHERE contains(T1.C3, ...)IS TRUE
```

(3) Notes on storing passing inter-function values to a list

(a) Storing passing inter-function values to a list

When target records are narrowed hierarchically (a narrowed search is performed), the results of the receive function for passing inter-function values can be obtained quickly by storing the passing inter-function values to a list.

To store passing inter-function values to a list, use the `ASSIGN LIST` statement to specify a send function for passing inter-function values in the search conditions for creating a list from a base table. The send function for passing inter-function values must be able to store the passing inter-function values to a list. (However, only one send function for passing inter-function values that can store such values to a list can be specified in the `ASSIGN LIST` statement.)

For information about whether the functions provided by a plug-in can store passing inter-function values to a list, refer to the manual for that plug-in.

You can also use the `ASSIGN LIST` statement to store passing inter-function values from a list that stores such values to a new list.

(b) Getting passing inter-function values from a list

To get passing inter-function values stored to a list without specifying a send function for passing inter-function values, specify a receive function for passing inter-function values that can get such values from a list (receive function for passing inter-function values for lists) in the selection expression of the cursor specification for search via a list.

For information about whether the functions provided by a plug-in can get passing inter-function values from a list, refer to the manual for that plug-in.

If a receive function for passing inter-function values that can get such values from a list is specified in the selection expression of the cursor specification for search via a list, HiRDB gets those values without evaluating the type of send function for passing inter-function values that stored those values to the list. Therefore, be sure to specify a receive function for passing inter-function values that corresponds to the send function for passing inter-function values specified when the list was created.

(c) Methods of executing set operations between lists

If a set operation between lists is to be performed, the set operation execution method changes depending on the send function for passing inter-function values that was specified in the search conditions for list creation.

The following table shows the passing inter-function values in the set operation results for the following:

list-name-1 {`AND` | `OR` | `AND NOT` | `ANDNOT`} *list-name-2*

Table 5-4: Passing inter-function values in set operation results

Send function for passing inter-function values when list-name-1 is created[#1]		Send function for passing inter-function values when list-name-2 is created[#1]		
		When passing inter-function values can be stored to a list		Other cases
		Passing inter-function values for narrowing used is specified	No set operation method is specified	
When passing inter-function values can be stored to a list.	`Passing inter-function values for narrowing used` is specified.	N	N	Passing inter-function values of *list-name-1*[#2]
	No set operation method is specified.	N	N	N
Other cases		N	N	None

Legend:

N: Cannot be executed.

#1: For information about send functions for storing inter-function variables that allow a set operation method to be specified, refer to the manual of the individual plug-in.

#2: The set operation result becomes the null value if the `OR` operation results do not include passing inter-function values.

Chapter
6. Client Environment Setup

This chapter explains how to install a HiRDB client and how to define the environment for creating and executing a UAP.

This chapter contains the following sections:

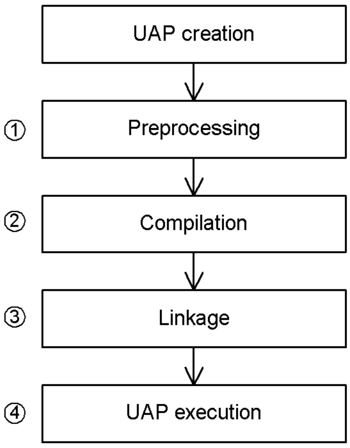
6.1 Types of HiRDB clients
6.2 Environment setup procedure for HiRDB clients
6.3 HiRDB client installation
6.4 Organization of directories and files for a HiRDB client
6.5 Setting the hosts file
6.6 Client environment definitions (setting environment variables)
6.7 Registering an environment variable group

6.1 Types of HiRDB clients

There are two programs that are categorized as HiRDB clients. These programs are called *HiRDB clients*:

- HiRDB/Developer's Kit
- HiRDB/Run Time

The available operations, from UAP creation to execution, depend on the type of HiRDB client. The following figure shows the procedure from UAP creation to execution.



For details about preprocessing, see Section *8.2 Preprocessing.*

For details about compiling, see Section *8.3 Compiling and linking.*

For details about linking, see Section *8.3 Compiling and linking.*

The operations available to each type of HiRDB client are as follows:

- HiRDB/Developer's Kit

  (1) - (4) are available.
- HiRDB/Run Time

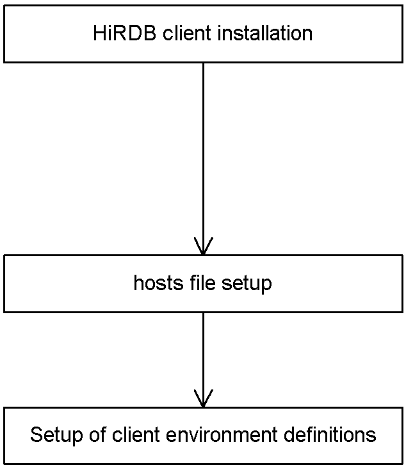
  (4) only is available. Because the programs provided for the HiRDB client are also provided by the HiRDB server, use the HiRDB server functions to execute (1) - (3).

Note

Use the same platform for the HiRDB/Developer's Kit used to create the UAP and the HiRDB/Developer's Kit used to execute the UAP.

6.2 Environment setup procedure for HiRDB clients

The procedure for setting the client environment is shown as follows.


HiRDB client installation
hosts file setup
Setup of client environment definitions

Install the HiRDB client programs. After the programs are installed, the system automatically creates the directory and files. For details about HiRDB client installation, see Section *6.3 HiRDB client installation.* For details about the directory and files that are created automatically, see Section *6.4 Organization of directories and files for a HiRDB client.*

Set information to the hosts file.
For details, see Section *6.5 Setting the hosts file*.

Set client environment definitions for each HiRDB client that was installed. For details about client environment definitions, see Section *6.6 Client environment definition (setting environment variables)*.

6.3 HiRDB client installation

The installation procedure is the same for both HiRDB/Developer's Kit and HiRDB/Run Time.

Note that the HiRDB client programs are already included in the HiRDB server. Therefore, if you want use the HiRDB client on the same server machine on which the HiRDB server operates, you do not need to install the HiRDB client on that server machine. See *Figure 1-5* for the operating mode in which the HiRDB client operates on the same server machine as the HiRDB server.

6.3.1 Installing a HiRDB client on a UNIX client

You use the Hitachi Program Product Installer to install a HiRDB client.

6.3.2 Installing a HiRDB client on a Windows client

When you install the HiRDB client, the environment definition file (`HIRDB.INI`) is stored in the system directory.

To install the HiRDB client:

1. Start the installer

   Execute `hcd_inst.exe` found on the integrated CD-ROM to start Hitachi Integrated Installer.

   At the Hitachi Integrated Installer screen, select one of the following, and then click the **Execute installation** button:

   - For HiRDB/Run Time, select **HiRDB/Run Time**.
   - For HiRDB/Developer's Kit, select **HiRDB/Developer's Kit**.

   The HiRDB setup program is activated.

   At the Select Program Product screen of the HiRDB setup program, select one of the following, and then click the **Next** button:

   - For HiRDB/Run Time, select **HiRDB/Run Time**.
   - For HiRDB/Developer's Kit, select **HiRDB/Developer's Kit**.

   The setup program of the selected program product is activated.

2. Register user information

   The User Information screen is displayed.

   Enter the user name and company name, and then click the **Next** button.

3. Start installation

The Select Installation Folder screen is displayed.

In **Installation folder**, specify the location where the HiRDB client is to be installed. If you omit this value, the drive in which Windows is installed is assumed. After specifying the installation folder, click the **Next** button.

4. Select the setup type

The Setup Type dialog box is displayed.

Depending on the setup type, you can change the functions to be installed.

Typical

Installs all functions.

Compact

Installs the minimum number of functions that are required.

Custom

Installs the functions you select.

The following table shows the functions that are installed when **Typical** and **Compact** are selected.

Table 6-1: Functions that are installed

No.	Client	Function	Product type	Setup type	
				All	Minimum
1	XDS client	Regular libraries	RT,DK	Y#	Y#
2		ODBC driver	RT,DK	Y	N
3		JDBC driver	RT,DK	Y	N
4		Sample UAP	DK	Y	N

No.	Client	Function	Product type	Setup type	
				All	Minimum
5	Client of a server that provides primary functions	Regular libraries	RT,DK	Y#	Y#
6		XA libraries	RT,DK	Y	N
7		ODBC driver	RT,DK	Y	N
8		HiRDB OLE DB provider	RT,DK	Y	N
9		HiRDB.NET data provider	RT,DK	Y	N
10		JDBC driver	RT,DK	Y	N
11		SQLJ	RT,DK	Y	N
12		Sample UAP	DK	Y	N

Legend:

RT: HiRDB/Run Time

DK: HiRDB/Developer's Kit

Y: Installed

N: Not installed

#

Commands and utilities are also installed with HiRDB/Run Time.

Commands, utilities, and the SQL preprocessor are also installed with HiRDB/Developer's Kit.

Once you have selected the setup type, click the **Next** button.

5. Select functions

   If you selected **Custom**, the Select Features dialog box is displayed.

   Select the functions that you wish to install and then click the **Next** button.

6. Set up the Path and CLASSPATH environment variables

   The Set up Environment Variable Path dialog box and the Set up Environment Variable CLASSPATH dialog box might be displayed. If these dialog boxes are displayed, select either the XDS client or the client for a server that provides primary functions, and then click the **Next** button.

7. Start copying files

   The Starting Copying Files dialog box is displayed.

   Check the current settings, and then click the **Next** button.

8. Check the execution status of the installation

   The execution status of the installation is displayed.

   If there is not enough space to install the HiRDB client, the message `Error occurred during data transfer` is displayed while the execution status of the installation is displayed, and the installation is canceled. In such a case, check that there is adequate space at the installation target and then re-execute the installation.

9. Terminate the installation program

   When installation is complete, the Setup Complete screen is displayed.

Notes

- The maximum length of a line in the environment definition file is 512 bytes. A definition line that exceeds 512 bytes is ignored.
- When you install a HiRDB client, you must have Administrator or Power User permissions. If a user who does not have Administrator or Power User permissions installs the client, the system environment variables are not updated. In Windows 2000, redistributed files are not updated either.

6.4 Organization of directories and files for a HiRDB client

When a HiRDB client is installed, directories and files are created automatically. This section explains the organization of the directories and files.

6.4.1 Directories and files for UNIX clients

Tables 6-2 to *6-7* list the files and directories that are created automatically during HiRDB client installation on a client machine.

Table 6-2: Files and directories for workstation - HiRDB/Developer's Kit

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
Header files	/HiRDB/ include	SQLCA.CBL	C	C	C	C	C	C	C
		SQLCA64.CBL	--	--	--	--	--	C	--
		SQLDA.CBL	C	C	C	C	C	C	C
		SQLDA64.CBL	--	--	--	--	--	C	--
		SQLIOA.CBL	C	C	C	C	C	C	C
		SQLIOA64.CBL	--	--	--	--	--	C	--
		pdbtypes.h	C	C	C	C	C	C	C
		pdberrno.h	C	C	C	C	C	C	C
		pdbmisc.h	C	C	C	C	C	C	C
		pdbmiscm.h	C	C	C	C	C	C	C
		pdbsqlda.h	C	C	C	C	C	C	C
		pdbsqlcsna.h	C	C	C	C	C	C	C
		pddbhash.h	C	C	C	C	C	C	C
		pdauxcnv.h	C	C	C	C	C	C	C
		SQLCAM.cbl	C	C	C	C	C	C	C
		SQLDAM.cbl	C	C	C	C	C	C	C

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
		`SQLIOAM.cbl`	C	C	C	C	C	C	C
		`SQLIOAMTH.CBL`	C	C	--	--	C	C	C
		`SQLIOAMTH64.CBL`	--	--	--	--	--	C	--
		`SQLCAMTH.CBL`	C	C	--	--	C	C	C
		`SQLCAMTH64.CBL`	--	--	--	--	--	C	--
		`SQLCSNA.CBL`	C	C	C	C	C	C	C
		`SQLCSNA64.CBL`	--	--	--	--	--	C	--
Archive files	`/`*HiRDB*`/client/lib`	`libclt.a`	C	C	C	C	C	C	C
		`libclt64.a`	--	C	--	C	--	C	--
		`libcltxa.a`	C	C	C	C	C	C	C
		`libcltya.a`	C	C	C	C	C	C	C
		`libcltm.a`	C	C	C	C	--	--	--
		`libcltxam.a`	NF	NF	NF	NF	--	--	--
		`libcltyam.a`	NF	NF	NF	NF	--	--	--
		`libcltk.a`	C	C	C	C	C	C	C
		`libcltk64.a`	--	C	--	C	--	C	--
		`libclts.a`	C	C	C	C	C	C	C
Shared library files[#2]	`/`*HiRDB*`/client/lib`	`libzclt.sl`	C	C	C	C	C	C	C
		`libzclt64.sl`	--	C	--	C	--	C	--
		`libzcltx.sl`	C	C	C	C	C	C	C

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
		`libzclty.sl`	C	C	C	C	C	C	C
		`libzcltm.sl`	C	C	C	C	--	--	--
		`libzcltxm.sl`	NF	NF	NF	NF	--	--	--
		`libzcltym.sl`	NF	NF	NF	NF	--	--	--
		`libzcltk.sl`	C	C	C	C	C	C	C
		`libzcltk64.sl`	--	C	--	C	--	C	--
		`libzpdodbc.sl`	C	C	--	--	--	--	--
		`libsqlauxf.sl`	C	C	C	C	C	C	C
		`libsqlauxf64.sl`	--	C	--	C	--	C	--
		`libzcltxk.sl`	C	C	C	C	C	C	NF
		`libzcltyk.sl`	C	C	C	C	C	C	NF
		`libzclts.sl`	C	C	C	C	C	C	C
		`libzcltxs.sl`	C	C	C	C	C	C	C
		`libzcltys.sl`	C	C	C	C	C	C	C
		`libzclty64.sl`	--	--	--	--	--	C	--
		`libzcltys64.sl`	--	--	--	--	--	C	--
JDBC driver	`/`*HiRDB*`/client/lib`	`libjjdbc.sl`	C	C	C	C	C	C	C
		`pdjdbc.jar`	C	C	C	C	C	C	C

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
		`pdjdbc2.jar`	C	C	C	C	C	C	C
ODBC driver	`/`*HiRDB*`/client/lib`	`libodbcdrv.sl`	--	--	--	--	--	--	C
		`libodbcdrv64.sl`	--	--	--	--	--	--	--
Command utilities	`/`*HiRDB*`/client/utl`	`pdcpp`	C	C	C	C	C	C	C
		`pdocc`	C	C	C	C	C	C	C
		`pdcbl`	C	C	C	C	C	C	C
		`pdocb`	C	C	C	C	C	C	C
		`pdprep`	C	C	C	C	C	C	C
		`pdtrcmgr`	C	C	C	C	C	C	C
		`pdodbcsetup`	C	C	--	--	--	--	--
		`pdodbcconfig`	C	C	--	--	--	--	--
	`/`*HiRDB*`/bin`	`pddef`	C	C	C	C	C	C	C
SQLJ	`/`*HiRDB*`/client/lib`	`pdruntime.jar`	C	--	C	--	C	--	C
		`pdnativert.jar`	C	--	C	--	C	--	C
		`pdsqlj.jar`	C	--	C	--	C	--	C
		`libpdparse.sl`	C	--	C	--	C	--	C
		`libpdsqljn.sl`	C	--	C	--	C	--	C
	`/`*HiRDB*`/client/utl`	`pdjava`	C	--	C	--	C	--	C
Message object file	`/`*HiRDB*`/lib`	`msgtxt`	C	C	C	C	C	C	C

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
Parsing libraries[#2]	/*HiRDB*/lib/sjis	libasqap.sl	C	C	C	C	C	C	C
	/*HiRDB*/lib/chinese		C	C	C	C	C	C	C
	/*HiRDB*/lib/lang-c/		C	C	C	C	C	C	C
	/*HiRDB*/lib/ujis		C	C	C	C	C	C	C
	/*HiRDB*/lib/utf-8		C	C	C	C	C	C	C
	/*HiRDB*/lib/chinese-gb18030		C	C	C	C	C	C	C
Sample source files	/*HiRDB*/client/samplep/uap	CREATE.ec	C	C	C	C	C	C	C
		SAMPLE1.ec	C	C	C	C	C	C	C
		SAMPLE2.ec	C	C	C	C	C	C	C
		SAMPLE3.ec	C	C	C	C	C	C	C
		sample1.ecb	C	C	C	C	C	C	C
		sample.mk	C	C	C	C	C	C	C
		inputf1	C	C	C	C	C	C	C
		inputf2	C	C	C	C	C	C	C
XML conversion commands[#3]	/*HiRDB*/client/utl	phdxmlcnv	C	C	C	C	C	C	C
XML conversion libraries[#3]	/*HiRDB*/client/lib	XMLConverter.jar	C	C	C	C	C	C	C

Legend:

HP (32): 32-bit mode HP-UX

HP (64): 64-bit mode HP-UX

Sol (32): 32-bit mode Solaris

Sol (64): 64-bit mode Solaris

AIX (32): 32-bit mode AIX

AIX (64): 64-bit mode AIX

Linux (32): 32-bit mode Linux

C: The file is created.

NF: The file is created, but the facility that uses that file does not operate.

--: The file is not created.

#1: The underlined portion indicates the HiRDB installation directory.

#2: The suffixes for the shared library files and parsing libraries differ according to the platform. In Solaris and Linux, the suffix is `.so`; in AIX, the suffix is `.a`.

#3: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-3: Files and directories for HiRDB/Run Time (UNIX client)

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
Archive files	`/HiRDB/client/lib`	`libclt.a`	C	C	C	C	C	C	C
		`libclt64.a`	--	C	--	C	--	C	--
		`libcltxa.a`	C	C	C	C	C	C	C
		`libcltya.a`	C	C	C	C	C	C	C
		`libcltm.a`	C	C	C	C	--	--	--
		`libcltxam.a`	NF	NF	NF	NF	--	--	--
		`libcltyam.a`	NF	NF	NF	NF	--	--	--
		`libcltk.a`	C	C	C	C	C	C	C
		`libcltk64.a`	--	C	--	C	--	C	--
		`libclts.a`	C	C	C	C	C	C	C

Name	Dir[#1]	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
Shared library files[#2]	/*HiRDB*/ client/ lib	`libzclt.sl`	C	C	C	C	C	C	C
		`libzclt64.sl`	--	C	--	C	--	C	--
		`libzcltx.sl`	C	C	C	C	C	C	C
		`libzclty.sl`	C	C	C	C	C	C	C
		`libzcltm.sl`	C	C	C	C	--	--	--
		`libzcltxm.sl`	NF	NF	NF	NF	--	--	--
		`libzcltym.sl`	NF	NF	NF	NF	--	--	--
		`libzcltk.sl`	C	C	C	C	C	C	C
		`libzcltk64.sl`	--	C	--	C	--	C	--
		`libzpdodbc.sl`	C	C	--	--	--	--	--
		`libsqlauxf.sl`	C	C	C	C	C	C	C
		`libsqlauxf64.sl`	--	C	--	C	--	C	--
		`libzcltxk.sl`	C	C	C	C	C	C	NF
		`libzcltyk.sl`	C	C	C	C	C	C	NF
		`libzclts.sl`	C	C	C	C	C	C	C
		`libzcltxs.sl`	C	C	C	C	C	C	C
		`libzcltys.sl`	C	C	C	C	C	C	C
		`libzclty64.sl`	--	--	--	--	--	C	--
		`libzcltys64.sl`	--	--	--	--	--	C	--
JDBC driver	/*HiRDB*/ client/ lib	`libjjdbc.sl`	C	C	C	C	C	C	C
		`pdjdbc.jar`	C	C	C	C	C	C	C
		`pdjdbc2.jar`	C	C	C	C	C	C	C

Name	Dir#1	File name	Platform						
			HP (32)	HP (64)	Sol (32)	Sol (64)	AIX (32)	AIX (64)	Linux (32)
ODBC driver	/*HiRDB*/client/lib	`libodbcdrv.sl`	--	--	--	--	--	--	C
		`libodbcdrv64.sl`	--	--	--	--	--	--	--
SQLJ Runtime Library	/*HiRDB*/client/lib	`pdruntime.jar`	C	--	C	--	C	--	C
		`pdnativert.jar`	C	--	C	--	C	--	C
		`libpdsqljn.sl`	C	--	C	--	C	--	C
Command utilities	/*HiRDB*/client/utl	`pdtrcmgr`	C	C	C	C	C	C	C
		`pdodbcsetup`	C	C	--	--	--	--	--
		`pdodbcconfig`	C	C	--	--	--	--	--
XML conversion commands#3	/*HiRDB*/client/utl	`phdxmlcnv`	C	C	C	C	C	C	C
XML conversion libraries#3	/*HiRDB*/client/lib	`XMLConverter.jar`	C	C	C	C	C	C	C

Legend:

HP (32): 32-bit mode HP-UX

HP (64): 64-bit mode HP-UX

Sol (32): 32-bit mode Solaris

Sol (64): 64-bit mode Solaris

AIX (32): 32-bit mode AIX

AIX (64): 64-bit mode AIX

Linux (32): 32-bit mode Linux

C: The file is created.

NF: The file is created, but the facility that uses that file does not operate.

--: The file is not created.

#1: The underlined portion indicates the HiRDB installation directory.

#2: The suffixes for the shared library files and parsing libraries differ according to the platform. In Solaris and Linux, the suffix is `.so`; in AIX, the suffix is `.a`.

#3: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-4: Files and directories for HiRDB/Developer's Kit (UNIX client in IPF machine)

Name	Directory	File name	Platform
			HP-UX
Header files	/*HiRDB*/include	SQLCA.CBL	C
		SQLCA64.CBL	C
		SQLDA.CBL	C
		SQLDA64.CBL	C
		SQLIOA.CBL	C
		SQLIOA64.CBL	C
		pdbtypes.h	C
		pdberrno.h	C
		pdbmisc.h	C
		pdbmiscm.h	C
		pdbsqlda.h	C
		pdbsqlcsna.h	C
		pddbhash.h	C
		pdauxcnv.h	C
		SQLCAM.cbl	C
		SQLDAM.cbl	C
		SQLIOAM.cbl	C
		SQLIOAMTH.CBL	C
		SQLIOAMTH64.CBL	C
		SQLCAMTH.CBL	C
		SQLCAMTH64.CBL	C
		SQLCSNA.CBL	C
		SQLCSNA64.CBL	C

Name	Directory	File name	Platform
			HP-UX
Shared libraries	/*HiRDB*/client/lib	libzclt.so	C
		libzclt64.so	C
		libzcltx.so	C
		libzcltx64.so	C
		libzclty.so	C
		libzclty64.so	C
		libzcltk.so	C
		libzcltk64.so	C
		libsqlauxf.so	C
		libsqlauxf64.so	C
		libzcltxk.so	NF
		libzcltxk64.so	NF
		libzcltyk.so	NF
		libzcltyk64.so	C
		libzclts.so	C
		libzclts64.so	C
		libzcltxs.so	C
		libzcltxs64.so	C
		libzcltys.so	C
		libzcltys64.so	C
JDBC drivers	/*HiRDB*/client/lib	libjjdbc.so	C
		libjjdbc32.so	C
		pdjdbc.jar	C
		pdjdbc2.jar	C
		pdjdbc32.jar	C

Name	Directory	File name	Platform
			HP-UX
ODBC driver	/*HiRDB*/client/lib	libodbcdrv.sl	--
		libodbcdrv64.sl	--
Command utilities	/*HiRDB*/client/utl	pdcpp	C
		pdocc	C
		pdcbl	C
		pdocb	C
		pdprep	C
		pdtrcmgr	C
	/*HiRDB*/bin	pddef	C
SQLJ	/*HiRDB*/client/lib	pdruntime.jar	C
		pdruntime32.jar	C
		pdnativert.jar	C
		pdnativert32.jar	C
		pdsqlj.jar	C
		pdsqlj32.jar	C
		libpdsqljn.so	C
		libpdsqljn32.so	C
		libpdparse.so	C
	/*HiRDB*/client/utl	pdjava	C
Message object file	/*HiRDB*/lib	msgtxt	C

Name	Directory	File name	Platform
			HP-UX
Syntax analysis libraries	/*HiRDB*/lib/sjis	libasqap.so	C
	/*HiRDB*/lib/chinese		C
	/*HiRDB*/lib/lang-c		C
	/*HiRDB*/lib/ujis		C
	/*HiRDB*/lib/utf-8		C
	/*HiRDB*/lib/chinese-gb18030		C
Sample source files	/*HiRDB*/client/sampleap/uap	CREATE.ec	C
		SAMPLE1.ec	C
		SAMPLE2.ec	C
		SAMPLE3.ec	C
		Sample1.ecb	C
		Sample.mk	C
		inputf1	C
		inputf2	C
XML conversion commands#	/*HiRDB*/client/utl	phdxmlcnv	C
XML conversion libraries#	/*HiRDB*/client/lib	XMLConverter.jar	C

Legend:

C: The file is created.

NF: The file is created, but the facility that uses that file does not operate.

--: The file is not created.

Note

The underlined portion indicates the HiRDB installation directory.

#: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-5: Files and directories for HiRDB/Run Time (UNIX client in IPF machine)

Name	Directory	File name	Platform
			HP-UX
Shared libraries	/*HiRDB*/client/lib	libzclt.so	C
		libzclt64.so	C
		libzcltx.so	C
		libzcltx64.so	C
		libzclty.so	C
		libzclty64.so	C
		libzcltk.so	C
		libzcltk64.so	C
		libsqlauxf.so	C
		libsqlauxf64.so	C
		libzcltxk.so	NF
		libzcltxk64.so	NF
		libzcltyk.so	NF
		libzcltyk64.so	C
		libzclts.so	C
		libzclts64.so	C
		libzcltxs.so	C
		libzcltxs64.so	C
		libzcltys.so	C
		libzcltys64.so	C

Name	Directory	File name	Platform
			HP-UX
JDBC driver	/*HiRDB*/client/lib	libjjdbc.so	C
		libjjdbc32.so	C
		pdjdbc.jar	C
		pdjdbc2.jar	C
		pdjdbc32.jar	C
ODBC driver	/*HiRDB*/client/lib	libodbcdrv.sl	--
		libodbcdrv64.sl	--
SQLJ runtime files	/*HiRDB*/client/lib	pdruntime.jar	C
		pdruntime32.jar	C
		pdnativert.jar	C
		pdnativert32.jar	C
		libpdsqljn.so	C
		libpdsqljn32.so	C
Command utilities	/*HiRDB*/client/utl	pdtrcmgr	C
XML conversion commands[#]	/*HiRDB*/client/utl	phdxmlcnv	C
XML conversion libraries[#]	/*HiRDB*/client/lib	XMLConverter.jar	C

Legend:

C: The file is created.

NF: The file is created, but the facility that uses that file does not operate.

--: The file is not created.

Note

The underlined portion indicates the HiRDB installation directory.

#

These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-6: Files and directories for HiRDB/Developer's Kit (Linux (EM64T))

Name	Directory	File name
Header files	/*HiRDB*/include	SQLCA.CBL
		SQLCA64.CBL
		SQLDA.CBL
		SQLDA64.CBL
		SQLIOA.CBL
		SQLIOA64.CBL
		pdbtypes.h
		pdberrno.h
		pdbmisc.h
		pdbmiscm.h
		pdbsqlda.h
		pdbsqlcsna.h
		pddbhash.h
		pdauxcnv.h
		SQLCAM.cbl
		SQLDAM.cbl
		SQLIOAM.cbl
		SQLIOAMTH.CBL
		SQLIOAMTH64.CBL
		SQLCAMTH.CBL
		SQLCAMTH64.CBL
		SQLCSNA.CBL
		SQLCSNA64.CBL

Name	Directory	File name
Shared libraries	/*HiRDB*/client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzclty64.so
		libzcltys64.so
		libzcltk.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so
		libzcltxk.so
		libzcltyk.so
		libzcltyk64.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
JDBC driver	/*HiRDB*/client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
ODBC driver	/*HiRDB*/client/lib	libodbcdrv.sl
		libodbcdrv64.sl

Name	Directory	File name
Command utilities	/*HiRDB*/client/utl	pdcpp
		pdocc
		pdcbl
		pdocc
		pdprep
		pdtrcmgr
	/*HiRDB*/bin	pddef
		pddivinfgt
SQLJ	/*HiRDB*/client/lib	pdsqlj.jar
		pdruntime.jar
		pdnativert.jar
		libpdparse.so
		libpdsqljn.so
	/*HiRDB*/client/utl	pdjava
Message object file	/*HiRDB*/lib	msgtxt
Parsing libraries	/*HiRDB*/lib/sjis	libasqap.so
	/*HiRDB*/lib/chinese	
	/*HiRDB*/lib/lang-c	
	/*HiRDB*/lib/ujis	
	/*HiRDB*/lib/utf-8	
	/*HiRDB*/lib/chinese-gb18030	

Name	Directory	File name
Sample source files	/*HiRDB*/client/sampleap/uap	CREATE.ec
		SAMPLE1.ec
		SAMPLE2.ec
		SAMPLE3.ec
		Sample1.ecb
		Sample.mk
		inputf1
		inputf2
XML conversion commands#	/*HiRDB*/client/utl	phdxmlcnv
XML conversion libraries#	/*HiRDB*/client/lib	XMLConverter.jar

Note

The underlined portion indicates the HiRDB installation directory.

#: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-7: Files and directories for HiRDB/Run Time (Linux (EM64T))

Name	Directory#	File name
Shared libraries	/*HiRDB*/client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzclty64.so
		libzcltys64.so
		libzcltk.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so
		libzcltxk.so
		libzcltyk.so
		libzcltyk64.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
JDBC driver	/*HiRDB*/client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
ODBC driver	/*HiRDB*/client/lib	libodbcdrv.sl
		libodbcdrv64.sl
SQLJ runtime	/*HiRDB*/client/lib	libpdsqljn.so
		pdruntime.jar
		pdnativert.jar
Command utility	/*HiRDB*/client/utl	pdtrcmgr
	/*HiRDB*/bin	pddivinfgt

Name	Directory#	File name
XML conversion commands#	/*HiRDB*/`client/utl`	phdxmlcnv
XML conversion libraries#	/*HiRDB*/`client/lib`	`XMLConverter.jar`

Note

The underlined portion indicates the HiRDB installation directory.

#: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

■ Archived files and shared library files used for each purpose

Table 6-8 shows the archived files that are used for each purpose. *Table 6-9* shows the shared library files that are used for each purpose.

Table 6-8: Archived files used for each purpose (UNIX client)

Purpose		File used
Normal UAP		`libclt.a`
XA interface connection	Dynamic connection (single thread)	`libcltxa.a`
	Static or dynamic connection (single thread)#1	`libcltya.a`
Multi-connection facility	DCE thread	`libcltm.a`
	Kernel thread	`libcltk.a`
	Solaris thread	`libcltk.a`
	Single thread	`libclts.a`

#: The connection type can be switched between static connection and dynamic connection by the switch registered to TM.

Table 6-9: Shared library files used for each purpose (UNIX client)

Purpose	File used
Normal UAP	`libzclt.sl`

Purpose			File used

XA interface connection	Dynamic connection	Single thread	`libzcltx.sl`
			`libzcltx64.sl`
			`libzcltxs.sl` (when the multi-connection facility is used)
			`libzcltxs64.sl` (when the multi-connection facility is used)

		Multiple threads	`libzcltxk.sl`

	Static or dynamic connection#	Single thread	`libzclty.sl`
			`libzclty64.sl`
			`libzcltys.sl` (when the multi-connection facility is used or `TUXEDO` is supported)
			`libzcltys64.sl` (when the multi-connection facility is used)

		Multiple threads	`libzcltyk.sl` `libzcltyk64.sl`
Multi-connection facility	DCE thread		`libzcltm.sl`
	Kernel thread		`libzcltk.sl`
	Solaris thread		`libzcltk.sl`
	Single thread		`libzclts.sl`
ODBC connection			`libzpodbc.sl`
SQL auxiliary functions			`libsqlauxf.sl`

Note

The suffix of the shared library files differs according to the platform. In Solaris and Linux, the suffix is `.so`; in AIX, the suffix is `.a`.

#: The connection type can be switched between static and dynamic by the switch that is registered to TM.

■ Library files used by each transaction manager

The following table lists the libraries that are used by each transaction manager.

Table 6-10: Library files used by each transaction manager (UNIX client)

Transaction manager	Library name	BES connection holding facility
OpenTP1	`libzcltx.sl`	Y
	`libzclty.sl`	Y
	`libzcltxs.sl`	Y
	`libzcltys.sl`	Y
	`libzcltx64.sl`	Y
	`libzclty64.sl`	Y
	`libzcltxs64.sl`	Y
	`libzcltys64.sl`	Y
TPBroker for C++	`libzcltxk.sl`	N
	`libzcltyk.sl`	N
TUXEDO	`libzcltys.sl`	Y
WebLogic Server	`libzcltyk.sl`	N
TP1/EE	`libzcltyk.sl`	Y
	`libzcltyk64.sl`	Y

Legend:

Y: The BES connection holding facility can be used.

N: The BES connection holding facility cannot be used.

Note

The suffix of the shared library files differs according to the platform. In Solaris and Linux, the suffix is `.so`; in AIX, the suffix is `.a`.

6.4.2 Directories and files for Windows clients

Tables 6-11 through *6-17* list the files and directories that are created automatically during HiRDB client installation.

Table 6-11: Files and directories for HiRDB/Developer's Kit (Windows client)

Name	Directory	File name
Header files	*xxxx*\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		SQLCA.CBL
		SQLIOA.CBL
		PDBMISCM.H
		SQLDA.CBL
		PDBSQLDA.H
		PDBSQLCSNA.H
		SQLIOAD.CBL
		SQLCAD.CBL
		PDDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
		SQLCAMTH.CBL
		SQLCSNA.CBL
Message object file	*xxxx*\LIB	msgtxt

Name	Directory	File name
Linkage libraries	*xxxx*\LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTX32.LIB
		PDCLTXM.LIB
		PDSQLAUXF.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
		PDCLTM71.LIB
		PDCLTM80S.LIB
Command utilities	*xxxx*\UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.EXE
		PDPREP7.EXE
		PDPREP8.EXE
		PDPREPA.EXE
		PDPREPC.EXE
		PDPREPG.EXE
		PDTRCMGR.EXE
		PDCLTADM.EXE

Name	Directory	File name
DLL files	*xxxx*\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTX32.DLL
		PDCLTXM.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF71.DLL
		PDOLEDB.DLL
		PDCLTXS.DLL
		PDCLTXM5.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
JDBC drivers	*xxxx*\UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
SQLJ	*xxxx*\UTL	PDSQLJ.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDPARSE.DLL
		PDJAVA.EXE
		PDSQLJN.DLL
HiRDB.NET data providers	*xxxx*\UTL and \Windows\assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL

Name	Directory	File name
Publisher policies for HiRDB.NET data provider	\Windows\assembly	`policy.9.1.pddndp.dll`
		`policy.9.1.pddndpcore.dll`
		`policy.9.1.pddndp20.dll`
		`policy.9.1.pddndpcore20.dll`
XML conversion commands#	xxxx\UTL	`phdxmlcnv.bat`
XML conversion libraries#	xxxx\UTL	`XMLConverter.jar`
ODBC driver	`\Windows\system32`	`pdodbcdrv3.dll`
		`pdodbstp3.dll`
		`pdclto32.dll`
Interface definition file	*xxxx*`\LIB`	`HIRDB.PKG`
Sample files	*xxxx*`\SAMPLEAP`	`CREATE.EC`
		`SAMPLE1.EC`
		`SAMPLE2.EC`
		`SAMPLE3.EC`
		`SAMPLE1.ECB`
		`INPUTF1`
		`INPUTF2`
README file	*xxxx*	`README.TXT`
Environment definition file	`\WINDOWS`	`HIRDB.INI`

#: These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Notes

1. *xxxx* indicates the name of the directory specified during installation. The directory name can be specified when HiRDB/Developer's Kit is installed. `\Windows` indicates the system directory.
2. This table does not include redistributed files and installer management files.

Table 6-12: Files and directories for HiRDB/Run Time (Windows client)

Name	Directory	File name
Linkage libraries	xxxx\LIB	PDCLTX32.LIB
		PDCLTXM.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
Command utilities	*xxxx*\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTP32.DLL
		PDCLTX32.DLL
		PDTRCMGR.EXE
		PDSQLAUXF.DLL
		PDSQLAUXF71.DLL
		PDCLTXM.DLL
		PDOLEDB.DLL
		PDCLTADM.EXE
		PDCLTXS.DLL
		PDCLTXM5.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
		phdxmlcnv.bat

Name	Directory	File name
		XMLConverter.jar
	xxxx\UTL and \Windows\assembly	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
	\Windows\assembly	policy.9.1.pddndp.dll
		policy.9.1.pddndpcore.dll
		policy.9.1.pddndp20.dll
		policy.9.1.pddndpcore20.dll
ODBC driver	\Windows\system32	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
README file	*xxxx*	README.TXT
Environment definition file	\WINDOWS	HIRDB.INI

Notes

1. *xxxx* indicates the name of the directory specified during installation. The directory name can be specified when HiRDB/Run Time is installed. \Windows indicates the system directory.
2. This table does not include *redistributed* files and installer management files.

Table 6-13: Files and directories for HiRDB/Developer's Kit (Windows client in IPF machine)

Name	Directory	File name
Header files	*xxxx*\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		PDBMISCM.H
		SQLDA.CBL
		PDBSQLDA.H
		PDBSQLCSNA.H
		SQLIOA.CBL
		SQLCA.CBL
		SQLIOAD.CBL
		SQLCAD.CBL
		PDDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
		SQLCAMTH.CBL
		SQLCSNA.CBL
Linkage libraries	*xxxx*\LIB	PDCLTM64.LIB
		PDCLTX64.LIB
		PDCLTXM64.LIB
		PDSQLAUXF64.LIB
		PDCLTXS64.LIB

Name	Directory	File name
Command utilities	*xxxx*\UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.EXE
		PDPREP7.EXE
		PDPREP8.EXE
		PDPREPA.EXE
		PDPREPC.EXE
		PDPREPG.EXE
		PDJAVA.EXE
		PDTRCMGR.EXE
		PDCLTADM.EXE
DLL files	*xxxx*\UTL	PDCLTM64.DLL
		PDCLTX64.DLL
		PDCLTXM64.DLL
		PDSQLAUXF64.DLL
		PDCLTXS64.DLL
JDBC drivers	*xxxx*\UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
SQLJ	*xxxx*\UTL	PDSQLJ.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDPARSE.DLL
		PDSQLJN.DLL

Name	Directory	File name
Sample files	*xxxx*\SAMPLEAP	CREATE.EC
		SAMPLE1.EC
		SAMPLE2.EC
		SAMPLE3.EC
		SAMPLE1.ECB
		INPUTF1
		INPUTF2
README file	*xxxx*	README.TXT
Environment definition file	\Windows	HIRDB.INI
Message object file	xxxx\LIB	msgtxt

Notes

1. *xxxx* indicates the name of the directory specified during installation. The directory name can be specified when HiRDB/Developer's Kit is installed. \Windows indicates the system directory.
2. This table does not include redistributed files and installer management files.

Table 6-14: Files and directories for HiRDB/Run Time (Windows client in IPF machine)

Name	Directory	File name
Command utilities	*xxxx*\UTL	PDCLTM64.DLL
		PDCLTX64.DLL
		PDCLTXM64.DLL
		PDSQLAUXF64.DLL
		PDCLTXS64.DLL
		PDTRCMGR.EXE
		PDCLTADM.EXE
		PDJDBC.JAR
		PDJDBC2.JAR
		JJDBC.DLL
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
README file	*xxxx*	README.TXT
Environment definition file	\Windows	HIRDB.INI

Notes

1. *xxxx* indicates the name of the directory specified during installation. The directory name can be specified when HiRDB/Run Time is installed. `\Windows` indicates the system directory.
2. This table does not include redistributed files and installer management files.

Table 6-15: Files and directories for HiRDB/Developer's Kit (EM64T machine Windows client)

Name	Directory	File name
Header files	*xxxx*\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		PDBMISCM.H
		SQLDA.CBL
		SQLDA64.CBL
		PDBSQLDA.H
		PDBSQLCSNA.H
		SQLIOA.CBL
		SQLIOA64.CBL
		SQLCA.CBL
		SQLCA64.CBL
		SQLIOAD.CBL
		SQLIOAD64.CBL
		SQLCAD.CBL
		SQLCAD64.CBL
		PDDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
		SQLIOAMTH64.CBL
		SQLCAMTH.CBL
		SQLCAMTH64.CBL
		SQLCSNA.CBL
		SQLCSNA64.CBL

Name	Directory	File name
Linkage libraries	*xxxx*\LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTM64.LIB
		PDCLTX32.LIB
		PDCLTXM.LIB
		PDSQLAUXF.LIB
		PDSQLAUXF64.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
		PDCLTM71.LIB
		PDCLTM80S.LIB
Command utilities	*xxxx*\UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.EXE
		PDPREP7.EXE
		PDPREP8.EXE
		PDPREPA.EXE
		PDPREPC.EXE
		PDPREPG.EXE
		PDJAVA.EXE
		PDTRCMGR.EXE
		PDCLTADM.EXE

Name	Directory	File name
DLL files	*xxxx*\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTM64.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTX32.DLL
		PDCLTXM.DLL
		PDOLEDB.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF64.DLL
		PDPARSE.DLL
		PDCLTXS.DLL
JDBC drivers	*xxxx*\UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
SQLJ	*xxxx*\UTL	PDSQLJ.JAR
		PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
HiRDB.NET data provider	*xxxx*\UTL	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
	xxxx\UTL and \Windows\assembly	PDDNDP20x.DLL
		PDDNDPCORE20x.DLL

Name	Directory	File name
Publisher policies for HiRDB.NET data provider	`\Windows\assembly`	`policy.9.1.pddndp20x.dll`
		`policy.9.1.pddndpcore20x.dll`
XML conversion commands#	`xxxx\UTL`	`phdxmlcnv.bat`
XML conversion libraries#	`xxxx\UTL`	`XMLConverter.jar`
ODBC 3.5 drivers	`\Windows\SysWOW64`	`pdodbcdrv3.dll`
		`pdodbstp3.dll`
		`pdclto32.dll`
	`\Windows\system32`	`pdodbcdrv3x.dll`
		`pdodbstp3x.dll`
		`pdclto64.dll`
Interface definition file	*xxxx*`\BIN`	`HIRDB.PKG`
Sample	`xxxx\SAMPLEAP`	`CREATE.EC`
		`SAMPLE1.EC`
		`SAMPLE2.EC`
		`SAMPLE3.EC`
		`SAMPLE1.ECB`
		`INPUTF1`
		`INPUTF2`
README file	*xxxx*	`README.TXT`
Environment definition file	`\Windows`	`HIRDB.INI`
Message object file	`xxxx\LIB`	`msgtxt`

Notes:

1. *xxxx* indicates the name of the HiRDB installation directory. This directory name can be specified during installation. `\Windows` indicates the system directory.
2. This table does not include redistributed files and installer management files.

#

These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-16: Files and directories for HiRDB/Run Time (EM64T machine Windows client)

Name	Directory	File name
Linkage libraries	xxxx\LIB	PDCLTX32.LIB
		PDCLTXM.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
Command utilities	*xxxx*\UTL	PDTRCMGR.EXE
		PDCLTADM.EXE
DLL files	*xxxx*\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTM64.DLL
		PDCLTX32.DLL
		PDCLTM71.DLL
		PDCLTM80S.DLL
		PDCLTXM.DLL
		PDOLEDB.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF64.DLL
		PDPARSE.DLL
		PDCLTXS.DLL
JDBC driver	*xxxx*\UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR

Name	Directory	File name
SQLJ runtime	*xxxx*\UTL	PDRUNTIME.JAR
		PDNATIVERT.JAR
		PDSQLJN.DLL
HiRDB data providers	*xxxx*\UTL	PDDNDP.DLL
		PDDNDPCORE.DLL
		PDDNDP20.DLL
		PDDNDPCORE20.DLL
	xxxx\UTL and \Windows\assembly	PDDNDP20x.DLL
		PDDNDPCORE20x.DLL
Publisher policies for HiRDB.NET data provider	\Windows\assembly	policy.9.1.pddndp20x.dll
		policy.9.1.pddndpcore20x.dll
XML conversion commands#	xxxx\UTL	phdxmlcnv.bat
XML conversion libraries#	xxxx\UTL	XMLConverter.jar
ODBC 3.5 drivers	\Windows\SysWOW64	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
	\Windows\system32	pdodbcdrv3x.dll
		pdodbstp3x.dll
		pdclto64.dll
README files	*xxxx*	README.TXT
Environment definition file	\Windows	HIRDB.INI

Notes:

1. *xxxx* indicates the name of the HiRDB installation directory. This directory name can be specified during installation. \Windows indicates the system directory.
2. This table does not include redistributed files and installer management files.

#

These items are not included in the HiRDB client provided as part of the HiRDB server products because they are included in HiRDB XML Extension.

Table 6-17: Files and directories for ODBC driver (Windows client)

Name	Directory	File name
Setup files	`\Windows`	`DRVSETUP.EXE`#
		`DRVSTP32.EXE`
Setup DLL		`HIRDBSTP.DLL`#
		`HRDSTP32.DLL`
Driver		`PDODBDRV.DLL`#
		`PDODBD32.DLL`
`HiRDB/ClientDLL`		`PDCLTLIB.DLL`#
		`PDCLTL32.DLL`

Note

`\Windows` indicates the system directory.

#: The file is not created on EM64T machines running Windows clients.

■ Linkage library files used by application

The following table lists the linkage library files that are used according to their purpose.

Table 6-18: Linkage library files used according to purpose (Windows client)

Purpose		File used
Usual UAP		`CLTDLL.DLL`
XA interface connection (static connection or dynamic connection)#	Single thread	`PDCLTX32.DLL` `PDCLTXS.DLL` (for OTS or TUXEDO)
	Multi-thread	`PDCLTXM.DLL`
Multi-connection facility (for multi-thread)		`PDCLTM32.DLL` `PDCLTM50.DLL` (for VisualC++5.0) `PDCLTM71.DLL` (when Visual Studio .NET 2003 is supported) `PDCLTM80S.DLL` (when Visual Studio 2005 is supported)
For SQL auxiliary functions		`PDSQLAUXF.DLL`

#: The connection type can be switched between static connection and dynamic connection by the switch registered to TM.

■ Library files used by each transaction manager

The following table lists the libraries that are used by each transaction manager.

Table 6-19: Library files used by each transaction manager (Windows client)

Transaction manager	Library name	BES connection holding facility
OpenTP1	`pdcltx32.dll`	Y
	`pdcltxs.dll`	Y
TPBroker for C++	`pdcltxm.dll`	N
TUXEDO	`pdcltxs.dll`	Y
WebLogic Server	`pdcltxm.dll`	N

Legend:

Y: The BES connection holding facility can be used.

N: The BES connection holding facility cannot be used.

■ List of libraries and compilers

The following table lists the libraries and compilers.

Table 6-20: List of libraries and compilers (Windows client)

Library name	Compiler version	VisualC runtime used
`cltdll.dll`	VisualC++ 2.0	Multi-thread static
`pdcltm32.dll`	VisualC++ 4.2	Multi-thread DLL
`pdcltx32.dll`		
`pdcltxm.dll`		
`pdcltxs.dll`		
`pdcltm50.dll`	VisualC++ 5.0	
`pdcltxm5.dll`		
`pdsqlauxf.dll`		
`pdcltm71.dll`	Visual Studio .NET 2003	Multi-thread static
`pdsqlauxf71.dll`		

Library name	Compiler version	VisualC runtime used
`jjdbcinter.dll`	VisualC++5.0	Multi-thread DLL
`jjdbcinter.dll`	Visual Studio 2003	Multi-thread static
`PDCLTM80S.dll`	Visual Studio 2005	

6.5 Setting the hosts file

When different machines are used for the client and the server, the following information must be specified in the `hosts` file of the client machine.

- IP address
- Host name

If DNS is used, the `hosts` file does not need to be set.

If you do not specify the standard host in the `hosts` file, you must specify `PDCLTRCVADDR` in the client environment definitions.

(1) HiRDB/Single Server

- IP address

  Specify the IP address of the HiRDB/Single Server.

- Host name

  Specify the host name of the HiRDB/Single Server.

■ **System-switching without IP address inheritance**

Specify the IP addresses and the host names of both the execution system and the standby system.

(2) HiRDB/Parallel Server

- IP address

  Specify the IP address of the server machine at which the front-end server is defined.

- Host name

  Specify the host name of the server machine at which the front-end server is defined.

■ **System-switching without IP address inheritance**

Specify the IP addresses and the host names of both the execution system and the standby system.

6.6 Client environment definitions (setting environment variables)

6.6.1 Environment setup format

To execute a UAP, you must specify client environment definitions for each client.

(1) UNIX environment

To execute commands and utilities, add the following directory to the `PATH` environment variable:

- Executing a client on the server machine
  `/opt/HiRDB/client/utl/`

- Logging into the HiRDB server from a remote system
  `$PDDIR/client/utl/`

- Retrieval sequence for client environment definitions

  If the client environment definitions are set at several locations, each client environment definition is retrieved in the sequence below. If a client environment definition has no specified value, the default value is applied.

  1. Environment variables group#
  2. User environment variables

  #: When the multi-connection facility is used, use `ALLOCATE CONNECTION HANDLE` to specify the file name. If a UAP under OLTP is used as the client, specify the file name in an open character string. For details about open character strings, see the *HiRDB Version 9 Installation and Design Guide*.

(a) sh (Bourne shell)

You must store the environment variables shown below in the `.profile` file. These environment variables execute automatically at the time of startup.

```
  $
PDHOST=HiRDB-server-host-name[,secondary-system-HiRDB-server-host-name]
  $ PDNAMEPORT=HiRDB-server-port-number
  $ PDFESHOST=front-end-server-host-name
               [:port-number-of-unit-containing-front-end-server]
                  [,secondary-system-front-end-server-host-name

[:port-number-of-unit-containing-secondary-system-front-end-server]]
  $ PDSERVICEGRP=server-name
  $ PDSRVTYPE={WS|PC}
  $ PDSERVICEPORT=high-speed-connection-port-number
                   [,secondary-system-high-speed-connection-port-number]
  $
PDFESGRP=FES-group[,switchover-FES-group[,switchover-FES-group]...]
  $ PDCLTRCVPORT=client-receive-port-number
  $ PDCLTRCVADDR={client-IP-address|client-host-name}
  $ PDTMID=OLTP-identifier
  $ PDXAMODE={0|1}
  $
PDTXACANUM=maximum-number-of-concurrent-transaction-executions-per-UAP
  $ PDXARCVWTIME=transaction-recovery-wait-time
  $ PDXATRCFILEMODE={LUMP|SEPARATE}
  $ PDXAAUTORECONNECT={YES|NO}
  $
HiRDB_PDHOST=HiRDB-server-host-name[,secondary-system-HiRDB-server-host-
name]
  $ HiRDB_PDNAMEPORT=HiRDB-server-port-number
  $ HiRDB_PDTMID=OLTP-identifier
  $ HiRDB_PDXAMODE={0|1}
  $ PDUSER=[authorization-identifier/password]
  $ PDCLTAPNAME=identification-name-of-UAP-to-be-executed
  $ PDCLTLANG={SJIS|CHINESE|UJIS|C|UTF-8|CHINESE-GB18030}
  $ PDLANG={UTF-8|SJIS|CHINESE|CHINESE-GB18030|ANY}
  $ PDDBLOG={ALL|NO}
  $ PDEXWARN={YES|NO}
  $ PDSUBSTRLEN={3|4|5|6}
  $ PDCLTCNVMODE={AUTO|NOUSE|UJIS|UJIS2|UTF8|UTF8MS|
   UTF8_TXT|UTF8_EX|UTF8_EX2|UTF8MS_TXT|UCS2_UJIS|
   UCS2_UTF8}
  $ PDCLTGAIJIDLL=user-defined-external-character-conversion-DLL-file-name
  $ PDCLTGAIJIFUNC=user-defined-external-character-conversion-function-name
  $ PDCLTGRP=client-group-name
  $ PDTCPCONOPT={0|1}
  $ PDAUTORECONNECT={YES|NO}
  $ PDRCCOUNT=CONNECT-retry-count-with-automatic-reconnect-facility
  $ PDRCINTERVAL=CONNECT-retry-interval-with-automatic-reconnect-facility
  $ PDUAPENVFILE=UAP-environment-definition-file-name
  $ PDDBBUFLRU={YES|NO}
  $ PDHATRNQUEUING=NO
  $ PDASTHOST=HiRDB-Control-Manager-Agent-host-name
              [,secondary-system-HiRDB-Control-Manager-Agent-host-name]
  $ PDASTPORT=HiRDB-Control-Manager-Agent-port-number
  $
PDSYSTEMID=HiRDB-identifier-of-HiRDB-server-managed-by-HiRDB-Control-Ma
nager-Agent
  $ PDASTUSER=OS-user-name/password
  $ PDCMDWAITTIME=maximum-client-wait-time-during-command-execution
  $ PDCMDTRACE=command-trace-file-size
  $ PDIPC={MEMORY|DEFAULT}
  $ PDSENDMEMSIZE=data-send-memory-size-in-client
  $ PDRECVMEMSIZE=data-receive-memory-size-in-client
  $ PDCWAITTIME=maximum-client-wait-time
  $ PDSWAITTIME=maximum-server-wait-time-during-transaction-processing
  $ PDSWATCHTIME=maximum-server-wait-time-outside-transaction-processing
  $ PDCWAITTIMEWRNPNT=output-time-for-SQL-runtime-warning
  $ PDKALVL={0|1|2}
  $ PDKATIME=packet-send-interval
  $ PDTIMEDOUTRETRY=retry-count
  $
PDNBLOCKWAITTIME=connection-establishment-monitoring-time-in-nonblock-mode
  $
PDCONNECTWAITTIME=maximum-wait-time-in-HiRDB-client-during-server-connect
ion
  $ PDCLTPATH=trace-file-storage-directory
  $ PDSQLTRACE=SQL-trace-file-size
  $ PDUAPERLOG=client-error-log-file-size
  $ PDERRSKIPCODE=SQLCODE[,SQLCODE]...
  $ PDPRMTRC={YES|NO|IN|OUT|INOUT}
  $ PDPRMTRCSIZE=
maximum-data-length-of-parameter-information-output-to-SQL-trace
  $ PDTRCMODE={ERR|NONE}
  $ PDUAPREPLVL={ [s] [u] [p] [r] |a}
  $ PDREPPATH=storage-directory-for-UAP-statistical-report-files
  $ PDTRCPATH=storage-directory-for-dynamic-SQL-trace-files
  $ PDSQLTRCOPENMODE={CNCT|SQL}
  $ PDSQLTEXTSIZE=SQL-statement-size
  $ PDSQLEXECTIME={YES|NO}
  $ PDRCTRACE=reconnect-trace-file-size
  $
PDWRTLNPATH=storage-directory-for-files-to-which-WRITE-LINE-statement-value-e
xpression-values-are-output
  $
PDWRTLNFILSZ=maximum-size-of-output-files-for-WRITE-LINE-statement-value-ex
pression-values
  $ PDWRTLNCOMSZ=total-size-of-WRITE-LINE-statement-value-expression-values
  $ PDUAPEXERLOGUSE={YES|NO}
  $ PDUAPEXERLOGPRMSZ=maximum-data-length-of-parameter-information
  $ PDARYERRPOS={YES|NO}
  $ PDVWOPTMODE={0|1|2}
  $ PDTAAPINFPATH=access-path-information-file-output-directory-name
  $ PDTAAPINFMODE={0|1}
  $ PDTAAPINFSIZE=access-path-information-file-size
  $ PDSTJTRNOUT={YES|NO}
  $ PDLOCKLIMIT=maximum-locked-resource-request-count-per-user
  $ PDDLKPRIO={96|64|32}
  $ PDLOCKSKIP={YES|NO}
  $ PDFORUPDATEEXLOCK={YES|NO}
  $ PDISLLVL=data-guarantee-level
  $ PDSQLOPTLVL=SQL-optimization-option[,SQL-optimization-option]...
  $ PDADDITIONALOPTLVL=SQL-extension-optimizing-option
                        [,SQL-extension-optimizing-option]...
  $
PDHASHTBLSIZE=hash-table-size-when-hash-join-or-subquery-hash-execution-is-a
pplied
  $ PDDFLNVAL={USE|NOUSE}
  $ PDAGGR=group-count-resulting-from-grouping
  $ PDCMMTBFDDL={YES|NO}
  $ PDPRPCRCLS={YES|NO}
  $ PDAUTOCONNECT={ON|OFF}
  $ PDDDLDEAPRPEXE={YES|NO}
  $ PDDDLDEAPRP={YES|NO}
  $ PDLCKWAITTIME=lock-release-wait-time
  $ PDCURSORLVL={0|1|2}
  $ PDDELRSVWDFILE=SQL-reserved-word-deletion-file-name
  $ PDHJHASHINGMODE={TYPE1|TYPE2}
  $ PDCALCMDWAITTIME=maximum-wait-time-for-CAL-COMMAND-statement
  $ PDSTANDARDSQLSTATE={YES|NO}
  $ PDBLKF=block-transfer-row-count
  $ PDBINARYBLKF={YES|NO}
  $ PDBLKBUFFSIZE=communication-buffer-size
  $ PDBLKFUPD={YES|NO}
  $ PDBLKFERRBREAK={YES|NO}
  $ PDNODELAYACK={YES|NO}
  $ PDBINDRETRYCOUNT=bind-system-call-retries-count
  $ PDBINDRETRYINTERVAL=bind-system-call-retry-interval
  $ PDCLTSIGPIPE={CATCH|IGNORE}
  $ PDDBACCS=generation-number-of-RDAREA-to-be-accessed
  $ PDDBORGUAP={YES|NO}
  $ PDSPACELVL={0|1|3}
  $ PDCLTRDNODE=XDM/RD-E2-database-identifier
  $ PDTP1SERVICE={YES|NO}
  $ PDCNSTRNTNAME={LEADING|TRAILING}
  $ PDBESCONHOLD={YES|NO}
  $ PDBESCONHTI=BES-connection-holding-period
  $ PDODBSTATCAHE={0|1}
  $ PDODBESCAPE={0|1}
  $ PDGDATAOPT={YES|NO}
  $ PDODBLOCATOR={YES|NO}
  $ PDODBSPLITSIZE=partition-acquisition-size
  $ PDODBCWRNSKIP={YES|NO}
  $ PDJETCOMPATIBLE={YES|NO}
  $ PDPLGIXMK={YES|NO}
  $ PDPLGPFSZ=initial-size-of-delayed-batch-creation-index-information-file
  $ 
PDPLGPFSZEXP=extension-value-of-delayed-batch-creation-index-information-file

  $ export PDHOST PDNAMEPORT PDFESHOST PDSERVICEGRP PDSRVTYPE
        PDSERVICEPORT PDFESGRP PDCLTRCVPORT PDCLTRCVADDR PDTMID 
PDXAMODE
           PDTXACANUM PDXARCVWTIME PDXATRCFILEMODE 
PDXAAUTORECONNECT PDUSER
           PDCLTAPNAME PDCLTLANG PDLANG PDDBLOG PDEXWARN 
PDSUBSTRLEN
           PDCLTCNVMODE PDCLTGAIJIDLL PDCLTGAIJUFUNC PDCLTGRP 
PDTCPCONOPT
           PDAUTORECONNECT PDRCCOUNT PDRCINTERVAL PDUAPENVFILE 
PDDBBUFLRU
         PDHATRNQUEUING PDASTHOST PDASTPORT PDSYSTEMID PDASTUSER
           PDCMDWAITTIME PDCMDTRACE PDIPC PDSENDMEMSIZE
           PDRECVMEMSIZE PDCWAITTIME PDSWAITTIME PDSWATCHTIME
           PDCWAITTIMEWRNPNT PDKALVL PDKATIME PDTIMEDOUTRETRY
        PDNBLOCKWAITTIME PDCONNECTWAITTIME PDCLTPATH PDSQLTRACE
           PDUAPERLOG PDERRSKIPCODE PDPRMTRC PDPRMTRCSIZE 
PDTRCMODE
           PDUAPREPLVL PDREPPATH PDTRCPATH PDSQLTRCOPENMODE 
PDSQLTEXTSIZE
           PDSQLEXECTIME PDRCTRACE PDWRTLNPATH PDWRTLNFILSZ 
PDWRTLNCOMSZ
           PDUAPEXERLOGUSE PDUAPEXERLOGPRMSZ PDARYERRPOS 
PDVWOPTMODE
          PDTAAPINFPATH PDTAAPINFMODE PDTAAPINFSIZE PDSTJTRNOUT
           PDLOCKLIMIT PDDLKPRIO PDLOCKSKIP PDFORUPDATEEXLOCK 
PDISLLVL
        PDSQLOPTLVL PDADDITIONALOPTLVL PDHASHTBLSIZE PDDFLNVAL 
PDAGGR
           PDCMMTBFDDL PDPRPCRCLS PDAUTOCONNECT PDDDLDEAPRPEXE
           PDDDLDEAPRP PDLCKWAITTIME PDCURSORLVL
           PDDELRSVWDFILE PDHJHASHINGMODE PDCALCMDWAITTIME
          PDSTANDARDSQLSTATE PDBLKF PDBINARYBLKF PDBLKBUFFSIZE 
PDBLKFUPD
           PDBLKFERRBREAK PDNODELAYACK PDBINDRETRYCOUNT
           PDBINDRETRYINTERVAL PDCLTSIGPIPE PDDBACCS
           PDDBORGUAPPDSPACELVL PDCLTRDNODE
           PDTP1SERVICE PDCNSTRNTNAME PDBESCONHOLD PDBESCONHTI
           PDODBSTATCAHE PDODBESCAPE PDGDATAOPT PDODBLOCATOR 
PDODBSPLITSIZE
           PDODBCWRNSKIP PDJETCOMPATIBLE PDPLGIXMK PDPLGPFSZ 
PDPLGPFSZEXP
```

(b) csh (C shell)

You must store the environment variables shown below in the `.login` or `.cshrc` file. These environment variables execute automatically at the time of startup.

```
  % setenv PDHOST HiRDB-server-host-name
                  [,secondary-system-HiRDB-server-host-name]
  % setenv PDNAMEPORT HiRDB-server-port-number
  % setenv PDFESHOST front-end-server-host-name
              [:port-number-of-unit-containing-front-end-server]
                [,secondary-system-front-end-server-host-name
             [:host-name-of-unit-containing-secondary-system-front-end-server]]
  % setenv PDSERVICEGRP server-name
  % setenv PDSRVTYPE {WS|PC}
  % setenv PDSERVICEPORT high-speed-connection-port-number
                   [,secondary-system-high-speed-connection-port-number]
  % setenv PDFESGRP
FES-group[,switchover-FES-group[,switchover-FES-group]...]
  % setenv PDCLTRCVPORT client-receive-port-number
  % setenv PDCLTRCVADDR {client-IP-address
                         |client-host-name}
  % setenv PDTMID OLTP-identifier
  % setenv PDXAMODE {0|1}
  % setenv PDTXACANUM
maximum-number-of-concurrent-transaction-executions-per-UAP
  % setenv PDXARCVWTIME transaction-recovery-wait-time
  % setenv PDXATRCFILEMODE {LUMP|SEPARATE}
  % setenv PDXAAUTORECONNECT {YES|NO}
  % setenv HiRDB_PDHOST HiRDB-server-host-name
                         [,secondary-system-HiRDB-server-host-name]
  % setenv HiRDB_PDNAMEPORT HiRDB-server-port-number
  % setenv HiRDB_PDTMID OLTP-identifier
  % setenv HiRDB_PDXAMODE {0|1}
  % setenv PDUSER authorization-identifier/password
  % setenv PDCLTAPNAME identification-name-of-UAP-to-be-executed
  % setenv PDCLTLANG
{SJIS|CHINESE|UJIS|C|UTF-8|CHINESE-GB18030}
  % setenv PDLANG {UTF-8|SJIS|CHINESE|CHINESE-GB18030|ANY}
  % setenv PDDBLOG {ALL|NO}
  % setenv PDEXWARN {YES|NO}
  % setenv PDCLTCNVMODE
{AUTO|NOUSE|UJIS|UJIS2|UTF8|UTF8MS|UTF8_TXT|
  UTF8_EX|UTF8_EX2|UTF8MS_TXT|UCS2_UJIS|UCS2_UTF8}
  % setenv PDCLTGAIJIDLL
user-defined-external-character-conversion-DLL-file-name
  % setenv PDCLTGAIJIFUNC
user-defined-external-character-conversion-function-name
```

```
  % setenv PDCLTGRP client-group-name
  % setenv PDTCPCONOPT {0|1}
  % setenv PDAUTORECONNECT {YES|NO}
  % setenv PDRCCOUNT CONNECT-retry-count-with-automatic-reconnect-facility
  % setenv PDRCINTERVAL
CONNECT-retry-interval-with-automatic-reconnect-facility
  % setenv PDUAPENVFILE UAP-environment-definition-file-name
  % setenv PDDBBUFLRU {YES|NO}
  % setenv PDHATRNQUEUING NO
  % setenv PDASTHOST HiRDB-Control-Manager-Agent-host-name
              [,secondary-system-HiRDB-Control-Manager-Agent-host-name]
  % setenv PDASTPORT HiRDB-Control-Manager-Agent-port-number
  % setenv PDSYSTEMID
HiRDB-identifier-of-HiRDB-server-managed-by-HiRDB-Control-Manager-Agent
  % setenv PDASTUSER [OS-user-name/password]
  % setenv PDCMDWAITTIME
maximum-client-wait-time-during-command-execution
  % setenv PDCMDTRACE command-trace-file-size
  % setenv PDIPC {MEMORY|DEFAULT}
  % setenv PDSENDMEMSIZE data-send-memory-size-in-client
  % setenv PDRECVMEMSIZE data-receive-memory-size-in-client
  % setenv PDCWAITTIME maximum-client-wait-time
  % setenv PDSWAITTIME
maximum-server-wait-time-during-transaction-processing
  % setenv PDSWATCHTIME
maximum-server-wait-time-outside-transaction-processing
  % setenv PDCWAITTIMEWRNPNT output-time-for-SQL-runtime-warning
  % setenv PDKALVL {0|1|2}
  % setenv PDKATIME packet-send-interval
  % setenv PDTIMEDOUTRETRY retry-count
  % setenv PDNBLOCKWAITTIME
connection-establishment-monitoring-time-in-nonblock-mode
  % setenv PDCONNECTWAITTIME
maximum-wait-time-in-HiRDB-client-during-server-connection
  % setenv PDCLTPATH trace-file-storage-directory
  % setenv PDSQLTRACE SQL-trace-file-size
  % setenv PDUAPERLOG client-error-log-file-size
  % setenv PDERRSKIPCODE SQLCODE[,SQLCODE]...
  % setenv PDPRMTRC {YES|NO|IN|OUT|INOUT}
  % setenv PDPRMTRCSIZE=
maximum-data-length-of-parameter-information-output-to-SQL-trace
  % setenv PDTRCMODE {ERR|NONE}
  % setenv PDUAPREPLVL {[s][u][p][r]|a}
  % setenv PDREPPATH storage-directory-for-UAP-statistical-report-files
  % setenv PDTRCPATH storage-directory-for-dynamic-SQL-trace-files
  % setenv PDSQLTRCOPENMODE {CNCT|SQL}
  % setenv PDSQLTEXTSIZE SQL-statement-size
  % setenv PDSQLEXECTIME {YES|NO}
  % setenv PDRCTRACE reconnect-trace-file-size
  % setenv PDWRTLNPATH
storage-directory-for-files-to-which-WRITE-LINE-statement-value-expression-values-are-output
  % setenv PDWRTLNFILSZ
maximum-size-of-output-files-for-WRITE-LINE-statement-value-expression-values
  % setenv PDWRTLNCOMSZ
total-size-of-WRITE-LINE-statement-value-expression-values
  % setenv PDUAPEXERLOGUSE {YES|NO}
  % setenv PDUAPEXERLOGPRMSZ
maximum-data-length-of-parameter-information
  % setenv PDARYERRPOS{YES|NO}
  % setenv PDVWOPTMODE {0|1|2}
  % setenv PDTAAPINFPATH access-path-information-file-output-directory-name
  % setenv PDTAAPINFMODE {0|1}
  % setenv PDTAAPINFSIZE access-path-information-file-size
  % setenv PDSTJTRNOUT {YES|NO}
  % setenv PDLOCKLIMIT maximum-locked-resource-request-count-per-user
  % setenv PDDLKPRIO {96|64|32}
  % setenv PDLOCKSKIP {YES|NO}
  % setenv PDFORUPDATEEXLOCK {YES|NO}
  % setenv PDISLLVL data-guarantee-level
  % setenv PDSQLOPTLVL
SQL-optimization-option[,SQL-optimization-option]...
  % setenv PDADDITIONALOPTLVL SQL-extension-optimizing-option
                                 [,SQL-extension-optimizing-option]...
  % setenv PDHASHTBLSIZE
hash-table-size-when-hash-join-or-subquery-hash-execution-is-applied
  % setenv PDDFLNVAL {USE|NOUSE}
  % setenv PDAGGR group-count-resulting-from-grouping
  % setenv PDCMMTBFDDL {YES|NO}
  % setenv PDPRPCRCLS {YES|NO}
  % setenv PDAUTOCONNECT {ON|OFF}
  % setenv PDDDLDEAPRPEXE {ON|OFF}
  % setenv PDDDLDEAPRP {YES|NO}
  % setenv PDLCKWAITTIME lock-release-wait-time
  % setenv PDCURSORLVL {0|1|2}
  % setenv PDDELRSVWDFILE SQL-reserved-word-deletion-file-name
  % setenv PDHJHASHINGMODE {TYPE1|TYPE2}
  % setenv PDCALCMDWAITTIME
maximum-wait-time-for-CAL-COMMAND-statement
  % setenv PDSTANDARDSQLSTATE {YES|NO}
  % setenv PDBLKF block-transfer-row-count
  % setenv PDBINARYBLKF {YES|NO}
  % setenv PDBLKBUFFSIZE communication-buffer-size
  % setenv PDBLKFUPD {YES|NO}
  % setenv PDBLKFERRBREAK {YES|NO}
  % setenv PDNODELAYACK {YES|NO}
  % setenv PDBINDRETRYCOUNT bind-system-call-retries-count
  % setenv PDBINDRETRYINTERVAL bind-system-call-retry-interval
  % setenv PDCLTSIGPIPE {CATCH|IGNORE}
  % setenv PDDBACCS generation-number-of-RDAREA-to-be-accessed
  % setenv PDDBORGUAP {YES|NO}
  % setenv PDSPACELVL {0|1|3}
  % setenv PDCLTRDNODE XDM/RD-E2-database-identifier
  % setenv PDTP1SERVICE {YES|NO}
  % setenv PDCNSTRNTNAME {LEADING|TRAILING}
  % setenv PDBESCONHOLD {YES|NO}
  % setenv PDBESCONHTI BES-connection-holding-period
  % setenv PDODBSTATCAHE {0|1}
  % setenv PDODBESCAPE {0|1}
  % setenv PDGDATAOPT {YES|NO}
  % setenv PDODBLOCATOR {YES|NO}
  % setenv PDODBSPLITSIZE partition-acquisition-size
  % setenv PDODBCWRNSKIP {YES|NO}
  % setenv PDJETCOMPATIBLE {YES|NO}
  % setenv PDPLGIXMK {YES|NO}
  % setenv PDPLGPFSZ
initial-size-of-delayed-batch-creation-index-information-file
  % setenv PDPLGPFSZEXP
extension-value-of-delayed-batch-creation-index-information-file
```

Notes on the UNIX environment

- The environment variables are required for preprocessing. For details about preprocessing, see *8.2 Preprocessing*.
- When you use the Type4 JDBC driver, client environment definitions set using this method are not valid.
- Client environment definitions that begin with PDJDB are not valid when they are set using this method.

(2) Windows environment

If you have selected in a Windows environment to set environment variables during installation, the directory is set in the `PATH` environment variable. However, the directory may not be set automatically if the path name is too long or if you do not have write privileges for `PATH`. You should therefore check whether the directory has been set to `PATH`. If the directory has not been set, you must add the following directory to `PATH`. *xxxx* indicates the directory name in which the HiRDB client is installed.

xxxx`\UTL`

Set the environment variables as system environment variables or user environment variables, or store them in the `HiRDB.INI` file in the Windows directory. If you are using a function in the UAP to set the environment variables, use the `SetEnvironmentVariable` function. Do not use the `putenv` function.

- Retrieval sequence for client environment definitions

  If the client environment definitions are set at several locations, each client environment definition is retrieved in the sequence below. If a client environment definition has no specified value, the default value is applied.

  1. Environment variables group#
  2. User environment variables
  3. `HIRDB.ini`

  #: When the multi-connection facility is used, use `ALLOCATE CONNECTION HANDLE` to specify the group name or file name. If a UAP under OLTP is used as the client, specify the group name or file name in an open character string. For details about open character strings, see the *HiRDB Version 9 Installation and Design Guide*.

A specification example of the `HiRDB.INI` file is shown below.

```
  [HIRDB]

PDHOST=HiRDB-server-host-name[,secondary-system-HiRDB-server-host-name]
  PDNAMEPORT=HiRDB-server-port-number
  PDFESHOST=front-end-server-host-name
            [:port-number-of-unit-containing-front-end-server]
              [,secondary-system-front-end-server-host-name

[:port-number-of-unit-containing-secondary-system-front-end-server]]
  PDSERVICEGRP=server-name
  PDSRVTYPE={WS|PC}
  PDSERVICEPORT=high-speed-connection-port-number
              [,secondary-system-high-speed-connection-port-number]
  PDFESGRP=FES-group[,switchover-FES-group[,switchover-FES-group]...]
  PDCLTRCVPORT=client-receive-port-number
  PDCLTRCVADDR={client-IP-address|client-host-name}
  PDXATRCFILEMODE={LUMP|SEPARATE}
  PDUSER= [authorization-identifier/password]
  PDCLTAPNAME=identification-name-of-UAP-to-be-executed
  PDCLTLANG={SJIS|CHINESE|UJIS|C|UTF-8|CHINESE-GB18030}
  PDLANG={UTF-8|SJIS|CHINESE|CHINESE-GB18030|ANY}
  PDDBLOG={ALL|NO}
  PDEXWARN={YES|NO}
```

PDSUBSTRLEN={3|4|5|6}

PDCLTCNVMODE={AUTO|NOUSE|UJIS|UJIS2|UTF8|UTF8MS|
UTF8_TXT|UTF8_EX|UTF8_EX2|UTF8MS_TXT|UCS2_UJIS|
UCS2_UTF8}

PDCLTGAIJIDLL=*user-defined-external-character-conversion-DLL-file-name*

PDCLTGAIJIFUNC=*user-defined-external-character-conversion-function-name*

PDCLTGRP=*client-group-name*

PDTCPCONOPT={0|1}

PDAUTORECONNECT={YES|NO}

PDRCCOUNT=*CONNECT-retry-count-with-automatic-reconnect-facility*

PDRCINTERVAL=*CONNECT-retry-interval-with-automatic-reconnect-facility*

PDUAPENVFILE=*UAP-environment-definition-file-name*

PDDBBUFLRU={YES|NO}

PDHATRNQUEUING=NO

PDCLTBINDLOOPBACKADDR={YES|NO}

PDASTHOST=*HiRDB-Control-Manager-Agent-host-name*
[,*secondary-system-HiRDB-Control-Manager-Agent-host-name*]

PDASTPORT=*HiRDB-Control-Manager-Agent-port-number*

PDSYSTEMID=*HiRDB-identifier-of-HiRDB-server-managed-by-HiRDB-Control Manager-Agent*

PDASTUSER=*OS-user-name*/*password*

PDCMDWAITTIME=*maximum-client-wait-time-during-command-execution*

PDCMDTRACE=*command-trace-file-size*

PDIPC={MEMORY|DEFAULT}

PDSENDMEMSIZE=*data-send-memory-size-in-client*

PDRECVMEMSIZE=*data-receive-memory-size-in-client*

PDCWAITTIME=*maximum-client-wait-time*

PDSWAITTIME=*maximum-server-wait-time-during-transaction-processing*

PDSWATCHTIME=*maximum-server-wait-time-outside-transaction-processing*

PDCWAITTIMEWRNPNT=*output-timing-for-SQL-runtime-warning*

PDKALVL={0|1|2}

PDKATIME=*packet-send-interval*

PDTIMEDOUTRETRY=*retry-count*

PDNBLOCKWAITTIME=*connection-establishment-monitoring-time-in-nonblock-mode*

PDCONNECTWAITTIME=*maximum-wait-time-in-HiRDB-client-during-server-connection*

PDCLTPATH=*trace-file-storage-directory*

PDSQLTRACE=*SQL-trace-file-size*

PDUAPERLOG=*client-error-log-file-size*

PDERRSKIPCODE=*SQLCODE*[,*SQLCODE*]...

PDPRMTRC={YES|NO|IN|OUT|INOUT}

PDPRMTRCSIZE=
maximum-data-length-of-parameter-information-output-to-SQL-trace

PDTRCMODE={ERR|NONE}
PDUAPREPLVL={[s][u][p][r]|a}
PDREPPATH=*storage-directory-for-UAP-statistical-report-files*
PDTRCPATH=*storage-directory-for-dynamic-SQL-trace-files*
PDSQLTRCOPENMODE={CNCT|SQL}
PDSQLTEXTSIZE=*SQL-statement-size*
PDSQLEXECTIME={YES|NO}
PDRCTRACE=*reconnect-trace-file-size*

PDWRTLNPATH=*storage-directory-for-files-to-which-WRITE-LINE-statement-value-expression-values-are-output*

PDWRTLNFILSZ=*maximum-size-of-output-files-for-WRITE-LINE-statement-value-expression-values*
PDWRTLNCOMSZ=*total-size-of-WRITE-LINE-statement-value-expression-values*
PDUAPEXERLOGUSE={YES|NO}
PDUAPEXERLOGPRMSZ=*maximum-data-length-of-parameter-information*
PDARYERRPOS={YES|NO}
PDDNDPTRACE=*method-trace-file-size*
PDVWOPTMODE={0|1|2}
PDTAAPINFPATH=*access-path-information-file-output-directory-name*
PDTAAPINFMODE={0|1}
PDTAAPINFSIZE=*access-path-information-file-size*
PDSTJTRNOUT={YES|NO}
PDLOCKLIMIT=*maximum-locked-resource-request-count-per-user*
PDDLKPRIO={96|64|32}
PDLOCKSKIP={YES|NO}
PDFORUPDATEEXLOCK={YES|NO}
PDISLLVL=*data-guarantee-level*
PDSQLOPTLVL=*SQL-optimization-option*[,*SQL-optimization-option*]...
PDADDITIONALOPTLVL=*SQL-extension-optimizing-option*
[,*SQL-extension-optimizing-option*]...

PDHASHTBLSIZE=*hash-table-size-when-hash-join-or-subquery-hash-execution-is-applied*
PDDFLNVAL={USE|NOUSE}
PDAGGR=*group-count-resulting-from-grouping*
PDCMMTBFDDL={YES|NO}
PDPRPCRCLS={YES|NO}
PDAUTOCONNECT={ON|OFF}
PDDDLDEAPRPEXE={YES|NO}
PDDDLDEAPRP={YES|NO}
PDLCKWAITTIME=*lock-release-wait-time*
PDCURSORLVL={0|1|2}
PDDELRSVWDFILE=*SQL-reserved-word-deletion-file-name*
PDHJHASHINGMODE={TYPE1|TYPE2}

PDCALCMDWAITTIME=*maximum-wait-time-for-CAL-COMMAND-statement*

PDSTANDARDSQLSTATE={YES|NO}

PDBLKF=*block-transfer-row-count*

PDBINARYBLKF={YES|NO}

PDBLKBUFFSIZE=*communication-buffer-size*

PDBLKFUPD={YES|NO}

PDBLKFERRBREAK={YES|NO}

PDNODELAYACK={YES|NO}

PDBINDRETRYCOUNT=*bind-system-call-retries-count*

PDBINDRETRYINTERVAL=*bind-system-call-retry-interval*

PDDBACCS=*generation-number-of-RDAREA-to-be-accessed*

PDDBORGUAP={YES|NO}

PDSPACELVL={0|1|3}

PDCLTRDNODE=*XDM/RD-E2-database-identifier*

PDTP1SERVICE={YES|NO}

PDRDCLTCODE={SJIS|UTF-8}

PDCNSTRNTNAME={LEADING|TRAILING}

PDBESCONHOLD={YES|NO}

PDBESCONHTI=*BES-connection-holding-period*

PDODBSTATCAHE={0|1}

PDODBESCAPE={0|1}

PDGDATAOPT={YES|NO}

PDODBLOCATOR={YES|NO}

PDODBSPLITSIZE=*partition-acquisition-size*

PDODBCWRNSKIP={YES|NO}

PDJETCOMPATIBLE={YES|NO}

PDPLGIXMK={YES|NO}

PDPLGPFSZ=*initial-size-of-delayed-batch-creation-index-information-file*

PDPLGPFSZEXP=*extension-value-of-delayed-batch-creation-index-information-file*

Notes on the Windows environment

- The environment variables are required for preprocessing. For details about how to preprocess, see *8.2 Preprocessing*.
- When you use the Type4 JDBC driver, client environment definitions set using this method are not valid.
- Client environment definitions that begin with PDJDB are not valid when they are set using this method.

6.6.2 Specifications for using a UAP under OLTP as the client

(1) Using a UAP under OpenTP1 as the client

For the operation mode in which a UAP under OpenTP1 is used as the client, specify the client environment definitions in the system service definitions for OpenTP1. The environment variables are specified in the following OpenTP1 definitions:

- **System environment**

  When a common specification is made for all environment variables
- **Transaction service**

  When a specification related to recovery control of a transaction error is made
- **User service default**

  When a common specification is made for all UAPs
- **Individual user service**

  When separate specifications are made for individual UAPs

The table below shows the OpenTP1 definitions in which the environment variables are specified.

The `putenv` format is used to specify environment variables.

Table 6-21: OpenTP1 definitions in which the environment variables are specified

Environment variable	System environment definition	Transaction service definition	User service default definition	User service definition
`HiRDB_PDHOST`[#9]	M[#1]	N	N	N
`HiRDB_PDNAMEPORT`[#9]	M[#2]	N	N	N
`HiRDB_PDTMID`[#9]	O[#3, #4]	N	N	N
`HiRDB_PDXAMODE`[#9]	O[#5]	N	N	N
`PDHOST`	N	M[#1, #6]	M[#1, #6]	O[#1, #6, #7]
`PDNAMEPORT`	N	M[#2, #6]	M[#2, #6]	O[#2, #6, #7]
`PDTMID`[#9]	N	O[#3, #4, #6]	O[#3, #4, #6]	O[#3, #4, #6, #7]
`PDXAMODE`[#9, #10]	N	O[#5, #6]	O[#5, #6]	O[#5, #6, #7]
`PDTXACANUM`[#9]	N	O	O	O
`PDCLTPATH`	N	O	O	O
`PDUSER`	N	N	M	M
`PDCWAITTIME`	N	O	O	O
`PDSWAITTIME`	N	M	M	M

Environment variable	System environment definition	Transaction service definition	User service default definition	User service definition
PDSQLTRACE	N	O	O	O
PDUAPERLOG	N	O	O	O
PDCLTAPNAME	N	O	O[#8]	O[#8]
PDSWATCHTIME	N	N	M	M
PDTRCMODE	N	O	O	O
PDUAPREPLVL	N	O	O	O
PDREPPATH	N	O	O	O
PDTRCPATH	N	O	O	O
PDSQLTRCOPENMODE	N	O	O	O
PDAUTOCONNECT	N	N	N	N
PDXARCVWTIME[#9]	N	O	N	N
PDCWAITTIMEWRNPNT	N	O	O	O
PDTCPCONOPT	N	O	O	O
PDAUTORECONNECT	N	N	N	N
PDRCCOUNT	N	N	N	N
PDRCINTERVAL	N	N	N	N
PDKALVL	N	N	N	N
PDKATIME	N	N	N	N
PDSQLTEXTSIZE	N	O	O	O
PDSQLEXECTIME	N	O	O	O
PDRCTRACE	N	N	N	N
Other environment variable	N	N	O	O

M: Required.

O: Optional; specify as needed.

N: Not required.

Note

For details about the OpenTP1 system service definitions, see the manual *OpenTP1 System Definition*.

#1: When HiRDB_PDHOST is specified, it is not necessary to specify PDHOST because the value specified in HiRDB_PDHOST is assumed for PDHOST. However, if HiRDB_PDHOST is not specified, PDHOST must be specified. If PDHOST and HiRDB_PDHOST are both specified, HiRDB_PDHOST takes precedence.

When PDHOST is specified in an environment variable group, the PDHOST specification of the environment variable group becomes effective.

For guidelines on the value to be specified in PDHOST, see *(7) Fixing the communication-target server by specifying the PDFESHOST name in PDHOST (limited to HiRDB/Parallel Server)*.

#2: When HiRDB_PDNAMEPORT is specified, it is not necessary to specify PDNAMEPORT because the value specified in HiRDB_PDNAMEPORT is assumed for PDNAMEPORT. However, if HiRDB_PDNAMEPORT is not specified, PDNAMEPORT must be specified. If PDNAMEPORT and HiRDB_PDNAMEPORT are both specified, HiRDB_PDNAMEPORT takes precedence.

When PDNAMEPORT is specified in an environment variable group, the PDNAMEPORT specification of the environment variable group becomes effective.

#3: This information must be specified for accessing one HiRDB server using an X/Open-compatible API from multiple OLTPs.

#4: When HiRDB_PDTMID is specified, it is not necessary to specify PDTMID because the value specified in HiRDB_PDTMID is assumed for PDTMID. However, if HiRDB_PDTMID is not specified, PDTMID must be specified. If PDTMID and HiRDB_PDTMID are both specified, HiRDB_PDTMID takes precedence.

#5: When HiRDB_PDXAMODE is specified, it is not necessary to specify PDXAMODE because the value specified in HiRDB_PDXAMODE is assumed for PDXAMODE. However, if HiRDB_PDXAMODE is not specified, PDXAMODE must be specified. If PDXAMODE and HiRDB_PDXAMODE are both specified, HiRDB_PDXAMODE takes precedence.

#6: The same information must be specified in the various definitions.

#7: The same specification must be made at the servers of all users who access HiRDB. For this reason, specify this information in the user service default definition, rather than in the separate user service definitions.

#8: So that the user servers can be distinguished, this information should be specified in the individual user service definitions, rather than in the separate user-service default definition.

#9: When the multi-connection facility is used, the environment variable specification variables become invalid, even if these environment variables are set to the environment variable group that was registered for each connection destination. Also, in the Windows environment, variables become invalid even though they are specified in the HiRDB.ini file. The information that was specified in the OpenTP1 system service definitions becomes valid for these environment variables.

#10: Note that if the `trnstring` option and the `PDXAMODE` setting do not match, the `xa` function results in a `-6` error.

(2) Using a UAP under TP1/LiNK as the client

If a UAP under TP1/LiNK is used as the client, the client environment definitions must be specified in the TP1/LiNK definitions. The specification procedures are as follows.

- **Specifying environment variables for recovery control if a transaction failure occurs:**

  In the **Resource Manager** window, click the **Options** button. When the **Options** dialog box is displayed, specify the environment variables in the **Transaction Service Environment Variables** field.

- **Specifying environment variables that are common to all UAPs:**

  Open the **SPP** (or **SUP**) **Environment Assignment** dialog box and specify the environment variables in the **Global** field of the **User Server Environment Variables** field.

- **Specifying environment variables individually for each UAP:**

  Open the **SPP** (or **SUP**) **Environment Assignment** dialog box and specify the environment variables in the **Local** field of the **User Server Environment Variables** field.

The following table shows the TP1/LiNK definitions in which the environment variables are specified.

Table 6-22: TP1/LiNK definitions in which the environment variables are specified

Environment variable	Transaction Service Environment Variables field	User Server Environment Variables field	
		Global field	Local field
`HiRDB_PDHOST`	N	N	N
`HiRDB_PDNAMEPORT`	N	N	N
`HiRDB_PDTMID`	N	N	N
`HiRDB_PDXAMODE`	N	N	N

Environment variable	Transaction Service Environment Variables field	User Server Environment Variables field	
		Global field	Local field
PDHOST	M#2	M#2	O#2, #3
PDNAMEPORT	M#2	M#2	O#2, #3
PDTMID#5	O#1, #2	O#1, #2	O#1, #2, #3
PDXAMODE#5	O#2	O#2	O#2, #3
PDTXACANUM#5	O	O	O
PDCLTPATH	O	O	O
PDUSER	N	N	M
PDCWAITTIME	O	O	O
PDSWAITTIME	M	M	M
PDSQLTRACE	O	O	O
PDUAPERLOG	O	O	O
PDCLTAPNAME	O	O#4	O#4
PDSWATCHTIME	M	M	M
PDTRCMODE	O	O	O
PDUAPREPLVL	O	O	O
PDREPPATH	O	O	O
PDTRCPATH	O	O	O
PDSQLTRCOPENMODE	O	O	O
PDAUTOCONNECT	N	N	N
PDXARCVWTIME	O	N	N
PDCWAITTIMEWRNPNT	O	O	O
PDTCPCONOPT	O	O	O
Other environment variable	N	O	O

M: Required.

O: Optional; specify as needed.

N: Not required.

Note

For details about the TP1/LiNK definitions, see the *TP1/LiNK User's Guide*.

#1: This information must be specified for accessing one HiRDB server using an X/Open-compatible API from multiple OLTPs.

#2: The same information must be specified in the various definitions.

#3: The same specification must be made at the servers of all users who access HiRDB. For this reason, specify this information in the **Global** field of the **User Server Environment Variables** field, rather than in the **Local** field.

#4: So that the user servers can be distinguished, this information should be specified in the **Local** field of the **User Server Environment Variables** field, rather than in the **Global** field.

#5: When the multi-connection facility is used, the environment variable specification variables become invalid, even if these environment variables are set to the environment variable group that was registered for each connection destination. Also, in the Windows environment, variables become invalid even though they are specified in the HiRDB.ini file. The information that was specified in the TP1/LiNK definitions becomes valid for these environment variables.

(3) Using a UAP under TPBroker for C++ as the client

If a UAP under TPBroker for C++ is used as the client, the client environment definitions must be specified in the TPBroker for C++ system definitions. For details about the TPBroker for C++ system definitions, see the *TPBroker User's Guide*.

The client environment definitions are specified with the following format.

- **Specifying client environment definitions in a transaction completion process:**

  Specify the client environment definitions in the transaction definition. In this case, use the `tsdefvalue` command of TPBroker for C++ to specify the definitions. The definition key is `/OTS`, and the definition parameter is `completion_process_env`.

  ```
  tsdefvalue /OTS completion_process_env
       -a 'environment-variable-name=specification-value',
  ['environment-variable-name=specification-value', ...]
  ```

- **Specifying client environment definitions in a transaction recovery process in the event of a transaction failure:**

  Specify the client environment definitions in the transaction definition. In this case, use the `tsdefvalue` command of TPBroker for C++ to specify the definitions. The definition key is `/OTS`, and the definition parameter is `recovery_process_env`.

```
tsdefvalue /OTS recovery_process_env
     -a 'environment-variable-name=specification-value',
['environment-variable-name=specification-value', ...]
```

- **Specifying client environment definitions individually for each UAP:**

  Specify the client environment definitions in the operating environment of each UAP. Specify the definitions according to the environment variable setting method (for example, the `SET` or `SETENV` format) of the operating environment.

- **Specifying client environment definitions individually for each UAP to be monitored:**

  Specify the client environment definitions in the definition file of each process monitoring definition of TPBroker for C++.

The following table shows the TPBroker for C++ definitions in which the environment variables are specified.

Table 6-23: TPBroker for C++ definitions in which the environment variables are specified

Environment variable	Transaction completion process	Transaction recovery process	Each UAP
`HiRDB_PDHOST`[#8]	O[#1, #4]	O[#1, #4]	O[#1, #4]
`HiRDB_PDNAMEPORT`[#8]	O[#1, #5]	O[#1, #5]	O[#1, #5]
`HiRDB_PDTMID`[#8]	O[#1, #3, #6]	O[#1, #3, #6]	O[#1, #3, #6]
`HiRDB_PDXAMODE`[#8]	O[#1, #7]	O[#1, #7]	O[#1, #7]
`PDHOST`	O[#1, #4]	O[#1, #4]	O[#1, #4]
`PDNAMEPORT`	O[#1, #5]	O[#1, #5]	O[#1, #5]
`PDTMID`[#8]	O[#1, #3, #6]	O[#1, #3, #6]	O[#1, #3, #6]
`PDXAMODE`[#8]	O[#1, #7]	O[#1, #7]	O[#1, #7]
`PDTXACANUM`[#8]	O	O	O
`PDCLTPATH`	O	O	O
`PDUSER`	M	N	M

Environment variable	Transaction completion process	Transaction recovery process	Each UAP
PDCWAITTIME	O	O	O
PDSWAITTIME	M	M	M
PDSQLTRACE	O	O	O
PDUAPERLOG	O	O	O
PDCLTAPNAME	O	O	O[#2]
PDSWATCHTIME	N	N	N
PDTRCMODE	O	O	O
PDUAPREPLVL	O	O	O
PDREPPATH	O	O	O
PDTRCPATH	O	O	O
PDSQLTRCOPENMODE	O	O	O
PDAUTOCONNECT	N	N	N
PDCWAITTIMEWRNPNT	O	O	O
PDTCPCONOPT	O	O	O
PDAUTORECONNECT	N	N	N
PDRCCOUNT	N	N	N
PDRCINTERVAL	N	N	N
PDKALVL	N	N	N
PDKATIME	N	N	N
PDSQLTEXTSIZE	O	O	O
PDSQLEXECTIME	O	O	O
PDRCTRACE	N	N	N
Other environment variable	O	N	O

M: Required.

O: Optional; specify as needed.

N: Not required.

#1: The same information must be specified in the client environment definitions for the transaction completion process, transaction recovery process, and each UAP.

#2: So that the processes can be distinguished, this information should be specified in the individual processes.

#3: This information must be specified for accessing one HiRDB server using an X/Open-compatible API from multiple OLTPs.

#4: When `HiRDB_PDHOST` is specified, it is not necessary to specify `PDHOST` because the value specified in `HiRDB_PDHOST` is assumed for `PDHOST`. However, if `HiRDB_PDHOST` is not specified, `PDHOST` must be specified. If `PDHOST` and `HiRDB_PDHOST` are both specified, `HiRDB_PDHOST` takes precedence.

When `PDHOST` is specified in an environment variable group, the `PDHOST` specification of the environment variable group becomes effective.

For guidelines on the value to be specified in `PDHOST`, see *(7) Fixing the communication-target server by specifying the PDFESHOST name in PDHOST (limited to HiRDB/Parallel Server)*.

#5: When `HiRDB_PDNAMEPORT` is specified, it is not necessary to specify `PDNAMEPORT` because the value specified in `HiRDB_PDNAMEPORT` is assumed for `PDNAMEPORT`. However, if `HiRDB_PDNAMEPORT` is not specified, `PDNAMEPORT` must be specified. If `PDNAMEPORT` and `HiRDB_PDNAMEPORT` are both specified, `HiRDB_PDNAMEPORT` takes precedence.

When `PDNAMEPORT` is specified in an environment variable group, the `PDNAMEPORT` specification of the environment variable group becomes effective.

#6: When `HiRDB_PDTMID` is specified, it is not necessary to specify `PDTMID` because the value specified in `HiRDB_PDTMID` is assumed for `PDTMID`. However, if `HiRDB_PDTMID` is not specified, `PDTMID` must be specified. If `PDTMID` and `HiRDB_PDTMID` are both specified, `HiRDB_PDTMID` takes precedence.

#7: When `HiRDB_PDXAMODE` is specified, it is not necessary to specify `PDXAMODE` because the value specified in `HiRDB_PDXAMODE` is assumed for `PDXAMODE`. However, if `HiRDB_PDXAMODE` is not specified, `PDXAMODE` must be specified. If `PDXAMODE` and `HiRDB_PDXAMODE` are both specified, `HiRDB_PDXAMODE` takes precedence.

#8: When the multi-connection facility is used, the environment variable specification variables become invalid, even if these environment variables are set in the environment variable group that was registered to each connection destination. Also, in the Windows environment, variables become invalid even though they are specified in the HiRDB.ini file. The information that was specified in the TPBroker for C++ system definitions becomes valid for these environment variables.

(4) Using a UAP under TUXEDO as the client

For the operation mode in which a UAP under TUXEDO is used as the client, specify the client environment definitions in the file specified by the ENVFILE parameter in the TUXEDO configuration file (UBBCONFIG file).

The following table shows the environment variables that can be specified.

Table 6-24: Environment variable specification status (for a UAP under TUXEDO)

Environment variable	Specification status
HiRDB_PDHOST	N
HiRDB_PDNAMEPORT	N
HiRDB_PDTMID	N
HiRDB_PDXAMODE	N
PDHOST	M#1
PDNAMEPORT	M#1
PDTMID#4	O#1, #3
PDXAMODE#4	M#1
PDTXACANUM	N
PDUSER	M
PDSWAITTIME	M
PDCLTAPNAME	O#2
PDSWATCHTIME	N
PDAUTORECONNECT	N
PDRCCOUNT	N
PDRCINTERVAL	N
PDKALVL	N
PDKATIME	N
PDRCTRACE	N
Other environment variable	O

M: Required.

O: Optional; specify as needed.

N: Do not specify.

#1: The same information must be specified in the environment variables for the transaction manager server, TUXEDO system server, and each UAP.

For guidelines on the value to be specified in PDHOST, see *(7) Fixing the communication-target server by specifying the PDFESHOST name in PDHOST (limited to HiRDB/Parallel Server)*.

#2: So that the processes can be distinguished, this information should be specified in the individual processes.

#3: This information must be specified for accessing one HiRDB server using an X/Open-compatible API from multiple OLTPs.

#4: In the Windows environment, variables become invalid even though they are specified in the HiRDB.ini file. The information that was specified in the file specified by the ENVFILE parameter in the TUXEDO configuration file becomes valid for these environment variables.

(5) Using a UAP under WebLogic Server as the client

For the operation mode in which a UAP under WebLogic Server is used as a client, specify the client environment definitions in the environment variables of the WebLogic Server process.

The following table shows the environment variables that can be specified.

Table 6-25: Environment variable specification status (for a UAP under WebLogic Server)

Environment variable	Specification status
HiRDB_PDHOST	N
HiRDB_PDNAMEPORT	N
HiRDB_PDTMID	N
HiRDB_PDXAMODE	N
PDHOST	M#3
PDNAMEPORT	M
PDTMID#4	O#1
PDXAMODE#4	M

Environment variable	Specification status
PDUSER	M
PDSWAITTIME	M
PDCLTAPNAME	O#2
PDSWATCHTIME	N
PDAUTORECONNECT	N
PDRCCOUNT	N
PDRCINTERVAL	N
PDKALVL	N
PDKATIME	N
PDRCTRACE	N
Other environment variable	O

M: Required.

O: Optional; specify as needed.

N: Do not specify.

#1: This environment variable must be specified when multiple OLTP programs use an X/Open-compliant API to access one HiRDB system.

#2: This environment variable should be specified in each process so that the individual processes can be identified.

#3: For guidelines on the value to be specified in PDHOST, see *(7) Fixing the communication-target server by specifying the PDFESHOST name in PDHOST (limited to HiRDB/Parallel Server)*.

#4: In the Windows environment, variables become invalid even though they are specified in the HiRDB.ini file. The information that was specified in the file specified by the ENVFILE parameter in the TUXEDO configuration file becomes valid for these environment variables.

Notes

1. For the timeout second count that can be specified in the transaction attributes of WebLogic Server, specify a value that is larger than the maximum wait time specified by PDCWAITTIME. If you specify a value that is smaller than the maximum wait time specified by PDCWAITTIME, the system may not be able to complete UAP transactions.

2. If the maximum number of concurrent transactions per process specified by PDTXACANUM is less than the number of connections specified by the JDBC connection pool of WebLogic Server, the number of connections established by the JDBC connection pool cannot exceed the PDTXACANUM value.

(6) Using a UAP under TP1/EE as the client (limited to the UNIX edition)

For the operation mode in which a UAP under TP1/EE is used as a client, specify the client environment definitions in the OpenTP1 system service definitions of the TP1/EE execution environment. For details, see *(1) Using a UAP under OpenTP1 as the client*.

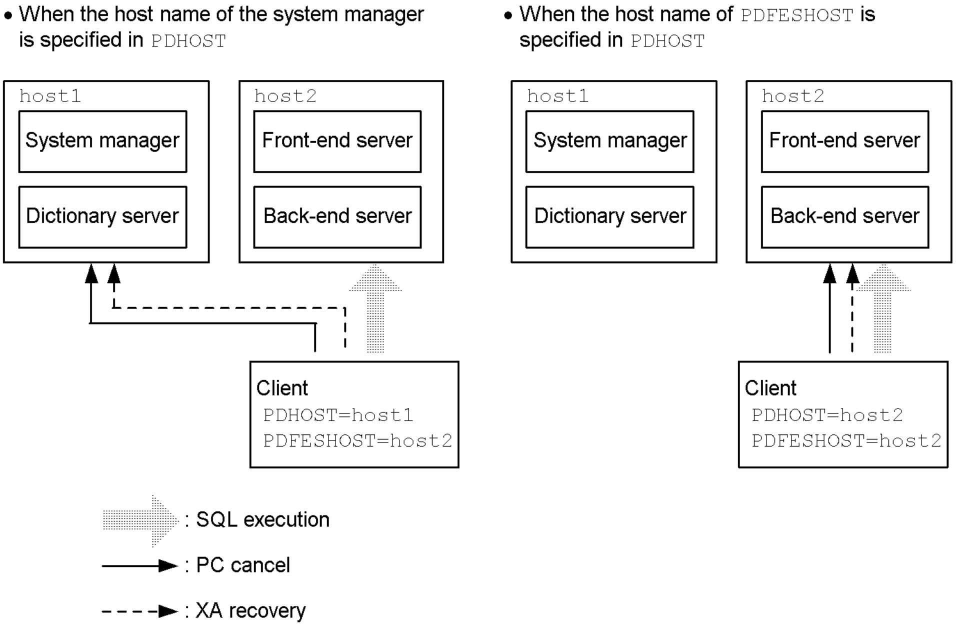
Be sure to specify PDXAMODE. If the value specified for the OpenTP1 system in which TP1/EE is executed and the value specified for PDXAMODE are different, specify PDXAMODE in the user service definitions of the OpenTP1 system.

(7) Fixing the communication-target server by specifying the PDFESHOST name in PDHOST (limited to HiRDB/Parallel Server)

When the PDFESHOST host name is specified in PDHOST, the HiRDB client can be connected to the HiRDB server if a failure occurs in the system manager unit. In addition, the SQL execution destination, the PC cancel destination, and the XA recovery request destination can be fixed to a single communication-target server. PC cancel refers to the server completion instruction when the PDCWAITTIME duration is exceeded. XA recovery refers to the transaction completion instruction when a UAP under OLTP is used. For certain combinations of the client and server versions, the host name that can be specified in PDHOST is limited to the host name of the system manager.

The following figure shows the differences between fixing and not fixing the communication-target server.

Figure 6-1: Differences between fixing and not fixing the communication-target server



Application standard

The following conditions must be satisfied if the communication-target server is to be fixed:

- HiRDB/Parallel Server is being used.
- The connection is a FES host direct connection or high-speed connection.
- The table below shows that specifying the PDFESHOST name in PDHOST is recommended for the UAP execution environment.

UAP execution environment		Specification recommended
Non-OLTP system	A UAP has one connection, or the same PDFESHOST is specified for each connection when a UAP has multiple connections.	Y
	A different PDFESHOST is specified for each connection when a UAP has multiple connections.	N

UAP execution environment					Specification recommended
OLTP system	OLTP system (WebLogic Server) operating with a single process (multi-thread)	The same PDFESHOST is specified for all connection destinations in the OLTP system (the connection destinations are the same).[#1]			Y
		Each thread operating in the OLTP system specifies a different PDFESHOST.[#1]			N
	OLTP system (OpenTP1, TUXEDO, TPBroker for C++, or TP1/ LiNK) operating with multiple processes	All UAPs specify the same PDFESHOST[#1]			Y
		Each UAP specifies a separate PDFESHOST[#1]	Client environment definition of a UAP for which PDFESHOST is specified[#2]	The same PDFESHOST is specified for all connection destinations of a UAP for which PDFESHOST is specified	Y
				A different PDFESHOST is specified for each connection destination of a UAP for which PDFESHOST is specified.	N
			Client environment definition for the transaction manager[#3]		N

Legend:

Y: Specifying the PDFESHOST host name in PDHOST is recommended.

N: Specify the host name of the transaction manager in PDHOST.

#1: Specify the host name in the following location:

- For OpenTP1

  Environment variables specified in the user service, user default, or system environment definitions

- For TUXEDO

  Client environment definitions of the transaction manager server, the TUXEDO system server, and each UAP

- For TPBroker for C++

Transaction definitions (for completed processes and recovery processes) and client environment definitions for each UAP

- For TP1/LiNK

  Global and **Local** fields of the **User server environment variables** field

- When the multi-connection facility is used

  Environment variable settings file

#2: Specify the host name in the following location:

- For OpenTP1

  Environment variables specified in the user service or user default definitions

- For TUXEDO

  Client environment definitions of each UAP

- For TPBroker for C++

  Transaction definitions (for completed processes) and client environment definitions of the UAP

- For TP1/LiNK

  Global and **Local** fields of the **User server environment variables** field

- When the multi-connection facility is used

  Environment variable settings file

#3: Specify the host name in the following location:

- For OpenTP1

  Environment variables specified in the transaction service definitions

- For TUXEDO

  Client environment definitions of the transaction manager server and the TUXEDO system server

- For TPBroker for C++

  Transaction definition (recovery process)

- For TP1/LiNK

  Transaction service environment variables field

- When the multi-connection facility is used

Environment variable settings file

Note

If a host number is specified in PDFESHOST, connect the port number of the connection destination to PDNAMEPORT.

6.6.3 Client environment definitions

The table below lists the client environment definitions. The numbers in the list correspond to the individual environment definition numbers used in *6.6.4 Environment definition information*.

- **Environment variables that must be specified**

The environment variables displayed in bold characters must be specified regardless of which HiRDB system environment is used. Specify all other environment variables according to the HiRDB system environment being used.

Table 6-26: Client environment definitions

No.	Environment variable	Function	Category
1	PDHOST	Specifies the host name of the HiRDB system to be connected.	System configuration[#3]
2	PDNAMEPORT	Specifies the port number of the HiRDB system.	
3	PDFESHOST	Specifies the host name of the front-end server.	
4	PDSERVICEGRP	Specifies the server name of the single server or front-end server.	
5	PDSRVTYPE	Specifies the HiRDB server type.	
6	PDSERVICEPORT	Specifies the port number for high-speed connection.	
7	PDFESGRP	Specifies the FES group to which connection is to be established when a high-speed connection is used.	
8	PDCLTRCVPORT	Specifies the receive port number of the client.	
9	PDCLTRCVADDR	Specifies the IP address or host name of the client.	
10	PDTMID	Specifies a unique identifier for each OLTP when multiple OLTPs access one HiRDB server.	Clients that use an X/Open-compliant API in an OLTP environment[#1]

No.	Environment variable	Function	Category
11	PDXAMODE	Specifies whether the transaction transfer function is to be used when the HiRDB client is linked with an OLTP system.	
12	PDTXACANUM	Specifies the maximum number of transactions to be executed simultaneously from a UAP that uses an X/Open-compliant API.	
13	PDXARCVWTIME	Specifies the wait time if a transaction cannot be recovered.	
14	PDXATRCFILEMODE	Specifies the format of each trace file name in the connection mode that uses the X/Open-compliant API.	
15	PDXAAUTORECONNECT	Specifies whether connection is to be re-established automatically when TP1/EE is linked.	
16	HiRDB_PDHOST	Specifies the host name of the HiRDB server to be connected.	
17	HiRDB_PDNAMEPORT	Specifies the port number of the HiRDB server.	
18	HiRDB_PDTMID	Specifies a unique identifier for each OLTP when multiple OLTPs access one HiRDB server.	
19	HiRDB_PDXAMODE	Specifies whether the transaction transfer function is to be used when the HiRDB client is linked with an OLTP system.	
20	PDUSER[#4]	Specifies the authorization identifier and password. This environment variable can be omitted in the UNIX environment.	User execution environment
21	PDCLTAPNAME	Specifies UAP identification information (UAP identifier) for accessing the HiRDB server.	
22	PDCLTLANG	Specifies the character code classification used in the descriptions of UAPs to be processed by the preprocessor.	
23	PDLANG	Specifies that the character code used when the UAP is executed is either Unicode or EUC Chinese kanji code. In the Linux edition, specifies that SJIS is used as the character code.	

No.	Environment variable	Function	Category
24	PDDBLOG	Specifies whether the database update log is to be collected when a UAP is executed.	
25	PDEXWARN	Specifies whether return codes with warnings are to be accepted from the server.	
26	PDSUBSTRLEN	Specifies the maximum number of bytes used to represent one character.	
27	PDCLTCNVMODE	Specifies whether character codes are to be converted if the HiRDB server and the HiRDB client use different character code classifications.	
28	PDCLTGAIJIDLL	Specifies the name of the user-defined external character conversion DLL file.	
29	PDCLTGAIJIFUNC	Specifies the name of the user-defined external character conversion function.	
30	PDCLTGRP	Specifies a client group name when the connection frame guarantee facility for client groups is used.	
31	PDTCPCONOPT	Specifies that the number of TCP ports used in server connection processing is to be reduced when the client connects to a HiRDB server with a version of 06-02 or later.	
32	PDAUTORECONNECT	Specifies whether the automatic reconnect facility is to be used.	
33	PDRCCOUNT	Specifies the number of times the CONNECT statement is to be retried by the automatic reconnect facility.	
34	PDRCINTERVAL	Specifies the retry interval for CONNECT statement execution by the automatic reconnect facility.	
35	PDUAPENVFILE	Specifies the UAP environment definition file that defines the execution environment if the UAP is to be executed in a separate environment.	
36	PDDBBUFLRU	Specifies whether the LRU method is used for caching in global buffer pages accessed by the UAP.	
37	PDHATRNQUEUING	Specifies that the client is not using the transaction queuing facility.	

No.	Environment variable	Function	Category
38	PDCLTBINDLOOPBACKADDR	Specifies whether a loopback address is to be used for bind() when the receive port used for communication with the HiRDB server is created.	
39	PDASTHOST	Specifies the host name of HiRDB Control Manager - Agent to be connected when the UAP is executed.	Command execution from a UAP
40	PDASTPORT	Specifies the port number of Control Manager - Agent to be connected when the UAP is executed.	
41	PDSYSTEMID	Specifies the HiRDB identifier of the HiRDB server managed by HiRDB Control Manager - Agent to be connected when the UAP is executed.	
42	PDASTUSER	Specifies the user name and password for the OS that will run commands.	
43	PDCMDWAITTIME	Specifies the maximum time the client is to wait from the time it sends a request to HiRDB Control Manager - Agent until a response is returned.	
44	PDCMDTRACE	Specifies the size of the command trace file when a file is output during UAP execution.	
45	PDIPC	Specifies the communication method between processes.	Inter-process memory communication facility
46	PDSENDMEMSIZE	Specifies the data storage area size when the client sends data to the server while the inter-process memory communication facility is used.	
47	PDRECVMEMSIZE	Specifies the data storage area size when the client receives data from the server while the inter-process memory communication facility is used.	

No.	Environment variable	Function	Category
48	PDCWAITTIME#4	Specifies the maximum time that the HiRDB client waits for a response to be returned after issuing a request to the HiRDB server.	System monitoring
49	PDSWAITTIME#4	Specifies the maximum time that the HiRDB server waits for the next request from the HiRDB client to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time during transaction processing.	
50	PDSWATCHTIME	Specifies the maximum time that the HiRDB server waits for the next request from the HiRDB client to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time outside transaction processing.	
51	PDCWAITTIMEWRNPNT	Specifies the output timing of the SQL runtime warning information file when the SQL runtime running output facility is used. The output timing is specified as a percentage of the maximum wait time of the HiRDB client or as a time.	
52	PDKALVL	Specifies whether the facility that sends packets regularly from the HiRDB client to the HiRDB server is to be used.	
53	PDKATIME	Specifies the interval for sending packets regularly from the HiRDB client to the HiRDB server.	
54	PDTIMEDOUTRETRY	Specifies the number of times the connect() system call is to be retried if an error occurs in the connect() system call executed when the HiRDB client connects with the HiRDB server.	
55	PDNBLOCKWAITTIME	Specifies the connection establishment monitoring time in nonblock mode when completion of the connection between the HiRDB server and client is monitored.	
56	PDCONNECTWAITTIME	Specifies the maximum wait time that the HiRDB client waits from a response from the HiRDB server during connection with the HiRDB server.	

No.	Environment variable	Function	Category
57	PDCLTPATH	Specifies the storage directory for the SQL trace file and client error log file created by the HiRDB client.	Trouble-shooting
58	PDSQLTRACE	Specifies the size of the SQL trace file (in byte units) into which the SQL trace of the UAP is to be output.	
59	PDUAPERLOG	Specifies the size of the client error log file (in bytes) to which UAP error logs are to be output.	
60	PDERRSKIPCODE	Specifies specific client error logs that are not to be output.	
61	PDPRMTRC	Specifies whether parameter information and retrieval data is to be output to the SQL trace information.	
62	PDPRMTRCSIZE	Specifies the maximum length of the parameter information and retrieval data to be output to the SQL trace information.	
63	PDTRCMODE	Specifies whether troubleshooting information other than SQL trace information is to be output.	
64	PDUAPREPLVL	Specifies output information for UAP statistical reports.	
65	PDREPPATH	Specifies the output directory for UAP statistical reports when these files are to be created in a directory different from the one specified by PDCLTPATH.	
66	PDTRCPATH	Specifies the storage directory for dynamic SQL trace files.	
67	PDSQLTRCOPENMODE	Specifies the open mode for the SQL trace file when PDREPPATH is specified.	
68	PDSQLTEXTSIZE	Specifies the size of the SQL statement to be output to the SQL trace information.	
69	PDSQLEXECTIME	Specifies whether the SQL runtime is to be output to the SQL trace information.	
70	PDRCTRACE	Specifies the size of the file that outputs the UAP reconnect trace.	

No.	Environment variable	Function	Category
71	PDWRTLNPATH	Specifies the storage directory for files to which value expression values of WRITE LINE statements are to be output.	
72	PDWRTLNFILSZ	Specifies the maximum size of the files to which value expression values of WRITE LINE statements are to be output.	
73	PDWRTLNCOMSZ	Specifies the total size of the value expression values in WRITE LINE statements.	
74	PDUAPEXERLOGUSE	Specifies whether the facility for output of extended SQL error information is to be used.	
75	PDUAPEXERLOGPRMSZ	Specifies the maximum data length for the parameter information to be output to the client error log file and to the SQL error report file when the facility for output of extended SQL error information is used.	
76	PDARYERRPOS	Specifies whether the value indicating an erroneous array element is to be set in the SQL Communications Area when update processing using arrays results in an error.	
77	PDDNDPTRACE	Specifies the file size for method traces output by the HiRDB.NET data provider supporting ADO.NET 2.0.	
78	PDVWOPTMODE	Specifies whether or not the access path information file is to be obtained.	Access path display utility
79	PDTAAPINFPATH	Specifies the output destination directory when the access path information file is output to the HiRDB client side. The file is not output without this specification.	Access path information file for HiRDB SQL Tuning Advisor
80	PDTAAPINFMODE	Specifies the file name format of the access path information file when it is output to the HiRDB client side.	
81	PDTAAPINFSIZE	Specifies the file size of the access path information file when it is output to the HiRDB client side.	
82	PDSTJTRNOUT	Specifies whether UAP statistical information is to be output to the client side.	Output unit of UAP statistical information
83	PDLOCKLIMIT	Specifies the maximum number of lock requests that a UAP can issue to one server.	Lock control

No.	Environment variable	Function	Category
84	`PDDLKPRIO`	Specifies the deadlock priority value of a UAP.	
85	`PDLOCKSKIP`	Specifies whether search using condition evaluation with no lock is to be performed.	
86	`PDFORUPDATEEXLOCK`	Specifies whether `WITH EXCLUSIVE LOCK` is to be applied to the lock option of the SQL statement in which the `FOR UPDATE` clause was specified (or assumed).	
87	`PDISLLVL`	Specifies the data guarantee level for SQL statements.	SQL-related
88	`PDSQLOPTLVL`	Specifies the optimization method (SQL optimization option) for determining the most efficient access path in consideration of the database status.	
89	`PDADDITIONALOPTLVL`	Specifies the optimization method (SQL extension optimizing option) for determining the most efficient access path in consideration of the database status.	
90	`PDHASHTBLSIZE`	Specifies the hash table size to be used when hash join or subquery hash execution is applied in SQL optimization.	
91	`PDDFLNVAL`	When table data is to be fetched into an embedded variable, specifies whether a default value is to be set into the embedded variable if the fetched value is a null value.	
92	`PDAGGR`	Specifies the maximum number of groups allowed in each server so that the memory size used in `GROUP BY` processing can be determined.	
93	`PDCMMTBFDDL`	Specifies whether a transaction that has that executed a data manipulation SQL statement must execute `commit` processing automatically before it executes a definition SQL statement.	
94	`PDPRPCRCLS`	Specifies whether an open cursor is to be closed automatically if the SQL identifier being used by the open cursor is used again by the `PREPARE` statement.	
95	`PDAUTOCONNECT`	Specifies whether autoconnection is to be executed if an SQL statement is executed while the client is not connected to HiRDB.	

No.	Environment variable	Function	Category
96	PDDDLDEAPRPEXE	Specifies that the preceding transaction's preprocessing results are to be ignored and the definition transaction is to be executed.	
97	PDDDLDEAPRP	Specifies whether the definition information of a table being used by a closed holdable cursor can be changed by another UAP between transactions.	
98	PDLCKWAITTIME	Specifies the maximum amount of time the HiRDB client is to monitor a lock request for the lock to be released, beginning when the lock request is placed on wait status.	
99	PDCURSORLVL	Specifies whether an open/close cursor request is to be sent automatically to the HiRDB server when a search is performed using the cursor.	
100	PDDELRSVWDFILE	Specifies the name of the SQL reserved word deletion file when the SQL reserved word deletion file is used.	
101	PDHJHASHINGMODE	Specifies the hashing method when Apply hash join, subquery hash execution is selected as the SQL extension optimizing option.	
102	PDCALCMDWAITTIME	Specifies the maximum amount of time the HiRDB client is to wait for termination of a command executed by the CALL COMMAND statement or a utility, beginning when the command's execution starts.	
103	PDSTANDARDSQLSTATE	Specifies whether the details of the SQLSTATE value are to be displayed.	
104	PDBLKF	Specifies the number of rows to be sent by a single transfer process when the HiRDB server transfers search results to the HiRDB client.	Block transfer facility
105	PDBINARYBLKF	Specifies whether the block transfer facility is to be applied when a table having a selection expression for BINARY-type data with a defined length of 32,001 bytes or more is searched.	
106	PDBLKBUFFSIZE	Specifies the size of the server-client communication buffer used by the block transfer facility.	

No.	Environment variable	Function	Category
107	PDBLKFUPD	Specifies whether the block transfer facility is to be suppressed when an extended cursor is used for retrieval for a query with FOR UPDATE specified.	
108	PDBLKFERRBREAK	Specifies when an error is to be returned to the UAP if an implicit rollback occurs while multiple rows are being acquired by the block transfer facility.	
109	PDNODELAYACK	Specifies whether immediate acknowledgement is to be used. This environment variable is limited to the AIX version.	HiRDB communication processing
110	PDBINDRETRYCOUNT	Specifies the retries count when EADDRINUSE is returned for a bind system call in a UNIX domain.	
111	PDBINDRETRYINTERVAL	Specifies the retry interval when EADDRINUSE is returned for a bind system call in a UNIX domain.	
112	PDCLTSIGPIPE	Specifies whether the HiRDB client's signal handler is to be set for the SIGPIPE signal.	
113	PDDBACCS	Specifies the generation number of an RDAREA if an RDAREA that is not the current RDAREA is to be accessed while the inner replica facility is being used.	Inner replica facility
114	PDDBORGUAP	Specifies that the UAP is to be executed for the original RDAREA for online reorganization hold.	Updatable online reorganization
115	PDSPACELVL	Specifies the space conversion level for data storage, comparison, and retrieval.	Space conversion for data
116	PDCLTRDNODE	Specifies the identifier of the XDM/RD E2 database to be connected when the XDM/RD E2 connection facility is used.	XDM/RD E2 connection facility
117	PDTP1SERVICE	Specifies whether OpenTP1 service names are to be reported to XDM/RD E2 when the XDM/RD E2 connection facility is used.	
118	PDRDCLTCODE	Specifies the character code classification that the client uses when the XDM/RD E2 connection facility is used.	

No.	Environment variable	Function	Category
119	PDCNSTRNTNAME	Specifies the position of the constraint name definition when a referential or check constraint is defined.	Referential or check constraint
120	PDBESCONHOLD	Specifies whether the BES connection holding facility is to be used.	BES connection holding facility
121	PDBESCONHTI	Specifies the BES connection holding period when the BES connection holding facility is used.	
122	PDODBSTATCACHE	Specifies whether column information or index information that is collected the first time an ODBC function (SQLColumns() or SQLStatistics()) is issued is to be cached.	ODBC functions
123	PDODBESCAPE	Specifies whether the & ESCAPE character is to be specified for the pattern character string in a retrieval that uses a cataloging ODBC function.	
124	PDGDATAOPT	Specifies that the SQLGetData function of ODBC is to fetch data from columns, even if the data has already been fetched from those columns.	
125	PDODBLOCATOR	Specifies whether the locator facility is to be used to partition and retrieve data when a database access tool is used to retrieve BLOB-type or BINARY-type column data. The database access tools are the ODBC driver, the OLE DB provider, and the HiRDB.NET data provider.	
126	PDODBSPLITSIZE	Specifies the partition acquisition size when PDODBLOCATOR=YES is specified.	
127	PDODBCWRNSKIP	Specifies whether warnings are to be skipped when an ODBC connection is used.	
128	PDJETCOMPATIBLE	Specifies whether the ODBC 3.5 driver is to be operated in a mode that is compatible with Microsoft Access, rather than operating on the basis of the ODBC 3.5 specification.	
129	PDPLGIXMK	Specifies whether delayed batch creation of plug-in indexes is to be used.	Plug-ins
130	PDPLUGINNSUB[#2]	For details, see the manual for the target plug-in.	

No.	Environment variable	Function	Category
131	PDPLGPFSZ	Specifies the initial size of the index information file for delayed batch creation of plug-ins.	
132	PDPLGPFSZEXP	Specifies the extension size of the index information file for delayed batch creation of plug-ins.	
133	PDJDBFILEDIR	Specifies the file output destination of the Exception trace log using the Type4 JDBC driver.	JDBC driver
134	PDJDBFILEOUTNUM	Specifies the number of outputs to the Exception trace log file using the Type4 JDBC driver.	
135	PDJDBONMEMNUM	Specifies the number of acquired information items in the Exception trace log memory using the Type4 JDBC driver.	
136	PDJDBTRACELEVEL	Specifies the trace acquisition level of the Exception trace log using the Type4 JDBC driver.	
137	PDXDSHOST	Specifies the host name of the XDS to be connected.	XDS client only
138	PDXDSPORT	Specifies the port number of the XDS to be connected.	
139	PDXVWOPT	Specifies whether the access path of the SQL statement used to access a table to be expanded to the memory database is to be output.	

#1: Environment variables in this category are specified only for clients that use an X/Open-compliant API in an OLTP environment to access the HiRDB server. These environment variables are invalid for any other clients, even if the variables are specified.

For details about whether each environment variable is necessary, see *6.6.2 Specifications for using a UAP under OLTP as the client*.

When the multi-connection facility is used, the values specified for the environment variables become invalid, even if they are registered to the environment variable group for each connection definition.

#2: This environment variable is set for plug-ins. The client libraries do not check the setting contents of this environment variable. Also, the information is not output to the SQL trace.

#3: For the environment variables related to system configuration, specify the necessary information when the client connects with the HiRDB server. Whether or not these environment variables can be specified depends on the connection format with the HiRDB server. For details about connection formats with the HiRDB server, see *6.6.5 Environment variables and connection types for HiRDB servers*.

#4: Can be specified in the `ENVIRONMENT` operand in the `CALL COMMAND` statement.

6.6.4 Environment definition information

(1) PDHOST=HiRDB-server-host-name[,secondary-system-HiRDB-server-host-name]

~ <identifier> ((up to 511 bytes))

This environment variable specifies the host name of the HiRDB server to be connected.

For HiRDB/Single Server, this environment variable specifies the host name of the server machine on which the single server is installed. For HiRDB/Parallel Server, this environment variable specifies the host name of the server machine on which the system manager is installed. If `PDFESHOST` is specified, the `PDFESHOST` host name can be specified. When the `PDFESHOST` host name is specified, the HiRDB client can be connected to the HiRDB server even if a failure occurs in the system manager unit.

The FQDN or the IP address can also be specified instead of the host name. The specification methods are shown below.

Host name

The host name that was specified in the `pdunit -x` operand of the system common definition must be specified.

Example:

```
PDHOST=host1
```

FQDN

The Free Qualified Domain Name (FQDN) connects the host name and domain name of the HiRDB server with a period.

Example:

```
PDHOST=host1.soft.hitachi.co.jp
```

IP address

The IP address is specified with a decimal number that has each group of 3 digits separated with a period.

Example:

```
PDHOST=172.18.131.34
```

System switchover with IP address inheritance

- For the UNIX edition

  Specify the host name of the primary system.

- For the Windows edition

  Specify the virtual network name registered for the MSCS or MSFC network name. For details about the virtual network, see the *HiRDB Version 9 System Operation Guide*.

System switchover without IP address inheritance

Specify the host names of both the primary system and the secondary system. If you specify only the host name of the primary system, you must change the specification of this environment variable after system switching occurs. After system switching, change the host name to that of the new primary system.

When an X/Open-compliant UAP under OLTP is the client and HiRDB_PDHOST is specified in the system environment definition

The `HiRDB_PDHOST` specification takes precedence. The `PDHOST` setting is replaced with the value specified in `HiRDB_PDHOST`.

Rules for FQDN specification

Do not specify an FQDN if the version of the HiRDB server to be connected is earlier than 05-03. If an FQDN is specified and there is a server process remaining after the maximum wait time (`PDCWAITTIME`) of the client elapses, the server process cannot be cancelled by sending a cancel process to the HiRDB server.

When a loopback address is specified in the pdunit -x operand in the system common definition

Specify the same loopback address in this environment variable in IP address format.

(2) PDNAMEPORT=HiRDB-server-port-number

~ <unsigned integer> ((5001-65535)) <<20000>>

This environment variable specifies the port number of the HiRDB server to be connected. Specify the HiRDB server port number to be accessed in the server machine of the host specified in `PDHOST`.

If there are multiple HiRDBs, the port number differs for each HiRDB server. However, specify the port number of the HiRDB server to be accessed.

For details about the `pd_name_port` operand, see the manual *HiRDB Version 9 System Definition*.

When an X/Open-compliant UAP under OLTP is the client and

HiRDB_PDNAMEPORT is specified in the system environment definition

The specification of HiRDB_PDNAMEPORT takes precedence. The setting of PDNAMEPORT is replaced with the value specified in HiRDB_PDNAMEPORT.

(3) PDFESHOST=front-end-server-host-name[:port-number-of-unit-containing-front-end-server][,secondary-system-front-end-server-host-name][:port-number-of-unit-containing-secondary-system-front-end-server]

~ <identifier> ((up to 523 bytes))

This environment variable is related to the HiRDB/Parallel Server.

If there are multiple front-end servers, this environment variable specifies the host name of the front-end server for the HiRDB server to be connected. If the client is to be connected to the host with a port number that is specified with -p in the pdunit system definition (when the system switchover facility is used), that port number must be specified.

The front-end servers determined by the system manager include recovery-unnecessary front-end servers.

The FQDN or the IP address can also be specified instead of the host name.

Host name

The host name that was specified in the pdunit -x operand of the system common definition must be specified.

Example:

```
PDFESHOST=host1
```

FQDN

The Free Qualified Domain Name (FQDN) connects the host name and domain name of the HiRDB server with a period.

Example:

```
PDFESHOST=host1.soft.hitachi.co.jp
```

IP address

The IP address is specified with a decimal number that has each group of 3 digits separated with a period.

Example:

```
PDFESHOST=172.18.131.34
```

System switchover with IP address inheritance

- For the UNIX edition

  Specify the host name of the primary system.

- For the Windows edition

  Specify the virtual network name registered for the MSCS or MSFC network name. For details about the virtual network, see the *HiRDB Version 9 System Operation Guide*.

System switchover without IP address inheritance

Specify the host names of both the primary system and the secondary system. If you specify only the host name of the primary system, you must change the specification of this environment variable after system switching occurs. After system switching, change the host name to that of the new primary system.

Rules for FQDN specification

Do not specify an FQDN if the version of the HiRDB server to be connected is earlier than 05-03. If an FQDN is specified, cancellation of the HiRDB server process might fail and a server process might remain after the client's maximum wait time (`PDCWAITTIME`) has elapsed.

Rules for port number omission

If the port number is omitted, the port number that was specified with `PDNAMEPORT` is used as the default value. The port number that was specified with `PDNAMEPORT` is also used as the default value if the port number of the host containing the secondary system front-end server is omitted.

Relationship with other environment variables

1. This environment variable must be specified for multiple front-end servers when the front-end server to be connected is selected by the client user or when `PDSERVICEPORT` is specified.
2. This environment variable should be specified with `PDSERVICEGRP`.

Notes

1. If a program that uses the X/Open XA interface connects to a recovery-unnecessary front-end server, that program cannot reference or update the database. In this case, specify `PDFESHOST` and `PDSERVICEGRP`, and be sure to connect to a front-end server that is not a recovery-unnecessary front-end server.
2. If there are multiple front-end servers, specify equivalent host names in `PDFESHOST` so that the load is not concentrated on the connected front-end server.
3. The host name specified in `PDFESHOST` can also be specified in `PDHOST`. This allows the HiRDB client to be connected to the HiRDB server even if an error occurs in the system manager unit.
4. If reflection processing is performed using the two-phase commit method

(fxa_sqle is specified for the reflection system definition commitment_method operand) as the synchronization point processing method in the reflected side Datareplicator, reflection processing fails if a recovery-unnecessary front-end server in the reflected-side HiRDB is connected. In this case, specify PDFESHOST and PDSERVICEGRP, and be sure to connect to a front-end server that is not a recovery-unnecessary front-end server.

(4) PDSERVICEGRP=server-name

~ <character string> ((up to 30 bytes))

This environment variable specifies the single-server name or front-end server name of the HiRDB server to be connected.

If multiple front-end servers are used with a HiRDB/Parallel Server, this environment variable specifies the server name of the front-end server to be connected.

Relationship with other environment variables

1. The time required to connect to the HiRDB server can be shortened by specifying this environment variable simultaneously with PDSERVICEPORT.
2. When using a HiRDB/Parallel Server, also specify PDFESHOST.

Note

1. If a program that uses the X/Open XA interface connects to a recovery-unnecessary front-end server, that program cannot reference or update the database. In this case, specify PDSERVICEGRP and PDFESHOST, and be sure to connect to a front-end server that is not a recovery-unnecessary front-end server.
2. If reflection processing is performed using the two-phase commit method (fxa_sqle is specified for the reflection system definition commitment_method operand) as the synchronization point processing method in the reflected side Datareplicator, reflection processing fails if a recovery-unnecessary front-end server in the reflected-side HiRDB is connected. In this case, specify PDFESHOST and PDSERVICEGRP, and be sure to connect to a front-end server that is not a recovery-unnecessary front-end server.

(5) PDSRVTYPE={WS|PC}

This environment variable specifies the server type of the HiRDB server to be connected.

WS

Specify this server type if the HiRDB server is the HP-UX, Solaris, or AIX version.

`PC`

Specify this server type if the HiRDB server is the Linux or Windows edition.

(6) PDSERVICEPORT=high-speed-connection-port-number[,secondary-system-high-speed-connection-port-number]

~ <unsigned integer> ((5001-65535))

This environment variable specifies the port number for high-speed connection when the high-speed connection facility is used. This is the port number for the scheduler process specified in the system definition. The port number for the scheduler process is specified in the following operands in the system definition:

- `pd_service_port` operand
- `pd_scd_port` operand
- `-s` option in the `pdunit` operand

For details about each operand, see the manual *HiRDB Version 9 System Definition*.

You specify this operand when a firewall or NAT is installed at the HiRDB server. For details about the settings applicable when a firewall or NAT is installed, see the *HiRDB Version 9 Installation and Design Guide*.

For multiple front-end servers, specify the high-speed connection port number of the front-end server to be connected.

Note that you must determine the connection target in such a manner that the server load is distributed, because the front-end server to be connected is fixed.

Benefits

Specifying this operand can shorten the amount of time required to connect to the HiRDB server.

Relationship with other environment variables

When this operand is specified, the following operands must also be specified:

If the HiRDB server runs Windows or Linux, specify `PC` in the `PDSRVTYPE` environment variable.

HiRDB/Single Server

- `PDHOST`
- `PDNAMEPORT`
- `PDSERVICEGRP`

HiRDB/Parallel Server

- `PDHOST`

- PDNAMEPORT
- PDFESHOST
- PDSERVICEGRP

Note

1. If you use the system switchover facility for mutual system switchover and specify different port numbers for each system in the pd_service_port operand of the system definitions, also specify a high-speed connection port number for the secondary system.
2. If a communication failure occurs in an environment where the HiRDB server is connected via a wide area LAN, the system manager process might be affected adversely and might not be able to accept many concurrent HiRDB connection requests. Therefore, if your system connects via a wide area LAN, we recommend that you specify this environment variable.

(7) PDFESGRP=FES-group[,switchover-FES-group[,switchover-FES-group]...]

~ <character string> ((up to 1024 bytes))

Specify this environment variable if you use a high-speed connection with a HiRDB/Parallel Server. When you specify this environment variable, you must also specify the following operands in the system definition:

- pd_service_port operand
- pd_scd_port operand
- -s option in the pdunit operand

To set up a high-speed connection, specify the FES group to be connected. In a configuration with multiple front-end servers, specify the FES group of the connection destination, and the switchover FES group for switching the connection if an error occurs in the first FES group.

The information to be specified for *FES-group* and *switchover-FES-group* is described below.

FES-group

A FES group collectively specifies all information for a high-speed connection destination (PDFESHOST, PDSERVICEGRP, and PDSERVICEPORT). Delimit the connection destinations with a colon (:). A specification example is shown below.

```
host1:fes1:20001
```

You can specify in PDFESHOST two host names (for the primary and secondary systems), but only one host name can be specified for a single FES group. Similarly, you can specify two port numbers in PDSERVICEPORT, but only one port number can be specified for a single FES group.

switchover-FES-group

In a configuration with multiple front-end servers, a switchover FES group is a FES group to which the connection can be switched over if a failure occurs in the front-end server of the connected FES group. If an error occurs when a switchover FES group is specified, the connection switches to the switchover FES group. If multiple switchover FES groups are specified, the connection is switched over according to the sequence in which the groups are specified.

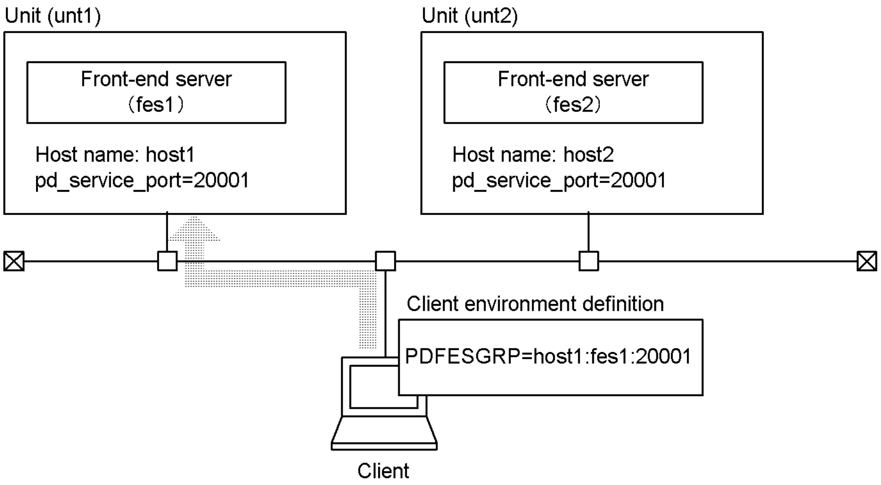
A switchover FES group is specified in the same way as a FES group.

Notes

1. If this environment variable is specified, the PDFESHOST, PDSERVICEGRP, and PDSERVICEPORT specifications are ignored.
2. When a switchover FES group is specified, the number of connections for the switchover FES group may increase temporarily because the connection destination is switched if a server error occurs or the number of connected users is exceeded. You must therefore check and, if necessary, revise the value of the pd_max_users operand for the switchover FES group.
3. When switchover FES groups are specified, it may take a while for HiRDB to return an error to the UAP if errors occur in all specified switchover FES groups or if the number of connected users is exceeded.
4. If a switchover FES group is specified and all connections to multiple FESs result in errors, only the error information for the last FES whose connection was attempted is returned to the UAP.

Usage example

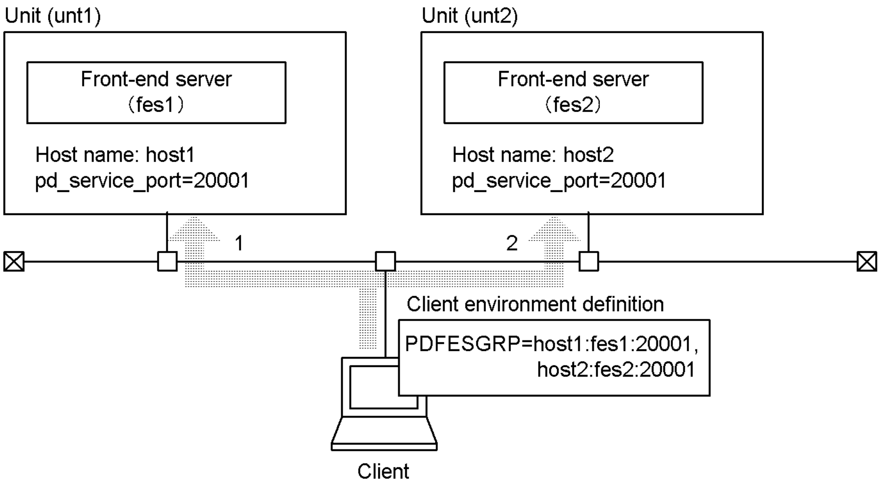
When only one FES group is specified



Explanation

If only one FES group is specified, the client is connected only to front-end server `fes1` of host `host1`.

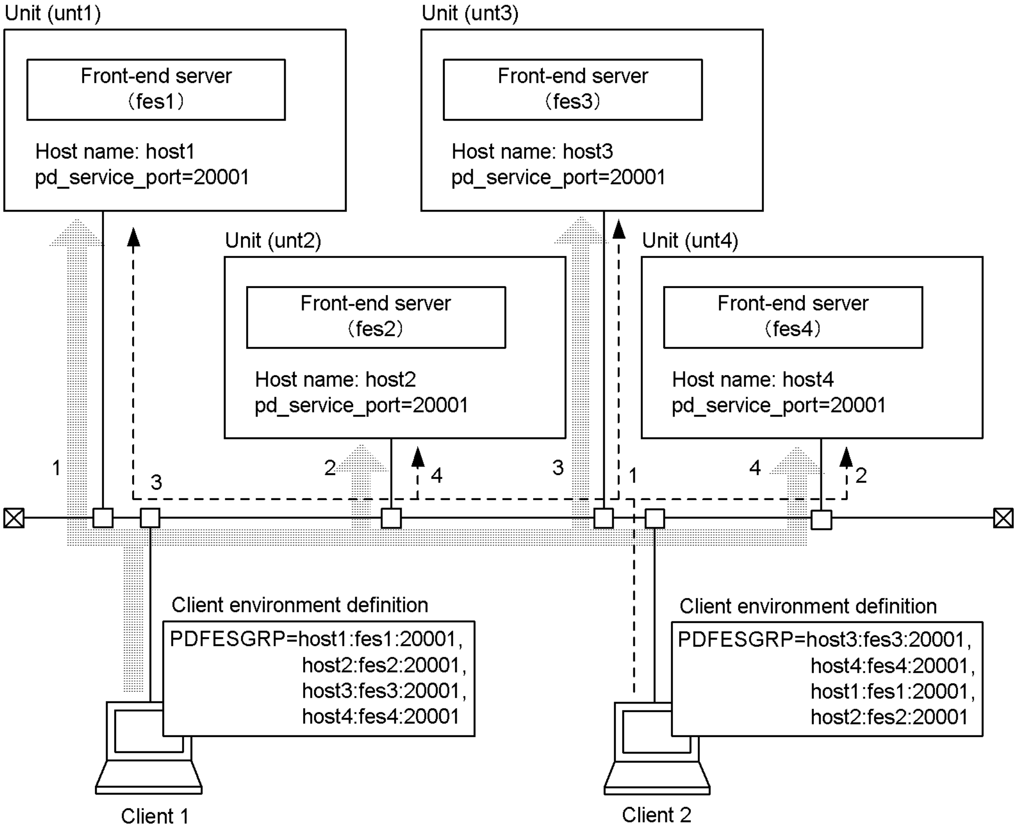
When one FES group and one switchover FES group are specified



Explanation

If an error occurs in the connection of 1, the client is connected with 2. If an error occurs in 2 as well, HiRDB returns an error to the UAP.

When one FES group and multiple switchover FES groups are specified



Explanation

If an error occurs in the connection of 1, Client 1 is connected with 2. If additional connection errors occur, Client 1 is connected with 3 and then 4. If errors occur in all connections, HiRDB returns an error to the UAP.

(8) PDCLTRCVPORT=client-receive-port-number

~ <unsigned integer> ((0, 5001-65535, 5001-65535, 5001-65535)) <<0>>

This environment variable specifies the receive port number or range of receive port numbers to be used when the HiRDB client communicates with the HiRDB server.

If this environment variable is omitted, the operating system automatically specifies the receive port number or range of receive port numbers. Therefore, this environment variable normally does not have to be specified.

Specification method

Specification examples of the receive port number are shown as follows.

- Specifying one port number:

  10000-10000 *or* 10000

- Specifying a range of port numbers:

  10000-10500

If `0` is specified, HiRDB assumes that this environment variable has not been specified.

Benefits

If a firewall has been set between the HiRDB server and the HiRDB client, and the receive port numbers that can pass through the firewall are limited, specifying this environment variable allows communications to pass through the firewall.

Notes

1. If a range of receive port numbers is specified in this environment variable, the HiRDB client automatically assigns an unused port number from the specified range. If there is no unused port number in the specified range, an error occurs.
2. The HiRDB client uses one port number for one connection to the HiRDB server. Consequently, one UAP uses multiple port numbers in the following cases:
   - The ODBC uses multiple connections.
   - The multi-connection facility is used.
3. When multiple UAPs are executed concurrently, one port number can be used only by one UAP. Therefore, if a range that includes the same port numbers is specified for multiple UAP to be executed concurrently, contention may occur when the port numbers are assigned. To ensure that the port numbers do not run out in this case, specify a range that includes more port numbers than the largest number of port numbers to be used.
4. Specify receive port numbers that are not in the range of port numbers that the operating system assigns automatically. The range of port numbers that the operating system assigns automatically differs for each operating system.
5. If the ODBC is used via a product such as Microsoft Access, multiple connections with the HiRDB server are established implicitly.
6. When specifying a range that includes 10 or more port numbers, make sure that the range includes about 20% more port numbers than will actually be used. If this margin is not included, the efficiency of the port number search process drops.
7. Any port number that is being used by other programs cannot be used by the

HiRDB client.

8. Port numbers being used by the HiRDB client cannot be used by other programs. If a service uses a fixed port number found in the range specified in this environment variable, there may be times when that service cannot be started.
9. Manage the programs found inside the firewall so that programs other than the HiRDB client do not improperly use port numbers that have been set, so that the HiRDB client can communicate through the firewall.

(9) PDCLTRCVADDR={client-IP-address|client-host-name}

~ <unsigned integer> or <identifier> ((up to 256 bytes))

When multiple communication paths are set to the host of an HiRDB client and you wish to identify a communication path for communicating with the HiRDB server, this environment variable specifies the IP address, FQDN, or host name for that communication path. The specification methods are shown below.

IP address:

Specify a decimal address, using a period to delimit each byte.

Example:

```
PDCLTRCVADDR=172.18.131.34
```

FQDN:

The FQDN is comprised of the host name and domain name of a HiRDB server separated by periods.

Example:

```
PDCLTRCVADDR=host1.soft.hitachi.co.jp
```

Notes

1. If you omit this environment variable, the IP address corresponding to the client machine's default host name is assumed. You must register the client machine's default host name in the `hosts` file or DNS (as a maximum of 256 bytes). An error will result if the default host name is not registered in the `hosts` file or DNS. If the HiRDB client and the HiRDB server are configured on the same machine in a HiRDB/Single Server and the default host name is not registered in the `hosts` file or DNS, the HiRDB server's host name will be assumed as the IP address.
2. If an invalid IP address or host name is specified in this environment variable, the HiRDB client cannot receive a response from the HiRDB server during connect processing to the HiRDB server. Therefore, an error (`SQLCODE -732`) occurs after the five-minute timer monitoring elapses.

3. PDCLTRCVADDR is ignored in the following cases:

   - The value 1 is specified in the pd_change_clt_ipaddr operand in the system definition.
   - MEMORY is specified in the PDIPC environment variable.

4. The following table shows the client's IP address to be used depending on the combination of the values specified in PDCLTRCVADDR and PDHOST:

Value specified in PDCLTRCVADDR			PDHOST		
			Loopback address	Local IP address	Another machine's IP address
Loopback address			Value specified in PDCLTRCVADDR		
Local IP address					
Omitted	Default host registered in the hosts file		Default host specified in the hosts file		
	No default host registered in the hosts file	Type4 JDBC driver	Value specified in PDHOST	KFPZ02444-E error	KFPZ02444-E error
		Other	Value specified in PDHOST	Value specified in PDHOST	KFPZ02444-E error

(10) PDTMID=OLTP-identifier

~ <identifier> ((4 characters))

This environment variable specifies a unique 4-character identifier of the applicable OLTP when multiple OLTPs use an X/Open-compliant API to access a HiRDB server.

If one of the following conditions applies to the specification of this environment variable, the HiRDB server cannot identify the OLTP to which a transaction belongs. Therefore, if a system failure or a transaction error occurs in an OLTP, the transaction completion timing is not synchronized.

- This environment variable is omitted in an operating mode in which multiple OLTPs access the HiRDB server.
- The identifiers assigned to the OLTPs are not unique in an operating mode in which multiple OLTPs access the HiRDB server.
- Multiple OLTP identifiers are assigned to the same OLTP.

When an X/Open-compliant UAP under OLTP is the client and HiRDB _PDTMID is specified in the system environment

The HiRDB_PDTMID specification takes precedence. The PDTMID setting is replaced with the value specified in HiRDB_PDTMID.

(11) PDXAMODE={0|1}

This environment variable specifies whether the transaction transfer facility is to be used when linking with a UAP that uses an X/Open-compliant API under OLTP.

`0`: Do not use the transaction transfer facility.

`1`: Use the transaction transfer facility.

This environment variable should be specified in accordance with instructions provided by the HiRDB administrator. For details about the transaction transfer facility, see the *HiRDB Version 9 Installation and Design Guide*.

When an X/Open-compliant UAP under OLTP is the client and HiRDB_PDXAMODE is specified in the system environment definition

The `HiRDB_PDXAMODE` specification takes precedence. The `PDXAMODE` setting is replaced with the value specified in `HiRDB_PDXAMODE`.

When the client is linked with OpenTP1

The `trnstring` operand of OpenTP1 and the specification of the `DXAMODE` setting must match.

When the client is linked with TPBroker for C++

At the completion of a TPBroker for C++ transaction, a transaction completion process that is different from the one used by the UAP is used. Therefore, `1` must be specified in `PDXAMODE` when the client is linked with TPBroker. If `0` is specified, UAP transactions cannot be completed.

When the client is linked with TUXEDO

At the completion of a TUXEDO global transaction, a transaction manager server (TMS) that is different from the one used by the UAP is used. Therefore, `1` must be specified in `PDXAMODE` when the client is linked with TPBroker. If `0` is specified, UAP transactions cannot be completed.

When the client is linked with WebLogic Server

Specify `1` in `PDXAMODE`. If `PDXAMODE` is omitted or if `0` is specified, UAP transactions cannot be completed.

When the client is linked with TP1/EE (limited to the UNIX edition)

Specify `0` in `PDXAMODE`. If `PDXAMODE` is omitted or if `1` is specified, transactions sometimes cannot be completed.

(12) PDTXACANUM=maximum-number-of-concurrent-transaction-executions-per-process

~ <unsigned integer> ((1-2147483647)) <<20>>

This environment variable specifies the maximum number of transactions that can be executed simultaneously per process when a UAP that uses an X/Open-compliant API supporting multi-thread or X/Open-compliant API using multi-connection facility accesses HiRDB.

Estimation method

Estimate the value to be specified for this environment variable based on the following formula:

specification-value = (*number-of-transactions-that-can-occur-in-target-process*) x (*number-of-HiRDB-servers-that-target-process-can-access*)

When linked with TP1/EE:

Add the number of recovery and monitoring threads of TP1/EE to the above value.

Specified value = (*number-of-transactions-that-can-occur-in-target-process*

+ *number-of-recovery-and-monitoring-threads*)

x (*number-of-HiRDB-servers-that-target-process-can-access*)

(13) PDXARCVWTIME=transaction-recovery-wait-time

~ <unsigned integer> ((0-270)) <<2>> (seconds)

This environment variable specifies how long to wait before sending the next connection request to HiRDB, when OpenTP1 that accesses HiRDB using an X/Open-compliant API cannot connect to HiRDB during a transaction recovery process or a resource manager monitoring process, or when HiRDB cannot recover a transaction.

If 0 is specified, a connection request is issued for each HiRDB transaction recovery instruction.

Estimation method

Estimate the specification value from the following calculation equation:

specification-value = $a \times b \div (c - d \times e)$

a: 270

b: Total number of OpenTP1 transaction recovery processes to be connected to HiRDB

c: Total number of automatic allocation port numbers on the HiRDB single server or the server machine containing the system manager

d: Number of port numbers used during peak transaction times on the HiRDB single server or the server machine containing the system manager

e: If the system switchover facility is being used, use 2. If not, use 1.

Notes

1. If you specify a small time value in PDXARCVWTIME, and several transactions are stopped in OpenTP1, a port number shortage may occur on the HiRDB single server or the server machine containing the system manager. Therefore, if the time calculated with the estimation method is larger than the default value, you should specify the time calculated with the estimation method.
2. If startup of the HiRDB single server or system manager unit is completed immediately after the OpenTP1 transaction recovery process begins the wait time specified in PDXARCVWTIME, recovery of the transaction connected to HiRDB may take longer to complete.

(14) PDXATRCFILEMODE={LUMP|SEPARATE}

This environment variable specifies the format of trace file names when the connection mode uses an X/Open-compliant API. If the connection mode does not use an API that is X/Open-compliant, the PDXATRCFILEMODE specification is ignored.

LUMP

Output the trace files without adding an execution process ID to each trace file name.

The LUMP specification is recommended if the UAP is non-resident and is executed repeatedly and the process ID changes with each execution. By specifying LUMP, you can prevent the number of trace files from increasing each time the non-resident UAP is executed and thus causing unstable operation of the operating system and other programs.

When LUMP is specified, the output destination for trace information becomes limited, and the trace output size may need to be increased. In addition, the processing time may increase because the trace output competes with the output of other processes.

SEPARATE

Output the trace files by adding an execution process ID to each trace file name.

If the UAP is a resident process, we recommend that you specify SEPARATE.

(15) PDXAAUTORECONNECT={YES|NO}

When TP1/EE is linked, this environment variable specifies whether connection is to be re-established automatically when the HiRDB server connection status is checked at the time a transaction begins and TP1/EE is not connected to the HiRDB server. If the cause of disconnection is neither a machine failure nor a network failure, connection is re-established automatically even if this environment variable is omitted.

This environment variable is ignored, if specified, in the following cases:

- TP1/EE is not linked.
- Dynamic registration is used with the transaction manager.
- The value `0` is specified in the `PDCWAITTIME` client environment variable.

`YES`

Re-establish connection automatically when a transaction begins. This option affects transaction performance because communication with the HiRDB server is performed each time a transaction begins.

`NO`

Do not re-establish connection automatically when a transaction begins. If connection is lost due to a machine or network failure, an SQL error might be returned because connection will not be re-established.

Notes

- The maximum amount of time spent checking the HiRDB server connection status is the value specified in the `PDCWAITTIME` client environment variable.
- If connection re-establishment fails, the error that caused the failure is returned to the UAP.

(16) HiRDB_PDHOST=HiRDB-server-host-name[,secondary-system-HiRDB-server-host-name]

~ <identifier>

This environment variable specifies the host name of the HiRDB server to be connected. The value specified in this environment value is replaced with the `PDHOST` setting.

For a HiRDB/Single Server, this environment variable specifies the host name of the server machine in which the Single Server is installed. For a HiRDB/Parallel Server, this environment variable specifies the host name of the server machine in which the system manager is installed.

Other than the host name, you can specify the FQDN or the IP address. The specification methods are shown below.

Host name

The host name that was specified in the `pdunit -x` operand of the system common definition must be specified.

Example:

```
PDHOST=host1
```

FQDN:

The FQDN is comprised of the host name and domain name of a HiRDB server separated by periods.

Example:

```
PDHOST=host1.soft.hitachi.co.jp
```

IP address

The IP address is specified with a decimal number that has each byte separated with a period.

Example:

```
PDHOST=172.18.131.34
```

System switchover with IP address inheritance

- For the UNIX edition

If the IP address is to be inherited, specify the host name of the primary system.

- For the Windows edition

  If the IP address is to be inherited, specify the virtual network name registered for the MSCS or MSFC network name. For details about the virtual network, see the *HiRDB Version 9 System Operation Guide*.

System switchover without IP address inheritance

Specify the host names of both the primary system and the secondary system. If you specify only the host name of the primary system, you must change the specification of this environment variable after system switching occurs. After system switching, change the host name to that of the new primary system.

(17) HiRDB_PDNAMEPORT=HiRDB-server-port-number

~ <unsigned integer> ((5001-65535))

This environment variable specifies the port number of the HiRDB server. The port number must be the value that was specified in `pd_name_port` of the system definition for the HiRDB server to be connected. The value specified in this environment variable is replaced by the `PDNAMEPORT` setting.

For multiple HiRDBs, the port number differs for each HiRDB server, so you must specify the port number of the HiRDB server to be connected.

For details about `pd_name_port`, see the manual *HiRDB Version 9 System Definition*.

(18) HiRDB_PDTMID=OLTP-identifier

~ <integer> ((4 characters))

This environment variable specifies a unique identifier for the applicable OLTP when multiple OLTPs use an X/Open-compliant API to access a HiRDB server. The value specified in this environment variable is replaced by the `PDTMID` setting.

If one of the following conditions applies to the specification of this environment variable, the server cannot identify the OLTP to which a transaction belongs. Therefore, if a system failure or transaction error occurs in an OLTP, the transaction conclusion timing is not synchronized.

- This environment variable and the PDTMID specification are omitted in an operating mode in which multiple OLTPs access the HiRDB server.
- The identifiers assigned to the OLTPs are not unique in an operating mode in which multiple OLTPs access the HiRDB server.

(19) HiRDB_PDXAMODE={0|1}

This environment variable specifies whether or not the transaction transfer facility is used when a UAP that uses an X/Open-compliant API under OLTP is being used as the client. The value specified in this environment variable is replaced with the `PDXAMODE` setting.

`0`: Do not use the transaction transfer facility.

`1`: Use the transaction transfer facility.

This operand should be specified in accordance with instructions provided by the HiRDB administrator. For details about the transaction transfer facility, see the *HiRDB Version 9 Installation and Design Guide*.

(20) PDUSER=authorization-identifier[/password]

~ <<current user's name without password>>

This environment variable cannot be omitted in the Windows environment. It can be omitted in the UNIX environment.

This environment variable specifies the authorization identifier and password in the format *authorization-identifier*/*password*. If specification of a password is not necessary (when setup is for users who do not have passwords), the password can be omitted.

Regardless of whether the specification uses upper case or lower case characters, the password is handled as upper case. However, if lower-case characters are enclosed in quotation marks, the password is handled as lower-case characters.

Note

- When you use OpenTP1, do not register `PDUSER` as a system environment variable. If you do, abort code `psti0rf` will be output when OpenTP1 starts, and HiRDB will quit.

Notes

1. When a UAP under OpenTP1 is used as the client, specify the authorization identifier and password in the format `'`*authorization-identifier*/*password*`'`. If you wish to use lower-case alphabetic characters for the authorization identifier and password, use the format `'"`*authorization-identifier*`"/"`*password*`"'`.
2. When you omit the password and specify only the authorization identifier, entry of a password may be requested, depending on the utility. In such a case, use the format *authorization-identifier*/*password* to specify a character string for the password. If the command is executed from the UAP (`COMMAND EXECUTE` is executed), the password cannot be omitted.

(21) PDCLTAPNAME=identification-name-of-UAP-to-be-executed

~ <character string> ((30 characters)) <<`Unknown`>>

This environment variable specifies identification information about the UAP that will access the HiRDB system (that is, a UAP identifier). This name is used to identify the UAP being executed.

The name specified in this environment variable is displayed as the UAP name in the following information:

- Display result of the `pdls` command
- SQL trace file
- Connection user information file (`%PDDIR%\spool\cnctusrinf)`

Note

- If non-alphanumeric characters are specified in the UAP identification name, the `pdcancel` command may not be executed. For this reason, only alphanumeric characters should be specified for name.
- Do not use the following character strings in the UAP identification names:
  - Character string that begins with `pd`#
  - Character string that begins with `hds`
  - Character string that begins with `0`

#: If a character string that begins with `pd` is specified in the identification name of a UAP, that UAP may not be monitored by the skipped effective synchronization point dump monitoring facility.

(22) PDCLTLANG={SJIS|CHINESE|UJIS|C|UTF-8|CHINESE-GB18030}

This environment variable specifies the character code classification to be used in the UAPs to be processed by the preprocessor. UJIS cannot be specified with the Windows edition. When this environment variable is omitted in the Windows edition, SJIS is assumed.

SJIS

Shift JIS kanji codes are set as the character code classification. Use this environment variable if you specify SJIS in the Linux edition.

CHINESE

chinese_s is set as the character code classification.

UJIS

ja_JP.EUC (ja_JP.eucJP or ja) is set as the character code classification.

C

Single-byte character codes are set as the character code classification.

UTF-8

UTF-8 is set as the character code classification.

CHINESE-GB18030

CHINESE-GB18030 is set as the character code classification.

During UAP preprocessing, the character code classification is determined as shown in the following table.

PDCLTLANG	Client operating system				
	HP-UX	Solaris	AIX	Linux	Windows
SJIS	ja_JP.SJIS	ja_JP.PCK	ja_JP	Shift JIS	Shift JIS
CHINESE	chinese-s	chinese-s	chinese-s	chinese-s	EUC Chinese character code (GB2312)
UJIS	ja_JP.eucJP	ja	ja_JP	ja_JP.eucJP	Error
C	C	C	C	C	C
UTF-8	UTF-8	UTF-8	UTF-8	UTF-8	UTF-8

PDCLTLANG	Client operating system				
	HP-UX	Solaris	AIX	Linux	Windows
CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030
No setting#	ja_JP.SJIS	ja	ja_JP	ja	Shift JIS
Other	Error	Error	Error	Error	Error

#: If a character code was set in the LANG environment variable during preprocessing, that character code is assumed.

The following table shows whether connection is possible based on the character code classification combination between the server and the client.

Character code classification of client#1	Character code classification of server					
	SJIS	CHINESE	UJIS	C	UTF-8	CHINESE-GB18030
SJIS	C	--	--	--	--	--
CHINESE	--	C	--	--	--	--
UJIS	--	--	C	--	--	--
C	C#2	--	C#2	C	--	--
UTF-8	--	--	--	--	C	--
CHINESE-GB18030	--	--	--	--	--	C

C: Can be connected.

--: Cannot be connected.

#1: A Windows client can connect with any character code classification used in a server. A VOS3 system client can connect with the server when the character code classification in the server is SJIS.

#2: Connection is possible when the server is set to the default character code classification. For Solaris and Linux, the default character code classification is UJIS. For other operating systems, the default character code classification is SJIS.

(23) PDLANG={UTF-8|SJIS|CHINESE|CHINESE-GB18030|ANY}

You use this environment variable when you use a character encoding that is not supported by the OS in a UAP execution environment. When this environment variable is omitted, the value of the LANG environment variable is assumed.

SJIS is supported by the Linux edition only. For the Windows edition, you can specify only ANY.

When ANY is specified, the client can connect to a server that uses any character codes. However, the client (application) needs to be aware of the character codes used by the connected server for data operations and creation of SQL statements.

For details about the value of PDLANG, see *8.4.1(1) Notes about the character code classification*.

(24) PDDBLOG={ALL|NO}

This environment variable specifies whether a database update log is to be collected when UAPs are executed.

ALL

Execute UAPs in the log collection mode.

When ALL is specified, the error correction operation becomes simple, but a significant amount of processing time is required when a large volume of data is updated.

NO

Execute UAPs in the no-log mode.

If a UAP terminates abnormally during execution, the database updates performed by the transaction cannot be recovered. The NO option reduces processing time to the extent that no time is spent on collecting a database update log. However, backups must be made before and after UAP execution, and approval to specify NO must be obtained from the HiRDB administrator.

For details about how to execute UAPs in the no-log mode, see the *HiRDB Version 9 System Operation Guide*.

The following log information is collected regardless of the specification of this environment variable:

- Log information about updates to the master directory, data directory, and data dictionary RDAREAs
- Log information about updates to user RDAREA definition information

(25) PDEXWARN={YES|NO}

This environment variable specifies whether return codes with warnings are to be accepted from the server.

YES: Accept return codes with warnings.

NO: Do not accept return codes with warnings.

When YES is specified for this environment variable, the error decision method must be changed for UAPs (including stored procedures) that process all SQLCODEs other than 0 and +100 as errors. For details about error decision methods, see *3.6 SQL error identification and corrective measures*.

(26) PDSUBSTRLEN={3|4|5|6}

This environment variable specifies the maximum number of bytes used to represent one character. This environment variable is valid only when the character code classification is Unicode (UTF-8); it affects the length of the SUBSTR scalar function. For details about SUBSTR, see the manual *HiRDB Version 9 SQL Reference*.

Relationship with system definition

If this environment variable is omitted, the setting for the pd_substr_length operand of the system common definition is assumed.

Note

For details about when you specify this environment variable, see the pd_substr_length operand in the manual *HiRDB Version 9 SQL Reference*.

(27) PDCLTCNVMODE={AUTO|NOUSE|UJIS|UJIS2|UTF8|UTF8MS|UTF8_TXT|UTF8_EX|UTF8_EX2|UTF8MS_TXT|UCS2_UJIS|UCS2_UTF8}

This environment variable specifies how character codes are to be converted when the character code classifications of the HiRDB server and the HiRDB client are different. Character code conversion can be performed only when the HiRDB client uses Shift JIS kanji codes or UCS-2, and the HiRDB server uses EUC Japanese kanji codes or Unicodes.

The following table lists and describes the specification values:

Specification value	Description
AUTO	The HiRDB client automatically checks the character code classification used in the HiRDB server and converts the character codes if possible. Character code conversion can be applied when the HiRDB client uses shift JIS kanji codes and the HiRDB server uses EUC Japanese kanji codes or Unicodes. When AUTO is specified, NOUSE, UJIS or UTF8 is set as the specification value.
NOUSE	Character code conversion is not used. Data is transferred without execution of character code conversion.
UJIS	The HiRDB client assumes that the HiRDB server uses EUC Japanese kanji codes, and converts character codes without checking what is used in the HiRDB server. The HiRDB client must be using shift JIS kanji codes. If the HiRDB client accepts data of variable-length character string types (VARCHAR, MVARCHAR, and NVARCHAR), it uses the number of spaces equivalent to the SQLLEN value to clear the SQLDATA area indicated by the SQL descriptor area.

Specification value	Description
UJIS2	The processing is the same as for UJIS. However, if the HiRDB client accepts data of the variable-length character string types (VARCHAR, MVARCHAR, and NVARCHAR), it does not use spaces to clear the SQLDATA area indicated by the SQL descriptor area.
UTF8	The HiRDB client uses shift JIS kanji codes and converts characters codes by assuming that the HiRDB server uses Unicodes (UTF-8). However, if the HiRDB client accepts data of variable-length character string types (VARCHAR and MVARCHAR), it uses the number of spaces equivalent to the SQLLEN value to clear the SQLDATA area indicated by the SQL descriptor area.
UTF8MS	The processing is the same as for UTF8. However, the HiRDB server uses MS-Unicodes, and the HiRDB client uses the Windows encoding character set to convert character codes.
UTF8_TXT	The processing is the same as for UTF8. However, the HiRDB client does not convert character codes for data of fixed-length character string types (CHAR and MCHAR) or variable-length character string types (VARCHAR and MVARCHAR).
UTF8_EX	Processing is the same as for UTF8. However, when the HiRDB client receives a backslash (0x5C) from the HiRDB server, it does not convert the character code, but handles it as a SJIS backslash (0x5C). If the HiRDB client receives a Unicode (UTF-8) backslash (0xC2A5), it converts it to a SJIS backslash (0x5C), in the same way as when UTF8 is specified. When a backslash (0x5C) is entered at the HiRDB client, the character code is not converted and 0x5C is passed to the HiRDB server.
UTF8_EX2	The processing is the same as for UTF8_EX. However, when a SJIS backslash (0x5C) is entered at the HiRDB client, it is converted to a Unicode (UTF-8) backslash (0xC2A5), in the same way as when UTF8 is specified, and it is passed to the HiRDB server.
UTF8MS_TXT	The processing is the same as for UTF8MS. However, the HiRDB client does not convert character codes for data of fixed-length character string types (CHAR and MCHAR) or variable-length character string types (VARCHAR and MVARCHAR).
UCS2_UJIS	To convert character codes, the HiRDB client uses UCS-2 and the HiRDB server uses EUC Japanese kanji codes#. If the HiRDB server uses character codes other than EUC Japanese kanji codes, an error is generated when the HiRDB server is connected. You can specify UCS2_UJIS only when accessing the system from a Unicode-compliant ODBC 3.0 driver, ODBC 3.5 driver, HiRDB.NET data provider, or HiRDB SQL Executer version 02-06 or later.
UCS2_UTF8	To convert character codes, the HiRDB client uses UCS-2 and the HiRDB server uses Unicodes (UTF-8)#. If the HiRDB server uses character codes other than Unicodes (UTF-8), an error is generated when the HiRDB server is connected. You can specify UCS2_UTF8 only when accessing the system from a Unicode-compliant ODBC 3.0 driver, ODBC 3.5 driver, HiRDB.NET data provider, or HiRDB SQL Executer version 02-06 or later.

#

The converted character code range is that of UCS-4.

AUTO is specified when the character code classification of the HiRDB server cannot be identified. UJIS is specified when the character code classification of the HiRDB server can be identified as EUC Japanese kanji codes.

The following character strings are converted:

- Character strings in SQL statements
- Data codes in the SQL descriptor area that are CHAR, VARCHAR, NCHAR, NVARCHAR, MCHAR, or MVARCHAR character strings
- Column names stored in the Column Name Descriptor Area (SQLCNDA)
- Error messages stored in the SQL Communications Area
- Data type names stored in the Type Name Descriptor Area (SQLTNDA)

The following table shows the PDCLTCNVMODE settings in terms of the combination of HiRDB client and HiRDB server character codes.

Character codes used by HiRDB client application	**Character codes at HiRDB server**				
	SJIS	**Unicode (UTF-8)**	**UJIS**	**C**	**GB18030**
SJIS	No conversion necessary	UTF8, UTF8MS, UTF8_TXT, UTF8_EX, UTF8_EX2, UTF8MS_TXT	UJIS, UJIS2	NOUSE	Cannot be specified
Unicode (UTF-8)	Cannot be specified	No conversion necessary	Cannot be specified	NOUSE	Cannot be specified
UJIS	Cannot be specified	Cannot be specified	No conversion necessary	NOUSE	Cannot be specified
UCS-2	Cannot be specified	UCS2_UTF8	UCS2_UJIS	Not needed	Cannot be specified
C	NOUSE	NOUSE	NOUSE	NOUSE	Cannot be specified
GB18030	Cannot be specified	Cannot be specified	Cannot be specified	Cannot be specified	No conversion necessary

Legend:

Cannot be specified: Cannot be specified because conversion is not possible.

No conversion necessary: Does not need to be specified because code conversion is not necessary.

Tables 6-27 and *6-28* show the differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified.

Table 6-27: Differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified (for characters received from a HiRDB server)

Character received from HiRDB server (Unicode (UTF-8))	PDCLTCNVMODE setting	Character code after HiRDB client conversion (SJIS)
0x5C (backslash)	UTF8	0x815F (double-byte backslash)
	UTF8_EX	0x5C (\ symbol)#
	UTF8_EX2	
0xC2A5 (\ symbol)	UTF8	0x5C (\ symbol)
	UTF8_EX	
	UTF8_EX2	

#: Character code is not converted.

Table 6-28: Differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified (for characters entered at the HiRDB client)

Character entered at HiRDB client (SJIS)	PDCLTCNVMODE setting	Character code after HiRDB client conversion (Unicode (UTF-8))
0x5C (\ symbol)	UTF8	0xC2A5 (\ symbol)
	UTF8_EX	0x5C (backslash)#
	UTF8_EX2	0xC2A5 (\ symbol)

#: Character code is not converted.

Notes

- If a data string contains 2-byte external characters, they are replaced with full-sized number signs (#), except when the client environment definitions PDCLTGAIJIDLL and PDCLTGAIJIFUNC are specified in the client. EUC 3-byte external characters cannot be used.
- Single-byte katakana characters are 1-byte codes in the shift JIS kanji codes and 2-byte codes in the EUC Japanese kanji codes. Therefore, if a character string contains single-byte katakana characters, the data length changes when the character codes are converted. If a character string received from

the server contains single-byte katakana characters, the character string becomes shorter after conversion. If a character string to be sent to the server contains single-byte katakana characters, the character string becomes longer after conversion.

In the Unicodes, characters that are not ASCII (0x0-0x7f) characters are expressed as 2- to 4-byte characters. Therefore, the data length changes when the character codes are converted. If a character string received from the server contains non-ASCII characters, the character string becomes shorter after conversion. If a character string to be sent to the server contains non-ASCII characters, the character string becomes longer after conversion.

When the character string length changes, the following processing takes place:

1. When data codes set in the SQL descriptor area are `CHAR` or `MCHAR` character strings:

   If the character string length becomes shorter, the HiRDB client pads the converted codes with single-byte spaces (0x20) until the original character string length is reached. (The length becomes shorter when the HiRDB client receives a UJIS character string containing single-byte katakana characters or a Unicode character string containing non-ASCII characters from the HiRDB server.)

   If the character string length becomes longer, the HiRDB client passes the entire converted character string to the HiRDB server without truncating the character string. (The length becomes shorter when the HiRDB client passes a UJIS-converted character string containing single-byte katakana characters or a Unicode-converted character string containing non-ASCII characters to the HiRDB server.)

   Therefore, an adequate length must be allocated for the column that stores the character string. If all characters can be identified as single-byte katakana characters, the area must have a byte count that is twice the character count (for Unicodes, the area must have a byte count that is three times the character count).

2. When data codes set in the SQL descriptor area are `VARCHAR` or `MVARCHAR` character strings, or for character strings in SQL statements, column names stored in the Column Name Descriptor Area (SQLCNDA), error messages stored in the SQL Communications Area, or data type names stored in the Type Name Descriptor Area (SQLTNDA):

   If the character string becomes shorter, the character string length is changed to the post-conversion character string length.

   If the character string becomes longer, the character string length is changed to the post-conversion character string length.

If all characters can be identified as single-byte katakana characters, the area must have a byte count that is twice the character count (for Unicodes, the area must have a byte count that is three times the character count).

3. For NCHAR or NVARCHAR character strings pointed to from the SQL Descriptor Area (data codes are NCHAR or NVARCHR character strings)

   Because single-byte katakana characters cannot be used, the length remains unchanged, even after conversion.

- If a CHAR or VARCHAR column is used to store binary data, the character code conversion process may produce unexpected conversions when the column is accessed. In this case, disable character code conversion (specify NOUSE in PDCLTCNVMODE).
- Character code conversion cannot be executed on BLOB-type data. For example, if a BLOB column is being used to store text data, have the UAP execute the character code conversion.
- The following two methods are used for character mapping between shift JIS kanji codes and Unicodes:

  JIS method

  This method conforms to the mapping method defined by JIS X 0221. The JIS method is used when a value other than UTF8MS or UTF8MS_TXT is specified in PDCLTCNVMODE.

  Conversion targets: Shift JIS to JIS X0221

  Kanji scope: JIS level-1 kanji set and JIS level-2 kanji set

  MS method

  This method conforms to the mapping method defined by Microsoft. The MS method is used when UTF8MS or UTF8MS_TXT is specified in PDCLTCNVMODE.

  Conversion targets: Windows signed character set to MS-UNICODE

  Kanji scope: JIS level-1 kanji set, JIS level-2 kanji set, IBM expansion kanji, NEC-selected IBM expansion kanji, NEC special characters
- Note that the shift JIS kanji codes and the Windows signed character set have different external character code ranges.
- To apply the MS method, which can handle more kanji characters, specify UTF8MS or UTF8MS_TXT in PDCLTCNVMODE. Before you use the MS method, make sure that you fully understand the problems that may occur as a result of mapping differences.
- A user-defined external character conversion DLL file for UJIS cannot be

applied directly to Unicode external character conversion. To execute Unicode external character conversion, you must use a user-defined external character conversion DLL file to which a Unicode external character conversion function has been added.

Notes when the client uses UCS-2 character codes

- Error messages (`SQLERRMC`) set in the SQL Communications Area may exceed 254 bytes, depending on the character code conversion. In such cases, a maximum of 254 bytes are set for column names and type names and any excess is truncated.
- To receive `CHAR`, `MCHAR`, `VARCHAR`, or `MVARCHAR` data, the SQL description area data length requires twice the maximum definition length.
- Data cannot be received if it exceeds the following applicable value after character code conversion:
    - 30,000 bytes for `CHAR`, `NCHAR`, and `MCHAR`
    - 32,000 bytes for `VARCHAR`, `NVARCHAR`, and `MVARCHAR`

  Therefore, data stored in the server may not be able to be searched if fixed-length character type data exceeds 15,000 bytes, and if variable-length character type data exceeds 16,000 bytes.
- The length of data specified by input parameters cannot exceed the following values:
    - 32,000 bytes for `VARCHAR`, `NVARCHAR`, and `MVARCHAR`
    - 32,000 bytes for `VARCHAR`, `NVARCHAR`, and `MVARCHAR`
- When data codes set in the SQL description area are `CHAR`, `MCHAR`, `VARCHAR,` or `MVARCHAR` and the character strings are sent to the server, the character string length is changed to the post-conversion character string length (for the fixed-length, `SQLLEN` is changed to the post-conversion character string length).
- Do not add `BOM` at the beginning of UCS-2 character strings. Such character strings with `BOM` are not converted correctly. The UCS-2 byte order is processed as the byte order of the host that runs the program.

(28) PDCLTGAIJIDLL=user-defined-external-character-conversion-DLL-file-name

~ <character string>

This environment variable is valid only in the Windows edition.

This environment variable specifies the name of the DLL file for user-defined external character conversion. This environment variable is effective during character code conversion only if a value other than NOUSE or UCS2_UTF8 is specified for PDCLTCNVMODE. If this environment variable is omitted, double-byte external characters are converted to double-byte number signs (#).

(29) PDCLTGAIJIFUNC=*user-defined-external-character-conversion-function-name*

~ <character string>

This environment variable is valid only in the Windows edition.

This environment variable specifies the name of the user-defined external character conversion function. This environment variable is effective for character code conversion only if PDCLTGAIJIDLL is specified.

Descriptive format for a user-defined external character conversion function

The descriptive format for a user-defined external character conversion function is as follows:

```
  _declspec(dllexport)#1 WINAPI#2
user-defined-external-character-conversion-function-name (
     long        direct,
     unsigned char far *instr,
     unsigned char far *outstr) ;
```

#1: The DLL declaration format differs according to the compiler being used. Comply with the DLL format of the compiler being used.

#2: The export function name (user-defined external character conversion function name) of the created DLL differs depending on which compiler is used. Use one of the following methods to check which export function name to specify:

- During DLL creation, specify the project settings so that a MAP file is output. Then check the export function name from the MAP file.
- Use the dumpbin command (dumpbin /exports *DLL-name*) of Microsoft Visual C++ to check the export function name.

Input

direct

Indicates the conversion direction. A value from 1 to 6 is set.

1: Data conversion from the HiRDB client to the HiRDB server

2: Data conversion from the HiRDB server to the HiRDB client

3: Data conversion from the HiRDB client to the HiRDB server (for Unicodes)

4: Data conversion from the HiRDB server to the HiRDB client (for Unicodes)

5: Data conversion from the HiRDB client UCS-2 to the HiRDB server UJIS

6: Data conversion from the HiRDB server UJIS to the HiRDB client UCS-2

7: Data conversion from the HIRDB client UTF-16 (surrogate pair) to the HiRDB server UJIS

Note

When the Gaiji conversion DLL passes a Unicode character, the data is converted to two or four bytes of UCS-2 (UTF-16) format data. Conversion to data in UTF-8 format is performed by a library.

instr

Indicates the pointer to the external character storage area for characters to be converted. The area size is fixed to two bytes when a value other than 7 is specified in direct, and four bytes when 7 is specified.

instr[0] = First byte of external character to be converted

instr[1] = Second byte of external character to be converted

instr[2] = Third byte of external character to be converted

instr[3] = Fourth byte of external character to be converted

outstr

Indicates the pointer to the post-conversion external character storage area. The area size is two bytes when a value other than 6 is specified in direct, and four bytes when 6 is specified.

outstr[0] = First byte of character code (external character) after conversion

outstr[1] = Second byte of character code (external character) after conversion

outstr[2] = Third byte of character code (external character) after conversion

outstr[3] = Fourth byte of character code (external character) after conversion

Even though code conversion could not be performed, set an appropriate value as the converted value (the passed value is used unconditionally).

For UCS-2 (Unicodes) external character codes, byte columns need to be returned with a big endian byte column. For example, for 東, set 0x67 for the first byte and 0x71 for the second byte.

Output

`*outstr`

Stores the converted character string.

Note

The following table shows the character code combinations that can be specified for `*instr` and `*outstr`.

Table 6-29: Character code combinations that can be specified for *instr and *outstr

direct	instr	outstr	PDCLTCNVMODE
1	External character codes of the shift JIS kanji codes	External character codes of the EUC Japanese kanji codes	`UJIS` or `UJIS2`
2	External character codes of the EUC Japanese kanji codes	External character codes of the shift JIS kanji codes	`UJIS` or `UJIS2`
3	External character codes of the shift JIS kanji codes	Unicode external character codes	`UTF8` or `UTF8_TXT`
	External character codes of the Windows signed character set	MS-Unicode external character codes	`UTF8MS` or `UTF8MS_TXT`
4	External character codes of the Unicode BMP (Basic Multilingual Plane)	External character codes of the shift JIS kanji codes	`UTF8` or `UTF8_TXT`
	MS-Unicode external character codes	External character codes of the Windows signed character set	`UTF8MS` or `UTF8MS_TXT`
5	External character codes of the Unicode BMP (Basic Multilingual Plane) and JIS level 3 and 4 kanji codes placed in Unicode BMP	External EUC Japanese kanji character codes	`UCS2_UJIS`
6	External EUC Japanese kanji character codes	Any UTF-16 format codes	`UCS2_UJIS`
7	Unicode surrogate pair	External character codes of the EUC Japanese kanji code	`UCS2_UJIS`

The following table shows the external code ranges for each character code classification.

Table 6-30: External code ranges for each character code classification

Character code	First byte	Second byte
Shift JIS kanji codes	`0xf0` to `0xfc`	`0x40` to `0x7e`
		`0x80` to `0xfc`
Windows signed character set	`0xf0` to `0xfa`	`0x40` to `0x7e`
		`0x80` to `0xfc`
EUC Japanese kanji codes	`0xf5` to `0xfe`	`0xa1` to `0xfe`
Unicodes or MS-Unicodes#	`0xe0` to `0xf8`	`0x00` to `0xff`

#: Because Microsoft has assigned its own characters to codes `0xe000` to `0xe757`, and `0xf8f0` to `0xf8ff`, the user-defined external character conversion DLL is not called for these external character codes.

The following table shows the surrogate pair range.

Table 6-31: Surrogate pair range

Encoding	High-order surrogate		Low-order surrogate	
	Byte 1	Byte 2	Byte 1	Byte 2
UTF-16	0xd8 to 0xdb	0x00 to 0xff	0xdc to 0xdf	0x00 to 0xff

(30) PDCLTGRP=client-group-name

~ <letter> ((1 character))

This environment variable specifies a client group name when the connection frame guarantee facility for client groups is used. The client group name that was specified in the `pdcltgrp` operand of the system definition is specified with an uppercase letter. Even if a lowercase letter is specified, the system assumes that an uppercase letter was specified.

If the `pdcltgrp` operand of the system definition is not specified, or if this environment variable specifies a client group name that is not specified by the `pdcltgrp` operand, the specification for this environment variable becomes invalid. For details about the connection frame guarantee facility for client groups, see the *HiRDB Version 9 System Operation Guide*.

(31) PDTCPCONOPT={0|1}

This environment variable is valid when the HiRDB client connects to a HiRDB server of version 06-02 or later. This environment variable is specified when the number of TCP ports used for communication to the server is to be reduced.

According to the TCP protocol specifications, after a TCP connection ends, the TCP port may switch to TIME_WAIT status for a fixed period of time (1 to 4 minutes) during which it cannot be used in a new connection. The TIME_WAIT-status port is used by the TCP connection that was completed. When PDTCPCONOPT is set to 1, the number of TIME_WAIT-status TCP ports that occur in the HiRDB client and server can be reduced.

0

Do not reduce the number of TCP ports that are used in communication with the HiRDB server.

1

Reduce the number of TCP ports that are used in communication with the HiRDB server.

The following table shows the number of TIME_WAIT-status TCP ports that can be eliminated when 1 is specified.

UAP execution environment	Connection mode from UAP to HiRDB server	Communication type	Environment variable effect	Number of TCP ports in TIME_WAIT status that can be eliminated[#1]	
				Client[#2]	Server
OLTP	Normal connection	Communication for connecting UAP to HiRDB server[#3]	V	1	1
		Failure recovery communication from OLTP to HiRDB server[#4, #5]	V	1	1
	High-speed connection and FES host direct connection	Communication for connecting UAP to HiRDB server	I	--	--
		Failure recovery communication from OLTP to HiRDB server[#4, #5]	V	1	1

UAP execution environment	Connection mode from UAP to HiRDB server	Communication type	Environment variable effect	Number of TCP ports in TIME_WAIT status that can be eliminated#1	
				Client#2	Server
Other	Normal connection	Communication for connecting UAP to HiRDB server#3	V	1	1
	High-speed connection and FES host direct connection	Communication for connecting UAP to HiRDB server	I	--	--

Legend:

V: Becomes valid when 1 is specified for PDTCPCONOPT.

I: Becomes invalid even if 1 is specified for PDTCPCONOPT.

--: Does not apply.

#1: The number of TCP ports that are switched to TIME_WAIT status depends on the timing when packets that participate in the termination protocol for TCP connections arrive. Therefore, the number changes according to the network status. Consequently, the number of TIME_WAIT-status TCP ports that can be deleted may change.

#2: During failure recovery communication from OLTP, the OLTP failure recovery process becomes the client.

#3: Some of the TCP ports that are used when the UAP connects to the HiRDB server are switched to TIME_WAIT status.

#4: Failure recovery communication from OLTP to the HiRDB server takes place when the OLTP failure recovery process calls an X/Open-compliant XA interface function (such as xa_open, xa_recover, or xa_rollback) to recover a transaction interrupted by a failure. At this time, some of the TCP ports used in XA interface execution are switched to TIME_WAIT status. The number of TIME_WAIT-status TCP ports that can be eliminated is the number that can be eliminated when one XA interface function is called. Therefore when *n* XA interface functions are called, *n* times that number can be eliminated.

#5: The method of specifying the environment variable for the OLTP failure recovery process differs in each OLTP environment. For example, in OpenTP1, the environment variable is specified in a transaction service definition.

Application standard

Specify 1 in PDTCPCONOPT if either of the following conditions is satisfied:

- If the number of TCP ports that the OS allocates automatically is less than 5000 (the TCP port range differs according to the OS)
- If `PDXAMODE` is set to `1` in a UAP under OpenTP1

However, if the specification value of the `pd_max_users` operand in the system definitions is less than 100, or if the `pd_registered_port` operand is specified, you do not need to specify `1` even if one of the above conditions is satisfied.

Notes

1. If the version of the HiRDB server to be connected is earlier than version 06-02, do not specify `1` in `PDTCPCONOPT`. If you specify `1`, a shortage may occur in the communication sockets that the HiRDB server can use.
2. When you specify 1 in `PDTCPCONOPT`, you must check and, if necessary, revise the value of the `maxfiles_lim` operating system parameter in the HiRDB server. For details about estimating values for operating system parameters, see the *HiRDB Version 9 Installation and Design Guide*.

(32) PDAUTORECONNECT={YES|NO}

This environment variable specified whether or not the automatic reconnect facility is to be used.

For details about the automatic reconnect facility, see *4.15 Automatic reconnect facility*.

`YES`

Use the automatic reconnect facility.

When this facility is used, it automatically reconnects the HiRDB client to the HiRDB server if the connection is disconnected because of a service process failure, system switchover, or network failure.

`NO`

Do not use the automatic reconnect facility.

Application standard

Apply the automatic reconnect facility if the HiRDB server is executing the system reconfiguration command (`pdchgconf`) or updating to the HiRDB update version (`pdprgcopy` or `pdprgrenew`). If the automatic reconnect facility is used in this situation, the HiRDB client can continue processing without returning an error to the UAP, even if the connection with the HiRDB server is disconnected.

Notes

1. Use `PDRCCOUNT` and `PDRCINTERVAL` to specify the number of `CONNECT` statement retries and the retry interval when reconnection is executed.

2. The time during which the automatic reconnect facility operates with SQL statements other than the CONNECT statement is monitored based on the PDCWAITTIME time. If the PDCWAITTIME time is exceeded, automatic reconnect processing is aborted.

3. If automatic reconnect fails, an error indicating the cause is returned to the UAP.

4. If specified, this environment variable is ignored and NO is assumed for an application that uses an API that is X/Open-compliant to access the HiRDB server.

5. If one of the following conditions is satisfied, the automatic reconnect facility is enabled only when the CONNECT statement is executed:

   - The HiRDB server version is earlier than 07-00.
   - The XDM/RD E2 connection facility is being used.
   - The XDM/RD E2 version is 10-02 or earlier.

(33) PDRCCOUNT=CONNECT-retry-count-with-automatic-reconnect-facility

~ <unsigned integer> ((1-200)) <<5>>

This environment variable specifies the number of times the CONNECT statement is retried during reconnection by the automatic reconnect facility. This environment variable becomes effective when PDAUTORECONNECT=YES is specified.

(34) PDRCINTERVAL=CONNECT-retry-interval-with-automatic-reconnect-facility

~ <unsigned integer> ((1-600)) <<5>> (seconds)

This environment variable specifies the CONNECT retry interval at which the automatic reconnect facility executes reconnect processing. The interval is specified in units of seconds. This environment variable becomes effective when PDAUTORECONNECT=YES is specified.

(35) PDUAPENVFILE=UAP-environment-definition-file-name

~ <identifier> ((up to 8 characters))

This environment variable specifies the name of the UAP environment definition file that defines the execution environment if the UAP is to be executed in a separate environment. Specifying PDUAPENVFILE allows you to switch the execution environment of each UAP.

For details about UAP environment definitions, see the manual *HiRDB Version 9 System Definition*.

If the UAP environment definitions contain an error, a definition error occurs during CONNECT execution. If the UAP environment definition file does not contain any definitions, the PDUAPENVFILE specification is ignored.

Uppercase and lowercase characters in the UAP environment definition file name are not discriminated in HiRDB for Windows systems. Note therefore that files that have the same name except for case differences are treated as the same file.

(36) PDDBBUFLRU={YES|NO}

This environment variable specifies whether or not the LRU method for global buffers is to be changed for each UAP in an OLTP environment.

`YES`:

Use the LRU method.

`NO`:

Do not use the LRU method. In this case, pages that do not hit the buffer become the target for being flushed out of the global buffer regardless of the access frequency when the global buffer becomes full. For that reason, the number of pages to be cached in the global buffer can be minimized.

Application standard

You will usually omit this environment variable (use the LRU method). In the OLTP environment, when a large number of searches are performed or a large number of UAP updates are executed using the global buffer, the most recent contents cached in the global buffer are flushed out, which may cause a temporary drop in system performance. In order to avoid this, specify `PDDBBUFLRU=NO` for a UAP that performs a large volume of searches or updates in an OLTP environment.

Notes

1. Pages accessed by a UAP that does not use the LRU method are subject to being flushed out of the global buffer regardless of the access frequency. For that reason, a UAP that does not use the LRU method could cause a drop in response performance, due to an increase in the number of inputs/outputs caused by the drop in the buffer hit ratio.
2. SQL processing by a UAP secures 1 to 4 global buffer sectors simultaneously. Therefore, even though the UAP does not use the LRU method, pages cached in the global buffer for each UAP may be flushed out from the 1 to 4 global buffer sectors.
3. If the LRU method is not used for a UAP to be updated, writing to the database becomes frequent. For that reason, log output triggers occur frequently compared to when the LRU method is used and the amount of output log information increases. In such a case, a lack of system log file capacity may occur, so you should take one of the following actions:
    - Re-evaluate the size of the system log file

- Specify NO in the PDDBLOG operand of the client environment definition.

The formula is shown below for estimating the log size when the LRU method is not used. Note that when the system definition's pd_log_rec_leng operand is set to 1,024, the amount of output log information when the LRU method is not used can be minimized.

Updated-GET-count# x *value-of-pd_log_rec_leng-operand*

#: You can check the updated GET count from the DIDUC value of the UAP statistics report, or from the DIDUC value of the UAP statistical information.

(37) PDHATRNQUEUING=NO

When queuing is specified in the pd_ha_transaction operand of the system definition, this environment variable is specified when application of the transaction queuing facility is to be changed for each client. If the transaction queuing facility is not to be applied to a client, specify NO.

NO

Do not apply the transaction queuing facility during connection processing from the client.

For details about the transaction queuing facility, see the *HiRDB Version 9 System Operation Guide*.

(38) PDCLTBINDLOOPBACKADDR={YES|NO}

This environment variable specifies whether a loopback address is to be used for bind() when the receive port used for communication with the HiRDB server is created.

YES

Use the loopback address to execute bind().

NO

Do not use the loopback address for bind().

Specify YES in this environment variable when you have specified a loopback address in the pdunit -x operand in the system common definition.

This environment variable is applicable to the Windows edition of HiRDB server. When you specify this environment variable, see the *HiRDB Version 9 Installation and Design Guide* for information about registration in the Windows firewall exception list.

(39) PDASTHOST=HiRDB-Control-Manager-Agent-host-name[,secondary-system-HiRDB-Control-Manager-Agent-host-name]

~ <identifier> <<PDHOST specification value>>

When a UAP executes a command, this environment variable specifies the host name of the HiRDB Control Manager-Agent to be connected. The `COMMAND EXECUTE` statement of SQL is used when a UAP executes a command.

When a UAP executes a command, the HiRDB Control Manager-Agent actually executes that command.

For HiRDB/Parallel Server, the host name of the server machine that contains the system manager is specified.

In addition to the host name, you can specify the FQDN or the IP address. The specification methods are as follows:

Host name

The host name that was specified in the `pdunit -x` operand of the system common definition must be specified.

Example:

```
PDHOST=host1
```

FQDN:

The FQDN is comprised of the host name and domain name of a HiRDB server, separated by periods.

Example:

```
    PDASTHOST=host1.soft.hitachi.co.jp
```

IP address

The IP address is specified with a decimal number that has each group of 3 digits separated with a period.

Example:

```
PDHOST=172.18.131.34
```

System switchover without IP address inheritance

Specify the host names of both the primary system and the secondary system. If you specify only the host name of the primary system, you must change the specification of this environment variable after system switching occurs. After system switching, change the host name to that of the new primary system.

(40) PDASTPORT=HiRDB-Control-Manager-Agent-port-number

~ <unsigned integer> ((5001-49999))

This environment variable specifies the port number of the HiRDB Control Manager - Agent to be connected when a command is executed from a UAP.

Specify a port number that is registered in the `services` file (for the UNIX edition, `/`

`etc/services`; for the Windows edition, `%windir%\system32\drivers\etc\services`).

(41) PDSYSTEMID=HiRDB-identifier-of-HiRDB-server-managed-by-HiRDB-Control-Manager-Agent

~ <identifier> ((4 characters))

When a command is executed from a UAP, this environment variable specifies the HiRDB identifier of the HiRDB server being managed by the HiRDB Control Manager - Agent to be connected. Specify the HiRDB identifier with the `pd_system_id` operand of the system definitions.

(42) PDASTUSER=OS-user-name/password

~ <<`PDUSER` specification value>>

This environment variable specifies the user name and password for the OS that runs commands executed from a UAP. This must be the user name and password for an OS that has the execution privilege for the commands. Specify in the format *user-name/password*.

If a password specification is not required (i.e., the setting is for a user without a password), the password can be omitted.

The user name and password for an OS are handled as upper case characters regardless of whether the specification is in upper case or lower case. However, if lower-case characters are enclosed in quotation marks, they are handled as lower-case characters.

(43) PDCMDWAITTIME=maximum-client-wait-time-during-command-execution

~ <unsigned integer> ((0, 6-43200)) <<0>> (minutes)

When a command is executed from a UAP, this environment variable specifies, the maximum time that the HiRDB client waits for a response from the HiRDB Control Manager - Agent after it sends a request to the server.

If `0` is specified, the HiRDB client continues to wait until a response is returned from the HiRDB Control Manager - Agent.

If there is no response from HiRDB Control Manager - Agent after the specified amount of waiting time has elapsed, an error is returned to the client (UAP). If a command in the UAP is still processing at that time, either HiRDB Control Manager - Agent or the command must be canceled.

(44) PDCMDTRACE=command-trace-file-size

~ <unsigned integer> ((0, 4096-2000000000)) (bytes)

When a command is executed from a UAP, this environment variable specifies, the size of the command trace output file.

If `0` is specified, the maximum file size is assumed, and a command trace that exceeds the maximum size is not output. If a value from 4,096 to 2,000,000,000 is specified, the specified value becomes the file size, and the output destination switches when the file size exceeds the specified value. If this environment variable is omitted, a command trace is not collected.

For details about command traces, see *11.1.5 Command trace facility*.

Relationship with other environment variables

The command trace output file is created in the directory specified by PDCLTPATH. If PDCLTPATH is omitted, the file is created in the current directory when the UAP is executed. (If the UAP is executed from OpenTP1, the file is created under the OpenTP1 installation directory \tmp\home\server-namexx.)

(45) PDIPC={MEMORY|DEFAULT}

This environment variable specifies the inter-process communication method to be used when the server and client are found in the same host.

MEMORY

Use the memory for inter-process communication. This is called the inter-process memory communication facility.

DEFAULT

Use the default communication method (TCP/IP or PIPE) in each platform for inter-process communication.

Notes

1. If the client and server are not on the same host, the PDIPC specification is ignored (the system assumes DEFAULT). In this case, the connection process may take longer.
2. If you use the multithread-capable XA interface libraries (pdcltxm.dll for Windows edition clients or libzcltxk.sl(so) or libzcltyk.sl(so) for UNIX edition clients) to access HiRDB with the XA interface, and a UAP running on TPBroker for C++ or Weblogic Server is set as the client, the specification of this environment variable is ignored, and HiRDB assumes that DEFAULT was specified for PDIPC.
3. If PDIPC=MEMORY is specified in UNIX-edition clients, HiRDB allocates a common memory size equal to values specified for PDSENDMEMSIZE and PDRECVMEMSIZE, for each client connection. Consequently, a common memory shortage may occur if multiple clients are executed concurrently. To avoid a memory shortage, consider the common memory size that can be used when specifying PDSENDMEMSIZE and PDRECVMEMSIZE.
4. If PDIPC=MEMORY is specified, the specification of PDCLTRCVADDR is ignored.

5. If `PDIPC=MEMORY` is specified, and concurrently `p`, `r`, or `a` is specified in `PDUAPREPLVL` or `PDWRTLNFILSZ` is specified, the specification for `PDIPC` becomes invalid.

(46) PDSENDMEMSIZE=data-send-memory-size-in-client

~ <unsigned integer> ((4-2097152)) <<16>> (kilobytes)

This environment variable specifies the data storage area size, in multiples of 4 KB, to be used when the client sends data to the server, when the inter-process memory communication facility is used. This environment variable becomes effective when `PDIPC=MEMORY` is specified.

If the specified value is not a multiple of 4, the value is rounded up to multiple of 4.

If data larger than the size specified here is sent, the inter-process memory communication facility cannot be used. (The communication method for `PDIPC=DEFAULT` is used.)

Estimation method

Estimate the value to be specified for this environment variable based on the following formula:

specification-value (bytes) = ↑(400 + 16 x *number-of-retrieved-columns* + 16 x *number-of-?-parameters* + *SQL-statement-length*) ÷ 4096↑ x 4

The value calculated with this formula differs from the data size that is actually sent during communication.

(47) PDRECVMEMSIZE=data-receive-memory-size-in-client

~ <unsigned integer> ((4-2097152)) <<32>> (kilobytes)

This environment variable specifies the data storage area size, in multiples of 4 KB, to be used when the client receives data from the server, when the inter-process memory communication facility is used. This environment variable becomes effective when `PDIPC=MEMORY` is specified.

If the specified value is not a multiple of 4, the value is rounded up to a multiple of 4.

If data larger than the size specified here is received, the inter-process memory communication facility cannot be used. (The communication method for `PDIPC=DEFAULT` is used.)

Estimation method

Estimate the value to be specified for this environment variable based on the following formula:

specification-value (bytes) = ↑(600 + 25 x *number-of-retrieved-columns* + Σ *column-data-lengths*) ÷ 4096↑ x 4

If the data type of *column-data-length* is VARCHAR, replace *column-data-length* with *structure-length* in the preceding formula. If the HiRDB client accepts array FETCH statements or repetition columns, use *column-data-length* x *number-of-array-columns* or *column-data-length* x *number-of-repetition-column-elements*.

If PDBLKF is specified, calculate the value based on the following formula:

specification-value (bytes) = ↑(600 + 19 x *number-of-retrieved-columns* + (7 x *number-of-retrieved-columns* + Σ *column-data-lengths*) x *PDBLKF-value*) ÷ 4096↑ x 4

The value calculated with this formula differs from the data size that is actually sent during communication.

(48) PDCWAITTIME=maximum-client-wait-time

~ <unsigned integer> ((0-65535)) <<0>> (seconds)

This environment variable specifies the maximum time that the HiRDB client waits for a response from the HiRDB server after sending a request to the HiRDB server. Specify PDCWAITTIME when implementing interval monitoring of long running SQL statements.

Estimation method

Specify the value that satisfies the condition shown below, because this will help you identify the cause of slow SQL processing:

`watch_time` or `tran_expiration_time` in the OpenTP1 system definition > `PDCWAITTIME` > `pd_lck_wait_timeout`

If you do not use OpenTP1, you can ignore the OpenTP1 system definition.

Notes

1. When 0 is specified, the HiRDB client continues to wait until it receives a response from the HiRDB server. If the HiRDB client does not receive a response from the HiRDB server before the maximum wait time elapses, the HiRDB client returns an error to the UAP. If this occurs during transaction processing, the process in the HiRDB server is cancelled.

2. When 0 is specified, the no response status may be set in the HiRDB client if one of the following errors occurs:

   - Communication error (communication error between a HiRDB client and a HiRDB server or between two HiRDB servers (including temporary errors))

   - Process not responding because of a disk error

Hitachi therefore recommends that you specify a nonzero value that is larger than maximum SQL execution time. If the UAP executes an SQL statement for which lock-release wait occurs, you must also consider the `pd_lck_wait_timeout` operand value in the system definitions when determining the `PDCWAITTIME` value.

(49) PDSWAITTIME=maximum-server-wait-time-during-transaction-processing

~ <unsigned integer> ((0-65535)) <<600>> (seconds)

This environment variable specifies the maximum time that the HiRDB server waits for the next request from the HiRDB client to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time during transaction processing (from startup of SQL execution to `commit` or `rollback`). The monitoring time is reset when the request from the HiRDB client arrives at the HiRDB server.

If the HiRDB server does not receive a request within the specified amount of time, it assumes that an error occurred in the UAP and rolls back the current transaction. The HiRDB server also severs the connection with the HiRDB client without notifying the HiRDB client.

If `0` is specified, the HiRDB server continues to wait until it receives a request from the HiRDB client. If the client machine or network is shut down, that shutdown status might not be detected.

If the specified value is too large and the client machine or network is shut down, it will take some time to detect the shutdown status.

Specify `PDSWAITTIME` to avoid process survival.

Notes

1. When the block transfer facility (`PDBLKF`) is used, the HiRDB client executes `FETCH` statement processing until all rows that were block-transferred from the HiRDB server are processed. The HiRDB client does not send another request to the HiRDB server until the `FETCH` statement processing ends. Therefore, if the block transfer facility is used, the value specified in this environment variable must include the amount of time the `FETCH` statement requires to process the number of blocks that will be transferred.
2. This environment variable must be specified for the operating mode in which the client is a UAP under OLTP. Otherwise, the default value of 600 seconds is used, and connections may be severed inappropriately.

(50) PDSWATCHTIME=maximum-server-wait-time-outside-transaction-processing

~ <unsigned integer> ((0-65535)) (seconds)

This environment variable specifies the maximum time that the HiRDB server waits for the next request from the HiRDB client to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time outside transaction processing (i.e., outside the interval from start of SQL execution to `commit` or `rollback`). The monitoring time is reset when the request from the HiRDB client arrives at the HiRDB server.

If the HiRDB server does not receive a request within the specified amount of time, it assumes that an error occurred in the UAP and severs the connection with the HiRDB client without reporting the disconnection to the HiRDB client.

Specify `0` for a UAP for which the client is always connected (such as a resident SPP in OpenTP1 or a Cosminexus UAP using a connection pool). If `0` is specified, the HiRDB server continues to wait until it receives a request from the HiRDB client.

If `0` is specified and the client machine or network is shut down, that shutdown status might not be detected.

If the specified value is too large and the client machine or network is shut down, it will take some time to detect the shutdown status.

Specify `PDSWATCHTIME` to avoid process survival.

Notes

1. This environment variable must be set to `0` for the operating mode in which the client is a UAP under OLTP, or if the UAP always connects to the HiRDB server regardless of whether a transaction is being processed.
2. If the HiRDB server disconnects the connection with the HiRDB client, it does not report the disconnection to the HiRDB client.

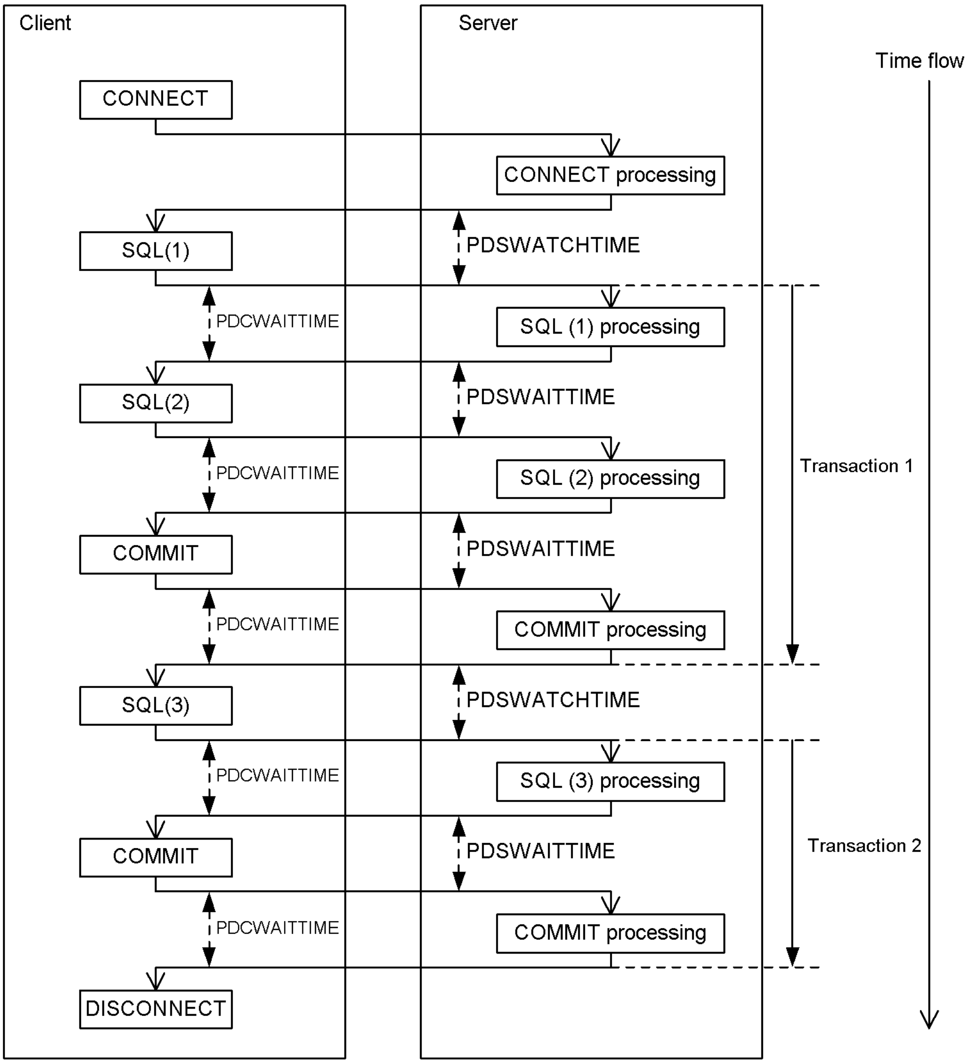
Relationship to the system definition

If this environment variable is omitted, the HiRDB server uses the value that was specified in the `pd_watch_pc_client_time` operand of the system definition and monitors processing until the start of a transaction. For details about the `pd_watch_pc_client_time` operand, see the manual *HiRDB Version 9 System Definition*.

Relationship with other environment variables

The following figure shows the relationships among `PDCWAITTIME`, `PDSWAITTIME`, and `PDSWATCHTIME`.

Figure 6-2: Relationships among PDCWAITTIME, PDSWAITTIME, and PDSWATCHTIME



PDSWATCHTIME: The server monitors the time outside transaction processing (i.e., time outside the interval from start of SQL execution until commit or rollback). If the server does not receive a request from the client within the specified amount of time, it severs the connection with the client.

PDCWAITTIME: The client monitors the time from when it sent a request to the server until it receives a response. If the client does not receive a response from the server within the specified amount of time, it severs the connection with the server.

PDSWAITTIME: The server monitors the time from when it issued a response to a request from the client until it receives the next request from the client. If the server does not receive a request from the client within the specified amount of time, it severs the connection with the client.

(51) PDCWAITTIMEWRNPNT=output-timing-for-SQL-runtime-warning

This environment variable specifies the output time of the SQL runtime warning information file when the SQL runtime warning output facility is used.

The SQL runtime warning output facility outputs an SQL runtime warning information file and a warning message (KFPA20009-W) if the runtime of an SQL statement exceeds a fixed time. For details about the SQL runtime warning output facility, see the *HiRDB Version 9 System Operation Guide*.

Use one of the following methods to specify the output timing of the SQL runtime running information file:

Percentage of the PDCWAITTIME specification value (when the decimal point is not specified)

~ <unsigned integer> ((0-99)) (%)

Specify the timing as a percentage of the PDCWAITTIME specification value. For example, if you specify 100 (seconds) in the PDCWAITTIME operand and 90 (%) in PDCWAITTIMEWRNPNT, HiRDB checks the SQL runtime after it executes an SQL statement. If the result indicates that the SQL runtime was 90 seconds or longer but less than 100 seconds, HiRDB outputs warning information.

Percentage of the PDCWAITTIME specification value (when the decimal point is specified)

~ <unsigned decimal number> ((0-99.999999)) (%)

Specify the timing as a percentage (including the decimal point) of the PDCWAITTIME specification value.

Output time of SQL runtime warning

~ <unsigned decimal number> ((0-PDCWAITTIME)) (seconds)

Specify the output time for the SQL runtime warning. (For example, if the output time is 60 seconds, specify PDCWAITTIMEWRNPNT=60sec.) A decimal point can be specified in the time specification. The specified value must be less than the

PDCWAITTIME specification value.

Relationship with system definitions

When PDCWAITTIMEWRNPNT is omitted, the specification value of the pd_cwaittime_wrn_pnt operand in the system definitions is assumed. For details about the pd_cwaittime_wrn_pnt operand, see the manual *HiRDB Version 9 System Definition*.

(52) PDKALVL={0|1|2}

This environment variable specifies whether the facility that periodically sends packets from the HiRDB client to the HiRDB server is to be used.

This environment variable is effective only when the multi-thread versions of the HiRDB client libraries are used.

If a value other than 0 is specified, one packet sending thread is generated for each connection with HiRDB. The packet send interval can be specified in PDKATIME.

If specified, this environment variable is ignored and 0 is assumed for an application that uses an API that is X/Open-compliant to access HiRDB.

0

Do not use the facility that sends packets periodically.

1

Use the facility that sends packets periodically. The packet transmission thread sends packets to the connection path with the HiRDB server at fixed time intervals.

HiRDB does not reset the PDSWAITTIME and PDSWATCHTIME monitoring times that the HiRDB server uses for time monitoring.

If the HiRDB client and the HiRDB server are installed in the same machine, do not specify 1.

2

Use the facility that sends packets periodically. The packet transmission thread sends packets to the connection path with the HiRDB server at fixed time intervals and receives packets returned from the HiRDB server.

HiRDB resets the PDSWAITTIME and PDSWATCHTIME monitoring times that the HiRDB server uses for time monitoring.

If a packet from the HiRDB server is not returned within the PDCWAITTIME time specified in the client environment definitions, the connection is invalidated. If this happens, the timeover SQLCODE (-732) is returned to the application when the SQL execution thread executes the next SQL statement.

If the SQL execution thread receives an SQL request from the application while the packet transmission thread is waiting for a response from the HiRDB server, the SQL execution thread is set to wait status until the packet transmission thread receives a response from the HiRDB server. Consequently, the SQL runtime may be delayed. Also, because the `select()` system call is issued during the reception wait period, the CPU usage is higher than when `1` is specified as the setting value. If the `PDSWAITTIME` and `PDSWATCHTIME` monitoring times that the HiRDB server uses for time monitoring do not need to be reset, Hitachi recommends that you specify `1` as the setting value.

Application standard

Network management applications such as routers and firewalls sometimes feature an idle-time monitoring facility that disconnects the connection if there is no packet flow for a fixed period of time. By specifying a value other than `0` in `PDKALVL`, you can retain the HiRDB connection and prevent a Web application waiting for a service request from using the network management application to improperly disconnect the HiRDB connection.

When the time-monitoring environment variables (`PDSWAITTIME` and `PDSWATCHTIME`) are set to infinite in the HiRDB server, uncompleted processes may still remain in the HiRDB server if the HiRDB client machine fails or a network failure occurs. By specifying `2` in `PDKALVL`, you can avoid connection disconnect by the time monitoring facilities of the HiRDB server without having to set the time-monitoring values in the HiRDB server to infinite.

Application examples

1. If the following conditions apply, specify `1` in `PDKALVL` and specify a time that is shorter than the firewall monitoring time in `PDKATIME`. (For example, if the firewall monitoring time is 1,200 seconds, specify `1,000` seconds in `PDKATIME`.)
    - The Web application issues SQL execution requests to the DB server at irregular times, and no SQL statements are executed for long periods of time.
    - A firewall has been set up between the Web server and the DB server, and the firewall disconnects the connection if there is no packet flow for a fixed period of time.
2. If the following conditions apply, specify `2` in `PDKALVL` and specify a time that is shorter than the `PDSWATCHTIME` monitoring time in `PDKATIME`. (For example, if the `PDSWATCHTIME` monitoring time is 3,600 seconds, specify `3,000` seconds in `PDKATIME`.)
    - A connection-pooling application accesses HiRDB.
    - A connection is reused for each SQL execution request but is sometimes disconnected according to the `PDSWATCHTIME` monitoring time because the connection is not used for a long time.

(53) PDKATIME=packet-send-interval

~ <unsigned integer> ((60-65535)) <<3000>> (seconds)

This environment variable specifies the interval at which the HiRDB client regularly sends packets to the HiRDB server. The interval is specified in units of seconds. Specify a time that is shorter than the reset monitoring time.

`PDKATIME` is enabled when a value other than `0` is specified in `PDKALVL`.

If the SQL execution thread is executing an SQL statement when a packet is scheduled to be sent, the packet transmission thread does not send the packet and instead waits until the next transmission time.

(54) PDTIMEDOUTRETRY=retry-count

~ <unsigned integer> ((0-32767)) <<2>>

This environment variable specifies the number of times the `connect()` system call can be retried when a `WSAETIMEDOUT` error (`ETIMEDOUT` error for the UNIX edition) of `winsock` occurs in the case of a `connect()` system call that is executed when a HiRDB client connects to the HiRDB server.

Benefit

When `connect()` system calls to the HiRDB server become too great, filling the `listen` queue, a `WSAETIMEDOUT` error or `ETIMEDOUT` error is returned from `connect()`. Such a connection error can be avoided by retrying the `connect()` system call.

Note

In the event of a `WSAETIMEDOUT` error or `ETIMEDOUT` error that occurs due to a network failure or server machine power outage, the return from the `connect()` system call may take some time. Therefore, if a large number of retries is set, it may take a while for a connection error to be returned to the UAP. If the IP address for client connection is not inherited during system switchover after a network failure, switchover to the standby system will take a long time. In such an environment, you can reduce the time required for switchover to the standby system by specifying a small number of retries.

(55) PDNBLOCKWAITTIME=connection-establishment-monitoring-time-in-nonblock-mode

~ <unsigned integer> ((0-120)) <<0>> (seconds)

This environment variable specifies the connection establishment monitoring time in nonblock mode when connection completion between the HiRDB server and client is monitored.

If `1` or a higher value is set for this environment variable, the communication between the HiRDB server and client is set to nonblock communication and completion of the `connect()` system call is monitored. This is called the *nonblock mode*. If `0` is specified, the system waits until the timeout time of the OS for the connection to be completed. This is called the *block mode*.

Application standard

Specify this environment variable (set the nonblock mode) if you want to avoid having the `connect()` system call wait several tens of seconds (the actual time depends on the OS) if a LAN failure occurs. Specifying this environment variable allows the system to detect LAN failures earlier. If specified in the nonblock mode, `PDTIMEDOUTRETRY=0` is always assumed.

Estimation method

If the specified value is too small, an unwarranted error may occur depending on the network status. Set a value higher than the value obtained from the following calculation expression:

MAX(A + 1, 8)

A:

Arrival time between the HiRDB server and client as measured by an OS command such as `ping`. The arrival time of `ping` and other commands fluctuates depending on the network load. Assume the highest load status when measuring the arrival time.

(56) PDCONNECTWAITTIME=maximum-wait-time-in-HiRDB-client-during-server-connection

~ <unsigned integer> ((1-300)) <<300>> (seconds)

This environment variable specifies the maximum wait time that the HiRDB client waits for a response from the HiRDB server when it connects with the HiRDB server.

If a system switchover or a system failure occurs after the HiRDB server accepts a connection request from the HiRDB client, the HiRDB client waits only the specified amount of time for a response.

Application standard

If the system switchover facility is being used, specify this environment variable to allow applications to detect failures early. If this environment variable is specified together with `PDNBLOCKWAITTIME`, failures are detected even earlier.

Estimation method

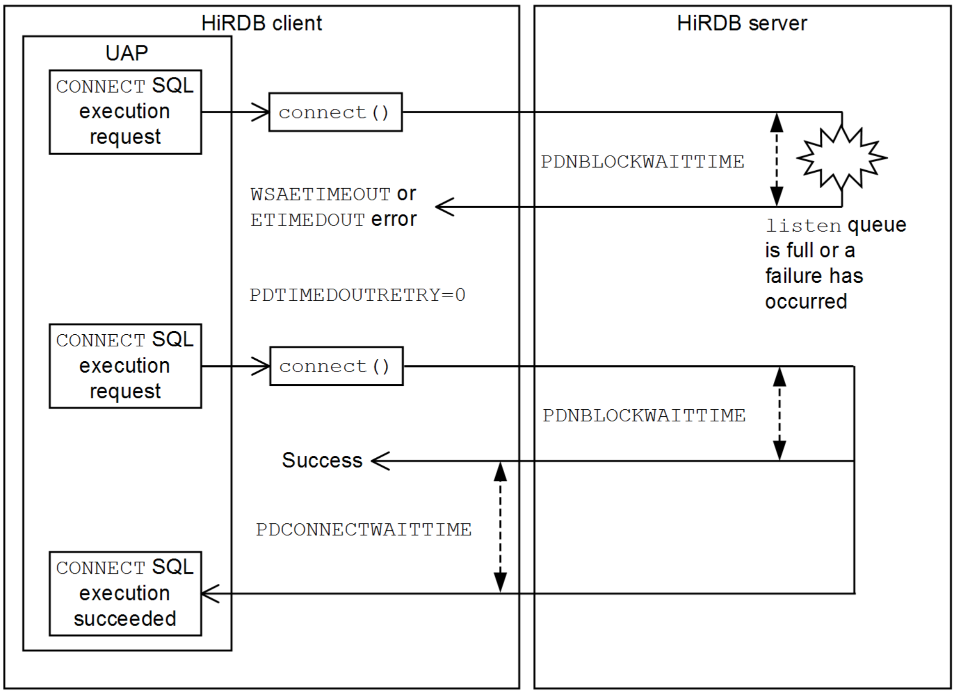
If the specified value is too small, normal connection processing may result in an error if the processing takes too long because of the network status or the scheduling wait during connection processing. Set a value higher than the value obtained from the following calculation expression:

MIN(*value-of-pd_max_users-operand-in-system-definition* x 0.2, 300)

Relationship with other environment variables

The following figure shows the relationships among PDTIMEDOUTRETRY, PDNBLOCKWAITTIME, and PDCONNECTWAITTIME.

Figure 6-3: Relationships among PDTIMEDOUTRETRY, PDNBLOCKWAITTIME, and PDCONNECTWAITTIME


HiRDB client
HiRDB server
UAP
CONNECT SQL execution request
connect()
PDNBLOCKWAITTIME
WSAETIMEOUT or ETIMEDOUT error
listen queue is full or a failure has occurred
PDTIMEDOUTRETRY=0
CONNECT SQL execution request
connect()
PDNBLOCKWAITTIME
Success
PDCONNECTWAITTIME
CONNECT SQL execution succeeded

Explanation:

PDNBLOCKWAITTIME: Monitors until the connect() system call terminates. Failures can be detected quickly in the nonblock mode with 1 or a greater value specified.

PDTIMEDOUTRETRY: Specifies the number of times the request is to be retried after a WSAETIMEOUT or ETIMEDOUT error occurs. In the nonblock mode, 0 is always assumed whether or not this environment variable is specified.

PDCONNECTWAITTIME: The client waits for a response from the server for the specified amount of time.

(57) *PDCLTPATH=trace-file-storage-directory*

~<path name> ((path name of current directory))

This environment variable specifies the storage directory for the SQL trace file and client error log file created by the HiRDB client.

(58) *PDSQLTRACE=SQL-trace-file-size*

~<unsigned integer> ((0, 4,096-2,000,000,000)) (bytes)

This environment variable specifies the size of the SQL trace file into which SQL trace information for the UAP is to be output.

If 0 is specified, the maximum file size is assumed, and an SQL trace that exceeds the maximum size is not output. If a value from 4,096 to 2,000,000,000 is specified, the specified value becomes the file size, and the output destination switches when the file size exceeds the specified value. When this environment variable is omitted, the SQL trace is not output.

For details about the SQL trace, see *11.1.1 SQL tracing*.

Relationship with other environment variables

The SQL trace is output to the directory specified by PDCLTPATH. If no value is specified for PDCLTPATH, the SQL trace is output to the current directory when the UAP is started. (When the UAP is started from OpenTP1, the current directory is %PDDIR%\tmp\home\server-namexx.)

Estimation method

Calculate the size of the SQL trace file from the number of SQL statements to be collected. For each SQL statement to be collected, calculate the size of the individual rows (80 bytes) and the size of the SQL statement, and use the overall total as an estimate for the value to be specified.

(59) PDUAPERLOG=*client-error-log-file-size*

~ <unsigned integer> ((0, 4096-2000000000)) <<65536>> (bytes)

This environment variable specifies the size (in bytes) of the client error log file to which the UAP error logs are to be output.

If 0 is specified, the maximum file size is assumed, and a client error log that exceeds the maximum size is not output. If a value from 4,096 to 2,000,000,000 is specified, the specified value becomes the file size, and the output destination switches when the file size exceeds the specified value.

For details about client error logs, see *11.1.2 Client error log facility*.

Relationship with other environment variables

Client error logs are output to the directory specified in PDCLTPATH. If no value is specified for PDCLTPATH, the SQL trace is output to the current directory when the UAP is started. (When the UAP is started from OpenTP1, the current directory is %PDDIR%\tmp\home\server-namexx.)

(60) PDERRSKIPCODE=SQLCODE[,SQLCODE]...

This environment variable specifies SQLCODEs for which message output to the client error log is to be suppressed. Up to 10 SQLCODEs can be specified.

For example, to suppress SQLCODEs -901 and -917, specify this environment

variable as follows:
`PDERRSKIPCODE=-901,-917`

Benefits

Depending on the UAP structure, there are errors that will inevitably occur during SQL processing. If this type of error occurs frequently during normal processing, the file system may be overwhelmed. Especially for a UAP that uses an API that is X/Open-compliant, two client error log files are created for each process. If this environment variable is specified, message output can be suppressed for specific errors, and the load on the file system can be reduced.

Application standard

Apply this environment variable if both of the following conditions are satisfied:

- Errors occur frequently because of the UAP structure.
- The cause of an error can be identified beforehand, and there is no need to investigate the cause.

When this environment variable is specified, the cause of unforeseen errors cannot be investigated. Use caution when applying this environment variable.

(61) PDPRMTRC={YES|NO|IN|OUT|INOUT}

This environment variable specifies whether parameter information and retrieval data are to be output in the SQL trace information. For details about the output contents, see *11.1.1 SQL tracing*.

`YES`

Output input parameter information in the SQL trace. If `YES` is specified, retrieval data information and the input parameters are output.

`NO`

Do not output parameter information in the SQL trace.

`IN`

Output the input parameter information in the SQL trace. This also applies to the `IN` and `INOUT`# parameters of the `CALL` statement.

`OUT`

Output the output parameter information and retrieval data information in the SQL trace. This also applies to the `OUT` and `INOUT`# parameters of the `CALL` statement.

`INOUT`

Output the input parameter information, the output parameter information, and the retrieval data information in the SQL trace. The INOUT parameter# of the CALL statement is output twice.

#: Information on the INOUT parameter of the CALL statement is used only as output data.

(62) PDPRMTRCSIZE=maximum-data-length-of-parameter-information-output-to-SQL-trace

~ <unsigned integer> ((4-32008)) <<256>> (bytes)

This environment variable specifies the maximum data length of the parameter information and retrieval data to be output in the SQL trace. For variable-length character string-type, BLOB-type, and BINARY-type data, the area of the character string length is included in the data length.

This environment variable is valid only when a value other than PDPRMTRC=NO is specified.

Increasing the specified value of this environment variable increases the amount of information that is output. Therefore, the size of the SQL trace file (PDSQLTRACE specification value) must also be increased.

(63) PDTRCMODE={ERR|NONE}

This environment variable specifies whether troubleshooting information (pderr*.trc information) other than SQL trace information is to be output.

ERR: Output pderr*.trc information.

NONE: Do not output pderr*.trc information.

(64) PDUAPREPLVL={[s][u][p][r]|a}

This environment variable specifies output information for the UAP statistical report. A file to which a UAP statistical report is output is called a UAP statistical report file. This environment variable becomes effective when PDCLTPATH is specified.

If this environment variable is omitted, only SQL trace information is output.

For details about UAP statistical reports, see *11.1.4 UAP statistical report facility*.

s: SQL unit information is output. SQL trace information is also output.

u: UAP unit information is output.

p: Access path information is output.

r: SQL runtime interim results are output.

a: All information is output.

s, u, p, and r can be specified in different combinations (such as su, sr, or upr). Specifying supr is the same as specifying a. If u, p, r, up, ur, pr, or upr is specified, SQL trace information is not output.

Notes

1. If access path information or SQL runtime interim results are to be collected, the server load might increase because SQL objects are re-created even if they already exist in the buffer.
2. Information for a UAP is not output in the following cases:
   - The program uses an API that is X/Open-compliant under OLTP.
   - The UAP terminates without executing `DISCONNECT`.
3. If the size of the access path information and SQL runtime interim results exceeds 1 gigabyte, the information is not output.
4. The value `0` is displayed in the time display (for example, SQL execution time, load wait time, or CPU time) if the value is too small to be retrieved by a system call of the operating system.
5. With a HiRDB/Parallel Server, privilege check processing by the connected dictionary server is not included in the UAP unit information.
6. If you specify output of access path information or SQL runtime interim results and also specify the inter-process memory communication facility (`PDIPC=MEMORY` in the client environment definitions), the `PDIPC` specification is ignored and `PDIPC=DEFAULT` is assumed.

(65) PDREPPATH=storage-directory-for-UAP-statistical-report-files

~ <path name> ((up to 256 bytes))

This environment variable specifies the directory in which UAP statistical report files are to be created if the files are to be created in a different directory from the directory specified by `PDCLTPATH`. This environment variable is effective only when `PDUAPREPLVL` is specified.

Information is output to the UAP statistical report file each time the UAP is connected or disconnected. The file name is formed from the connection time (*HH:MM:SS:mmm*) and the connection number (*XXX*). Examples are `pd`*HHMMSSmmm_XXX*`_1.trc` and `pd`*HHMMSSmmm_XXX*`_2.trc`.

(66) PDTRCPATH=storage-directory-for-dynamic-SQL-trace-files

~ <path name> ((up to 256 bytes))

This environment variable specifies the storage directory for dynamic SQL trace files that the HiRDB client creates. This environment variable must be specified when dynamic SQL trace files are collected with the trace acquisition command (`pdtrcmgr`).

When the directory specified here is specified in the `pdtrcmgr` command, an SQL trace file is created in the specified directory from the next connection. For details about pdtrcmgr, see *11.1.6 SQL trace dynamic acquisition facility*.

(67) PDSQLTRCOPENMODE={CNCT|SQL}

This environment variable specifies the open mode for SQL trace files when PDREPPATH is specified.

CNCT

Opens and closes the SQL trace file in CONNECT and DISCONNECT units, and outputs trace information. When CNCT is specified instead of SQL in PDSQLTRCOPENMODE, the SQL trace output time can be shortened because the overhead is reduced.

When CNCT is specified, the system continues to write information as long as the SQL trace file is open. Therefore, some SQL trace information may be discarded if DISCONNECT cannot be executed properly.

SQL

Opens and closes the SQL trace file in operation units (SQL units), and outputs trace information.

(68) PDSQLTEXTSIZE=SQL-statement-size

~ <unsigned integer> ((4096-2000000)) <<4096>> (bytes)

This environment variable specifies the size of the SQL statement to be output to the SQL trace.

If this environment variable is omitted during access path acquisition, 2000000 is assumed instead of 4096.

(69) PDSQLEXECTIME={YES|NO}

This environment variable specifies whether the SQL runtime is to be output to the SQL trace.

YES

Output the SQL runtime.

The unit for the SQL runtime that is output is microseconds. Normally runtime values over 24 hours are not output in the SQL trace.

NO

Do not output the SQL runtime.

(70) PDRCTRACE=reconnect-trace-file-size

~ <unsigned integer> ((0, 4096-2000000000)) (bytes)

This environment variable specifies the size of the output file for UAP reconnect trace information.

If 0 is specified, the maximum file size is assumed, and UAP reconnect trace information that exceeds the maximum size is not output. UAP reconnect trace information also is not output when this environment variable is omitted.

If a value from 4,096 to 2,000,000,000 is specified, the specified value becomes the file size, and the output destination switches when the file size exceeds the specified value.

The reconnect trace is output to the directory specified in PDCLTPATH. If PDCLTPATH is not specified, the reconnect trace is output in the current directory when the UAP is executed (current directory during J2EE server execution if the UAP is executed from Cosminexus). For details about the reconnect trace, see *11.1.7 Reconnect trace facility*.

(71) PDWRTLNPATH=storage-directory-for-files-to-which-WRITE-LINE-statement-value-expression-values-are-output

~ <path name> ((up to 256 bytes))

This environment variable specifies the storage directory for files to which value expression values of WRITE LINE statements are to be output. For details about the WRITE LINE statement, see the manual *HiRDB Version 9 SQL Reference*.

If PDWRTLNPATH is omitted, the directory specified in PDCLTPATH is assumed.

Two files are created in the specified directory (or the directory specified in PDCLTPATH if PDWRTLNPATH is omitted). The files that are created differ depending on whether or not an X/Open-compliant API (TX_function) is used. The names of the created files are shown as follows.

If TX_function is not used

pdwrtln1.trc and pdwrtln2.trc

If TX_function is used

pdwrtln*xxxxx*-1.trc and pdwrtln*xxxxx*-2.trc

xxxxx: Process ID when the UAP is executed

(72) PDWRTLNFILSZ=maximum-size-of-output-files-for-WRITE-LINE-statement-value-expression-values

~ <unsigned integer> ((0, 4096-2000000000)) (bytes)

This environment variable specifies the maximum size of the files to which value expression values of WRITE LINE statements are to be output.

If 0 is specified, the maximum file size is the maximum file size that the OS can manage. If the maximum size is exceeded, the value expression values of WRITE LINE statements are not output. Value expression values of WRITE LINE statements also are not output if this environment variable is omitted.

If a value from 4,096 to 2,000,000,000 is specified, the specified value becomes the maximum file size, and the output destination switches when the file size exceeds the specified value.

Notes

1. If both `PDWRTLNFILSZ` and `PDIPC=MEMORY` are specified, the `PDIPC` specification becomes invalid.
2. The files are output to the directory specified in PDWRTLNPATH.
3. If a file becomes full when values are being output, values are output to the other file. When this happens, the information already stored in the switchover-destination file is deleted, and new information is written to that file. Therefore, if the file contains required information, save that information before switchover occurs. To find out which file is currently being used, use the following method. The file that has the more recent update date is the current file.
   - In UNIX: Execute the `ls -l` command of the OS.
   - In Windows: Execute the `DIR` command from the command prompt, or use Windows Explorer to check the files.

(73) PDWRTLNCOMSZ=total-size-of-WRITE-LINE-statement-value-expression-values

~ <unsigned integer> ((1024-131072)) <<1024>> (bytes)

This environment variable specifies the total size of the value expression values in `WRITE LINE` statements.

If the total size of the value expression values in `WRITE LINE` statements exceeds the value specified in this environment variable, the excess information is ignored. In this case, `**PDWRTLNCOMSZover**` is output in the following line.

(74) PDUAPEXERLOGUSE={YES|NO}

This environment variable specifies whether the facility for output of extended SQL error information is to be used.

For details about the facility for output of extended SQL error information, see *11.1.3 Facility for output of extended SQL error information*.

`YES`

Use the facility for output of extended SQL error information.

`NO`

Do not use the facility for output of extended SQL error information.

Relationship to the system definition

When this environment variable is omitted, the specification value of the pd_uap_exerror_log_use operand in the system definition is assumed.

(75) PDUAPEXERLOGPRMSZ=maximum-data-length-of-parameter-information

~ <unsigned integer> ((0-32008)) (bytes)

This environment variable specifies the maximum data length for the parameter information to be output to the client error log file and SQL error report file when the facility for output of extended SQL error information is used. Parameter information is output when a value of 1 or higher is specified, but parameter information is not output when 0 is specified.

Relationship to the system definition

When this environment variable is omitted, the specification value of the pd_uap_exerror_log_param_size operand in the system definition is assumed.

Notes

1. For variable-length character string-type, BLOB-type, and BINARY-type data, the area of the character length is included in the specification value of this environment variable.
2. If the data length of the parameter information to be output exceeds the specification value of this environment variable, the excess portion of the information is truncated.

(76) PDARYERRPOS={YES|NO}

This environment variable specifies whether the value indicating an erroneous array element is to be set in the SQL Communications Area when update processing using arrays results in an error.

YES

Set the value indicating an erroneous array element in the SQL Communications Area.

NO

Do not set the value indicating an erroneous array element in the SQL Communications Area.

The SQL Communications Area used to set the value indicating an erroneous array element is as follows:

- In C language, the location is set in SQLERRD2.
- In COBOL, the location is set in SQLERRD(3).

An erroneous array element is indicated by setting a value from -1 to -*n*, where -1 indicates the first element of the array and `-`*n* represents the last element. For details about the SQL Communications Area, see Appendix *A. SQL Communications Area*.

(77) PDDNDPTRACE=method-trace-file-size

~ <unsigned integer> ((0, 65536 to 2147483647)) (bytes)

This environment variable specifies the file size for method traces that are output by the HiRDB.NET data provider supporting ADO.NET 2.0.

When this environment variable is specified, the system outputs the method traces to the directory specified in `PDCLTPATH`.

If you specify `0`, as much method trace information will be output as there is space on the disk. If a value in the range from 65536 to 2147483647 is specified, the specified value is treated as the maximum size. When this maximum size is reached, the trace output destination is changed. If this environment variable is omitted, no method trace information is output. For details about method traces, see *16.10 Troubleshooting function of HiRDB.NET Data Provider*.

(78) PDVWOPTMODE ={0|1|2}

This environment variable specifies whether access path information is to be acquired for the access path display utility.

The access path information file is created under the SQL information directory (`%PDDIR%\spool\pdsqldump`) of the unit containing the single server or the front-end server to which the UAP is connected.

For details about the access path display utility, see the *HiRDB Version 9 Command Reference* manual.

`0`

Do not collect access path information.

`1`

Collect access path information and output the information to the access path information file. No information is output for SQL statements that have SQL objects in the buffer.

`2`

Collect access path information and output the information to the access path information file. For SQL statements that have SQL objects in the buffer, the SQL objects are re-created and the information is output.

Notes

1. Specify `PDTAAPINFPATH` to acquire access path information for HiRDB SQL Tuning Advisor.

2. Note that when 1 is specified, no information is output for SQL statements that have SQL objects in the buffer. If you want the information output to include information about SQLs that have SQL objects in the buffer, specify 2.

3. If 2 is specified, the server load increases compared to when 1 is specified because SQL objects are also re-created for SQL statements that have SQL objects in the buffer.

4. If the total of *%PDDIR%-path-length* + *authorization-identifier-length* + *UAP-name-length* is larger than 220 characters when the Windows-edition HiRDB is used, creation of the access path information file may fail. If this happens, use the UAP statistical report facility and get access path information. For details about the UAP statistical report facility, see *11.1.4 UAP statistical report facility*.

5. The following table shows the relationships between the SQL types and the PDVWOPTMODE specification values.

SQL type	Condition	PDVWOPTMODE specification value		
		0	1	2
Static SQL	SQL objects are not found in the buffer.	--	Y	Y
	SQL objects are found in the buffer.	--	--	Y
dynamic SQL	SQL objects are not found in the buffer.	--	Y	Y
	SQL objects are found in the buffer.	--	--	Y
Routine definition	None	--	Y	Y
CALL statement	Index information for procedure SQL objects is invalid as a result of index addition or deletion.	--	Y	Y
	Other condition	--	--	--

Y: Access path information is output.

--: Access path information is not output.

(79) PDTAAPINFPATH=access-path-information-file-output-directory-name

~<path name>

This environment variable specifies the output destination directory when an access path information file for HiRDB SQL Tuning Advisor is output. If an output processing error occurs even with this environment variable specified, because the output destination directory does not exist or because there is no write privilege for the specified directory, the access path information is not output. Note that even when an output processing error occurs, there is no error in the executing SQL.

Notes

- This environment variable is ignored when the dynamic browsing function of HiRDB SQL Tuning Advisor is used.
- The inter-process memory communication facility cannot be used when this environment variable is specified. Even though you specify `MEMORY` for the `PDIPC` operand in the client environment definition, operation is the same as when `DEFAULT` is specified.

(80) PDTAAPINFMODE={0|1}

This environment variable specifies the file name format of the access path information files that are output for HiRDB SQL Tuning Advisor.

`0`:

The file names are `pdtaapinf1` and `pdtaapinf2`.

`1`:

The file names are in the format `pdtaapinf`*HHMMSSmmm*_*XXX*`_1` and `pdtaapinf`*HHMMSSmmm*_*XXX*`_2`.

HHMMSSmmm:

Connection time (same as the connection start time of the applicable `CONNECT` output in an SQL trace)

XXXXXXXXXX:

Connection sequence number (maximum of 10 digits)

(81) PDTAAPINFSIZE=access-path-information-file-size

~<unsigned integer>((100000 - 2000000000)) (409600) (bytes)

This environment variable specifies the file size of an access path information file that is output for HiRDB SQL Tuning Advisor. When the file size specified here is reached in the current access path information file, the output destination is switched to the other file. After that, the two files are used alternately by repeating this switching.

(82) PDSTJTRNOUT={YES|NO}

This environment variable specifies whether UAP statistical information is to be output to a statistical log file for each transaction.

`YES`

Output UAP statistical information to a statistical log file for each transaction.

`NO`

Output UAP statistical information to a statistical log file for each connection.

To specify the start of UAP statistical information output, use the `pdstbegin` operand of the system definition or the `pdstbegin` command. For details about the `pdstbegin` operand, see the manual *HiRDB Version 9 System Definition*. For details about the `pdstbegin` command, see the manual *HiRDB Version 9 Command Reference*.

If this environment variable is omitted when the UAP is operating in an OLTP environment, UAP statistical information is output to a statistical log file for each transaction. If this environment variable is omitted when the UAP is operating in another environment, UAP statistical information is output to a statistical log file for each connection.

(83) PDLOCKLIMIT=maximum-locked-resource-request-count-per-user

~ <unsigned integer> ((0-200000000)) <<0>> (count)

This environment variable specifies the maximum number of lock requests (that is, the maximum number of locked resource requests) that a UAP can issue to one server.

If `0` is specified or this environment variable is omitted, the HiRDB system does not check the maximum number of lock requests. In this case, the maximum possible number of lock requests is issued.

If a locked resources shortage occurs, an SQL error results.

Estimation method

There are two methods:

- One method determines the maximum number of locked resources from the number of locked resources used by one UAP.

  The number of locked resources depends on the SQL. Estimate the number of locked resources depending on the lock processing to determine the value to be specified in this operand. For details about how to estimate the locked resource count, see the manual *HiRDB Version 9 System Definition*. For details about lock processing, see *3.4 Locking*.

- The other method determines the maximum number of locked resources from the number of locked resources that can be used by one UAP among all the locked resources available to the HiRDB server that connects the UAP.

  For details about how to estimate the number of locked resources, see the manual *HiRDB Version 9 System Definition*.

(84) PDDLKPRIO={96|64|32}

This environment variable specifies the deadlock priority value of the UAP and becomes effective when `Y` is specified in the `pd_deadlock_priority_use` operand of the system definition.

If a deadlock occurs between two programs, the program with the smaller value specified in this environment variable is processed at a higher priority. The program with the larger value is terminated with an error, and that program is rolled back.

If a deadlock occurs between two programs that have the same deadlock priority value, the one with the transaction that started earlier is processed first. The following table lists the deadlock priority values:

PDDLKPRIO specification			Deadlock priority value
96			96
64			64
32			32
Omitted	When the X/Open XA interface is used		96
	When the X/Open XA interface is not used		64
--	Utility		64
	Operation command	`pddbchg`, `pdhold` (`-b` and `-s`), `pdorbegin`, or `pdorend`	Specification value of the `pd_command_deadlock_priority` operand in the system definition
		Other operation command	64

--: Not applicable

(85) PDLOCKSKIP={YES|NO}

This environment variable specifies whether an unlocked conditional search can be performed.

`YES`

Enables an unlocked conditional search.

`NO`

Disables an unlocked conditional search.

When `YES` is specified in this environment variable, the conditional search of a retrieval process (including retrieval for `DELETE` and `UPDATE`) is performed without locking all items. For details about unlocked conditional search, see *3.4.5 Unlocked conditional search*.

(86) PDFORUPDATEEXLOCK={YES|NO}

This environment variable specifies whether `WITH EXCLUSIVE LOCK` is to be applied to the lock option of SQL statements in which the `FOR UPDATE` clause is specified (or assumed) in the UAP. If `WITH EXCLUSIVE LOCK` is applied, the value specified in

PDISLLVL in the client environment definition is ignored.

YES

Apply WITH EXCLUSIVE LOCK to the lock option of SQL statements in which the FOR UPDATE clause is specified.

NO

Apply the PDISLLVL specification value to the lock option of SQL statements in which the FOR UPDATE clause is specified.

If PDFORUPDATEEXLOCK is specified for an SQL statement in a routine, the specification becomes invalid. To apply WITH EXCLUSIVE LOCK to an SQL statement specifying the FOR UPDATE clause in a routine, specify WITH EXCLUSIVE LOCK as an SQL compile option when defining the routine.

(87) PDISLLVL=data-guarantee-level

~ <unsigned integer> ((0-2)) <<2>>

This environment variable specifies the data guarantee level of an SQL statement. The data guarantee level is the point in a transaction up to which data is to be guaranteed. This environment variable has the same function as the WITHOUT LOCK option that is specified in the SELECT statement.

This environment variable enables batch determination of all lock options for the SQL statements in a UAP. Note that the data guarantee level specified in the lock option of an SQL statement takes precedence over this operand.

For details about the data guarantee level, see *4.6 Data guarantee levels*.

0

If another user is updating data, users are allowed to reference the same data without having to wait for completion of the update processing. This specification can improve the processing concurrency level. However, if the same row of data is retrieved twice within the same transaction, the same data might not be received. For example, if a stock table is retrieved with SELECT * FROM STOCK, the user can retrieve the desired data without having to wait for lock release, even when another user is updating the stock table. This corresponds to the SELECT statement with WITHOUT LOCK NOWAIT.

1 is always assumed for a cursor declaration used with update processing, even if 0 is specified.

1

If a user is retrieving data, other users are not allowed to update that data until retrieval processing is completed. Other users are allowed to reference or update that data when the retrieval terminates, even if the transaction has not terminated. This specification therefore improves the apparent concurrent execution property.

However, if the same row is retrieved twice in the same transaction, the same data may not be retrieved. For example, if data is being retrieved from a stock table with `SELECT * FROM STOCK`, other users are allowed to update or reference the stock table after the retrieval ends, without having to wait for the transaction to terminate. This corresponds to the `SELECT` statement with `WITHOUT LOCK WAIT`.

2

All other users are prohibited from updating the data being retrieved until the retrieval transaction terminates. For example, if a stock table is retrieved with `SELECT * FROM STOCK`, the contents of the stock table are guaranteed until the transaction terminates. This corresponds to the `SELECT` statement with `WITH SHARE LOCK`.

For a cursor declaration that accompanies an update, `WITH EXCLUSIVE LOCK` is assumed.

Notes

1. The data guarantee level of SQL statements in a stored procedure is determined by the specifications for `CREATE PROCEDURE`, `CREATE TYPE`, `ALTER PROCEDURE`, and `ALTER ROUTINE`. Therefore, when a procedure is executed, the data guarantee level is not affected by this environment variable.
2. If this environment variable is omitted along with the lock option in an SQL statement, the `WITH SHARE LOCK` option is assumed for the SQL statement. For details about lock options, see the *HiRDB Version 9 SQL Reference* manual.

(88) PDSQLOPTLVL=SQL-optimization-option[,SQL-optimization-option]...

~ <identifier or unsigned integer>

This environment variable specifies optimization methods for determining the most efficient access path by taking the database status into consideration.

Although SQL optimization options can be specified either with identifiers (character strings) or numbers, specifying the options with identifiers is usually recommended.

Specifying the SQL optimization methods with identifiers
`PDSQLOPTLVL="`*identifier*`"[,"`*identifier*`"]...`

Examples

- Applying *prioritized nest-loop-join* and *rapid grouping processing:*

```
PDSQLOPTLVL="PRIOR_NEST_JOIN","RAPID_GROUPING"
```

- Applying no optimization method:

```
PDSQLOPTLVL="NONE"
```

Rules

1. Specify at least one identifier.
2. When specifying two or more identifiers, separate them with commas.
3. For details about the information (optimization methods) that can be specified with identifiers, see *Specification values for the SQL optimization option*.
4. If no optimization is to be applied, specify `NONE` as the identifier. However, if another identifier is specified together with `NONE`, then `NONE` becomes invalid.
5. The identifiers can be specified with uppercase and lowercase characters.
6. Even if the same identifier is specified more than once, HiRDB recognizes only one specification. However, try not to specify the same identifier more than once.
7. The character string specified for `"`*identifier*`"[,"`*identifier*`"]...` can have up to 575 bytes.

Specifying the SQL optimization methods with numbers

`PDSQLOPTLVL=`*unsigned-integer*`[,`*unsigned-integer*`]...`

Examples

- Applying *making multiple SQL objects*, *suppressing use of AND multiple indexes*, and *forcing use of multiple indexes*

  Specification when unsigned integers are separated by commas:

  ```
  PDSQLOPTLVL=4,10,16
  ```

  Specification when the sum of the unsigned integers is specified:

  ```
  PDSQLOPTLVL=30
  ```

- Specification when `14` (`4+10`) is already specified and `16` is added:

  ```
  PDSQLOPTLVL=14,16
  ```

- Applying no optimization method:

  ```
  PDSQLOPTLVL=0
  ```

Rules

1. When HiRDB is updated from a version before 06-00 to version 06-00 or later, the total value specification of the earlier version remains

effective. If the optimization options do not need to be changed after HiRDB is updated to version 06-00 or later, the specification value of this operand does not need to be changed.

2. Specify at least one unsigned integer.
3. When specifying two or more unsigned integers, separate them with commas.
4. For details about the information (optimization methods) that can be specified with unsigned integers, see *Specification values for the SQL optimization option*.
5. If no optimization is to be applied, specify 0 as the identifier. However, if another identifier is specified together with 0, then 0 becomes invalid.
6. Even if the same unsigned integer is specified more than once, HiRDB recognizes only one specification. However, try not to specify the same unsigned integer more than once.
7. Multiple optimization methods can also be specified by specifying the sum of the unsigned integers. However, do not add the same optimization method value more than once. (Otherwise, the specified result may be interpreted as an unintended optimization methods.)
8. If multiple optimization method values are added together and specified, it becomes difficult to determine which optimization methods are being specified. Hitachi therefore recommends that you separate the values with commas. If several optimization method values have already been added and specified, and a new optimization method becomes necessary, you can separate the new value with a comma and specify it after the previous specification.
9. The character string specified for "*unsigned-integer*"[,"*unsigned-integer*"]... can have up to 575 bytes.

Relationship to the system definition

1. When this environment variable is omitted, the value specified in the `pd_optimize_level` operand of the system definition is assumed. For details about the `pd_optimize_level` operand, see the manual *HiRDB Version 9 System Definition*.
2. If the `pd_floatable_bes` or `pd_non_floatable_bes` operand is specified in the system definitions, the specifications for increasing the target floatable servers (back-end servers for fetching data) and limiting the target floatable servers (back-end servers for fetching data) specifications become invalid.
3. If KEY is specified (for index key value locking) in the

pd_indexlock_mode operand of the system definitions, the specification for suppressing creation of update-SQL work tables becomes invalid.

4. We recommend that you specify all the following identifiers when you omit the pd_optimize_level operand in the system definition:

   "PRIOR_NEST_JOIN","PRIOR_OR_INDEXES",

   "DETER_AND_INDEXES","RAPID_GROUPING",

   "DETER_WORK_TABLE_FOR_UPDATE",

   "APPLY_ENHANCED_KEY_COND",

   "MOVE_UP_DERIVED_COND"

Relationship with SQL

The SQL optimization option for an SQL statement in a stored procedure is determined by the specifications for CREATE PROCEDURE, CREATE TYPE, ALTER PROCEDURE, or ALTER ROUTINE, and is not affected by the PDSQLOPTLVL specification.

If an SQL optimization specification is specified in an SQL statement, the SQL optimization specification has priority over the SQL optimization option. For details about SQL optimization specifications, see the manual *HiRDB Version 9 SQL Reference*.

Specification values for the SQL optimization option

The following table shows the values that can be specified for the SQL optimization option.

Table 6-32: Specification values of the SQL optimization option

Number	Optimization method	Specification value	
		Identifier	Unsigned integer
1	Forced nest-loop-join	"FORCE_NEST_JOIN"	*4*
2	Making multiple SQL objects	"SELECT_APSL"	*10*
3	Increasing the target floatable servers (back-end servers for fetching data)[#1, #2]	"FLTS_INC_DATA_BES"	*16*
4	Prioritized nest-loop-join	"PRIOR_NEST_JOIN"	32
5	Increasing the number of floatable server candidates[#2]	"FLTS_MAX_NUMBER"	64

Number	Optimization method	Specification value	
		Identifier	Unsigned integer
6	Priority of `OR` multiple index use	`"PRIOR_OR_INDEXES"`	`128`
7	Group processing, `ORDER BY` processing, and `DISTINCT` set function processing at the local back-end server[#2]	`"SORT_DATA_BES"`	`256`
8	Suppressing use of `AND` multiple indexes	`"DETER_AND_INDEXES"`	`512`
9	Rapid grouping processing	`"RAPID_GROUPING"`	`1024`
10	Limiting the target floatable servers (back-end servers for fetching data)[#1, #2]	`"FLTS_ONLY_DATA_BES"`	`2048`
11	Separating data collecting servers[#1, #2]	`"FLTS_SEPARATE_COLLECT_SVR"`	`2064`
12	Suppressing index use (forced table scan)	`"FORCE_TABLE_SCAN"`	`4096`
13	Forcing use of multiple indexes	`"FORCE_PLURAL_INDEXES"`	`32768`
14	Suppressing creation of update-SQL work tables	`"DETER_WORK_TABLE_FOR_UPDATE"`	`131072`
15	Deriving high-speed search conditions	`"DERIVATIVE_COND"`	`262144`
16	Applying key conditions that include scalar operations	`"APPLY_ENHANCED_KEY_COND"`	`524288`
17	Facility for batch acquisition from functions provided by plug-ins	`"PICKUP_MULTIPLE_ROWS_PLUGIN"`	`1048576`
18	Facility for moving search conditions into derived table	`"MOVE_UP_DERIVED_COND"`	`2097152`

#1: If increasing the target floatable servers (back-end servers for fetching data) and limiting the target floatable servers (back-end servers for fetching data) are both specified, neither optimization method becomes effective. Instead, the servers operate as separating data collecting servers.

#2: When a HiRDB/Single Server is used, this option becomes invalid, even if specified.

Recommended specification values

The recommended specification values are indicated with item numbers in the examples and the following table. These numbers correspond to the numbers in the *Number* column of *Table 6-32*.

- HiRDB/Single Server

  Specify item numbers 4, 6, 8, 9,14,16 and 18. An example of how these numbers are specified with identifiers is shown as follows.

```
PDSQLOPTLVL="PRIOR_NEST_JOIN",
            "PRIOR_OR_INDEXES",
            "DETER_AND_INDEXES",
            "RAPID_GROUPING",
            "DETER_WORK_TABLE_FOR_UPDATE",
            "APPLY_ENHANCED_KEY_COND",
            "MOVE_UP_DERIVED_COND"
```

- HiRDB/Parallel Server

  The following table shows the recommended specification values for the SQL optimization option.

Table 6-33: Recommended specification values for the SQL optimization option (for HiRDB/Parallel Server)

Condition		Specification value

Use as many back-end servers as feasible for SQL processing so that the individual SQL statements can be processed quickly	To process SQL statements involving mass-data searches quickly	Specify item numbers 3 to 9, 14 and 16.
		Identifier specification example:
		`PDSQLOPTLVL="FLTS_INC_DATA_BES",`
		`"PRIOR_NEST_JOIN",`
		`"FLTS_MAX_NUMBER",`
		`"PRIOR_OR_INDEXES",`
		`"SORT_DATA_BES",`
		`"DETER_AND_INDEXES",`
		`"RAPID_GROUPING",`
		`"DETER_WORK_TABLE_FOR_UPDATE"`
		`"APPLY_ENHANCED_KEY_COND"`

	To process searches with numerous results (several tens of data items) quickly	Specify item numbers 3, 4, 6 to 9, 14 and 16.
		Identifier specification example:
		`PDSQLOPTLVL="FLTS_INC_DATA_BES",`
		`"PRIOR_NEST_JOIN",`
		`"PRIOR_OR_INDEXES",`
		`"SORT_DATA_BES",`
		`"DETER_AND_INDEXES",`
		`"RAPID_GROUPING",`
		`"DETER_WORK_TABLE_FOR_UPDATE"`
		`"APPLY_ENHANCED_KEY_COND"`

Condition		Specification value

To separate back-end servers for each job	To process SQL statements involving mass-data searches quickly	Specify item numbers 4 to 10, 14 and 16.
		Identifier specification example:
		`PDSQLOPTLVL="PRIOR_NEST_JOIN",`
		`"FLTS_MAX_NUMBER",`
		`"PRIOR_OR_INDEXES",`
		`"SORT_DATA_BES",`
		`"DETER_AND_INDEXES",`
		`"RAPID_GROUPING",`
		`"FLTS_ONLY_DATA_BES",`
		`"DETER_WORK_TABLE_FOR_UPDATE"`
		`"APPLY_ENHANCED_KEY_COND"`

	To process searches with numerous results (several tens of data items) quickly	Specify item numbers 4, 6 to 10, 14 and 16.
		Identifier specification example:
		`PDSQLOPTLVL="PRIOR_NEST_JOIN",`
		`"PRIOR_OR_INDEXES",`
		`"SORT_DATA_BES",`
		`"DETER_AND_INDEXES",`
		`"RAPID_GROUPING",`
		`"FLTS_ONLY_DATA_BES",`
		`"DETER_WORK_TABLE_FOR_UPDATE"`
		`"APPLY_ENHANCED_KEY_COND"`

Other conditions		Specify item numbers 4, 6 to 9, 14 and 16.
		Identifier specification example:
		`PDSQLOPTLVL="PRIOR_NEST_JOIN",`
		`"PRIOR_OR_INDEXES",`
		`"SORT_DATA_BES",`
		`"DETER_AND_INDEXES",`
		`"RAPID_GROUPING",`
		`"DETER_WORK_TABLE_FOR_UPDATE"`
		`"APPLY_ENHANCED_KEY_COND"`

Explanation of optimization methods

1. Forced nest-loop-join

   If indexes are defined in the columns of the join condition, only nest-loop-join is used in join processing. For details about the join processing method for nest-loop-join, see *4.5.6 Join methods*.

   However, if one of the following conditions applies, a method other than nest-loop-join may be used in join processing:

   - An entity (for example, a scalar operation) other than a column is specified in the join condition.
   - The join condition is not a `=` predicate.
   - The column in the join condition is not the first configuration column of the index. Also, if the column in the join condition is the n-th configuration column of the index, a `=` predicate or a restriction condition of the `IS NULL`

predicate is not specified for any of the preceding configuration columns (first configuration column to *n* -1 -th configuration column).

- A join condition is not specified in the ON search condition for an outer join.
- In the join condition, a plug-in presentation function that executes a search using the indexes of two tables to be joined, or a structured repetition predicate, is specified for the two tables.
- A HiRDB/Parallel Server is used, and a partitioned column of the inner table is not specified in the join condition for an outer join that uses a partitioned table as the inner table.
- A HiRDB/Parallel Server is used, and the outer join uses a flexible hashed partitioned table as the inner table.

Notes

1. If a joined table is to be processed by nest-loop-join, the table that was specified as the outer table in the SQL is used as the outer table.

2. If an index is defined in only one of the columns of the join condition and the join is to processed with nest-loop-join, the table with the defined index becomes the inner table.

3. Except when a joined table is involved, if a join where indexes are defined in the columns on both sides of the join condition is processed by nest-loop-join, HiRDB judges and determines the outer and inner tables of the nest-loop-join. However, if a view table or WITH clause query name is not specified in the FROM clause, HiRDB determines the outer and inner tables according to the following rules:

- If partitioned tables of a HiRDB/Parallel Server are to be joined, and all partitioned columns of one table but not all partitioned columns of the other table are specified in the join condition, the table for which all partitioned columns are specified in the join condition becomes the inner table.

- If (a) previously does not apply, the first table specified in the FROM clause becomes the outer table.

4. If *forced nest-loop-join* is applied in the HiRDB/Parallel Server and mass data is to be joined, partition the tables with joined columns as much as possible.

2. Making multiple SQL objects

   Multiple SQL objects are created in advance, and the optimum SQL object is selected during execution, based on the value of an embedded variable or the ? parameters.

3. Increasing the target floatable servers (back-end servers for fetching data)

   Normally, back-end servers that are not used for fetching data are used as floating servers. With this optimization method, back-end servers that are used for fetching data can also be used as floating servers. However, the HiRDB system calculates the number of back-end servers that can be used as floating servers, and not all back-end servers end up being used as floating servers. To use all back-end servers, also specify the specification for increasing the number of floatable server candidates.

For details about how to allocate floatable servers, see *4.5.4 Allocating floatable servers (HiRDB/Parallel Server only)*.

This specification is applicable only to a HiRDB/Parallel Server.

4. Prioritized nest-loop-join

If indexes are defined in the columns of the join condition, nest-loop-join is used with priority in join processing. For details about the join processing method for nest-loop-join, see Section *4.5.6 Join methods*.

This optimization method is different from *1. Forced nest-loop-join*. Forced nest-loop-join always executes nest-loop-join if indexes are defined in the join condition, even if there is no narrowing condition (except when restrictions apply). On the other hand, while prioritized nest loop join always executes nest-loop-join if a narrowing condition is specified, HiRDB determines the join method if there is no narrowing condition. However, if one of the following conditions applies, a method other than nest-loop-join may be used in join processing, even if a narrowing condition is specified:

- An entity (for example, a scalar operation) other than a column is specified in the join condition.
- The join condition is not a = predicate.
- The column in the join condition is not the first configuration column of the index. Also, if the column in the join condition is the n-th configuration column of the index, a = predicate or a restriction condition of the `IS NULL` predicate is not specified for any of the preceding configuration columns (first configuration column to *n* `-1` -th configuration column).
- A join condition is not specified in the `ON` search condition for an outer join.
- In the join condition, a plug-in presentation function that executes a search using the indexes of two tables to be joined or a structured repetition predicate is specified for the two tables.
- A HiRDB/Parallel Server is used, and a partitioned column of the inner table is not specified in the join condition for an outer join that uses a partitioned table as the inner table.
- A HiRDB/Parallel Server is used, and the outer join uses a flexible hashed partitioned table as the inner table.

- The optimizing information collection utility (`pdgetcst`) is being executed.
- The narrowing condition is a search condition that includes only a `CHAR`, `VARCHAR`, `MCHAR`, or `MVARCHAR` column that has a definition length of at least 256 bytes; an `NCHAR` or `NVARCHAR` column that has a definition length of at least 128 characters; or a BLOB column.
- The narrowing condition is a search condition that includes only a `NOT` or `OR` operator.

Notes

1. If a joined table is to be processed by nest-loop-join, the table that was specified as the outer table in the SQL is used as the outer table.

2. If an index is defined in only one of the columns of the join condition and the join is processed with nest-loop-join, the table with the defined index becomes the inner table.

3. Except when joined tables are involved, if an index is defined in both columns of the join condition and the join is to be processed with nest-loop-join, HiRDB determines which table becomes the outer table and which becomes the inner table in the nest-loop-join. However, if a view table or `WITH` clause query name is not specified in the `FROM` clause, and only a join clause is specified in the search conditions, HiRDB determines the outer and inner tables according to the following rules:

- If partitioned tables of HiRDB/Parallel Server are being joined, specify all partitioned columns of one table in the join conditions. If the partitioned columns of the other table being joined contains columns that were not specified in the search conditions, the table with the partitioned columns that were all specified in the join conditions becomes the inner table.

- If the preceding rule does not apply, the first table specified in the `FROM` clause becomes the outer table.

4. If *1. Forced nest-loop-join* is also specified, this optimization option becomes invalid.

5. Increasing the number of floatable server candidates

Normally, the HiRDB system calculates and allocates the number of floating servers that are necessary from the floatable servers that can be used. When this optimization method is applied, all usable floating servers are used, except for back-end servers that are used for fetching data.

If you wish to include the back-end servers used for fetching data for use as floatable servers, also specify the value for increasing the target floatable servers (back-end servers for fetching data).

For details about how to allocate floatable servers, see *4.5.4 Allocating floatable* *servers (HiRDB/Parallel Server only)*. This specification is applicable only to a HiRDB/Parallel Server.

6. Priority of OR multiple index use

Specify this method to give application priority to the method that uses OR multiple indexes in searching. The OR multiple index use method is used when multiple conditions are combined with OR in the search condition. This method uses an index to search each condition and evaluates the search condition by taking a sum set of the search results.

When `A OR B OR C ... OR Z` is specified in the WHERE clause or OR search condition and the data is narrowed by using = for all conditions combined with OR, a high-speed search can be realized by applying *priority of OR multiple index use*.

Even when the value for priority of OR multiple index use is not specified, HiRDB applies OR multiple index use when retrieving data if the number of ORs is small. However, as the number of ORs increases, the retrieval costs that HiRDB expends in internal calculations also increases, and HiRDB may stop applying OR multiple index use. If this happens, specify the value for priority of OR multiple index use so that OR multiple index use is always applied, even if the number of ORs becomes large.

Notes

1. If an AND condition is specified together with the OR conditions, and the AND condition uses an index to narrow the data, that index may be used in the search process.

2. This optimization method is applied when all conditions specified with OR are narrowed with = in the comparison predicate. Also, a single-column index or the index that becomes the first configuration column of a multi-column index must be defined for all columns that were narrowed with =.

3. In a join search of two or more tables, this optimization method may not be applied if HiRDB determines that searching the data by using a joined column index would be faster.

4. For some SQL statements, AND multiple index use, which involves sum sets, is applied instead of OR multiple index use. In such cases, high-speed retrieval is also possible, just as when OR multiple index use is applied. However, if AND conditions are specified, product sets and sum sets may be combined when use of AND multiple indexes is applied.

If the performance is poor when use of AND multiple indexes specified with product sets is applied, you can improve the performance with either of the following methods:

- Specify both *priority of OR multiple index use* and *suppressing use of AND* *multiple indexes* at the same time.

- If several column conditions linked with AND can be narrowed, define a multi-column index that includes these condition columns.

5. If *application of optimizing mode 2 based on cost* is not used in the SQL extension optimizing option, multiple indexes are not used in the join search. However, if there is a condition that cannot be evaluated without applying multiple index use, multiple indexes are used regardless of the specification of this optimization method.

7. Group processing, ORDER BY processing, and DISTINCT set function processing at the local back-end server

   Usually, group processing, ORDER BY processing, and DISTINCT set function processing use a floating server. However, when this optimization is used for a single table search, group processing, ORDER BY processing, and DISTINCT set function processing are performed at the back-end server (local back-end server) where the table is defined.

   For details about grouping processing methods, see *4.5.5 Grouping processing methods (HiRDB/Parallel Server only)*.

   When rapid grouping processing is used, or when as a result of an index search HiRDB determines that there is no need to sort for group processing, ORDER BY processing, or DISTINCT set function processing, a faster processing method is selected.

8. Suppressing use of AND multiple indexes

   This specification prevents the use of access paths that use AND multiple indexes.

   AND multiple indexes are used when a search condition contains multiple conditions connected by AND, and a different index is defined for each column (e.g., SELECT ROW FROM T1 WHERE C1=100 AND C2=200). In this case, the indexes are used to create work tables for the rows that satisfy the conditions, and a product set is obtained from the resulting work tables.

   If the AND multiple indexes being used include OR multiple indexes, this specification suppresses multiple index use for the AND portion but not for the OR portion.

   Depending on the data characteristics, the product set is effective in some cases and can worsen performance in others. Multiple index use is effective when using those indexes significantly narrows the number of items to be searched, and the amount of duplicated data is reduced when the product set is taken.

   Apply this optimization method if you think that using AND multiple indexes is not effective.

   The use of AND multiple indexes cannot be suppressed if a query specification contains multiple conditions that include columns for the same table, in the following locations:

- In a retrieval condition for a structured repetition predicate
- In the first argument of a plug-in distribution function that retrieves data with an index

9. Rapid grouping processing

   This optimization method uses hashing to rapidly process the groups specified in the `GROUP BY` clause of the SQL statement.

   For details about the rapid grouping facility, see *4.9 Rapid grouping facility*.

10. Limiting the target floatable servers (back-end servers for fetching data)

    Normally, back-end servers that are not used for fetching data are used as floating servers. When this optimization method is applied, only back-end servers used for fetching data are used as floating servers. This application is permitted only for a HiRDB/Parallel Server. For details about how to allocate floatable servers, see *4.5.4 Allocating floatable servers (HiRDB/Parallel Server only)*.

11. Separating data collecting servers

    If increasing the target floatable servers (back-end servers for fetching data) or limiting the target floatable servers (back-end servers for fetching data) is specified, the separating data collecting server method is applied.

    When this method is applied, the back-end servers that are not transmitting data are allocated for data collecting if the SQL statement requires data from multiple back-end servers to be collected in one back-end server. The back-end servers that are not used for data collecting (included back-end servers for fetching data) are allocated as floatable servers for other uses.

    For details about how to allocate floatable servers, see *4.5.4 Allocating floatable servers (HiRDB/Parallel Server only)*.

12. Suppressing index use *(forced table scan)*

    Normally, the HiRDB system determines whether or not an index is to be used. When this optimization method is applied, the method that does not use an index is forcibly used.

    However, index use cannot be suppressed if nest loops are linked with the `JOIN` command, a structured repetition predicate is specified in the retrieval condition, or a condition for an index-type plug-in-dependent function is specified.

13. Forcing use of multiple indexes

    Specify this optimization option to forcibly select the use of `AND` multiple indexes when searching tables.

If several conditions linked with AND area specified and this optimization method is not specified, normally only up to two indexes are used even if AND multiple index use is selected. The number of indexes to be used changes slightly according to the table definitions, index definitions, and search conditions.

When this optimization method is specified, all conditions that can narrow the search range by using indexes are used.

The use of AND multiple indexes is effective when using those indexes significantly narrows the number of items to be searched, and the amount of duplicated data is reduced when the product set is taken.

If *application of optimizing mode 2 based on cost* is not used in the SQL extension optimizing option, multiple indexes are not used in the join search. However, if there is a condition that cannot be evaluated without applying multiple index use, multiple indexes are used regardless of the specification of this optimization method.

14. Suppressing creation of update-SQL work tables

    If this optimization method is used when index key value with no lock is applied, HiRDB does not create work tables for internal processing even if an index is used for a search, UPDATE statement, or DELETE statement specified in the FOR UPDATE clause. The SQL statement can therefore be processed at high speed.

    If an index key value with no lock is not applied, HiRDB creates work tables.

    To check whether an index is being used, use the access path display utility.

    If you specify suppressing creation of update-SQL work tables and also use non-locking of index key values, the restrictions of table manipulation when a cursor is used are relaxed.

    The following table shows the relationships between SQL statements that create work tables and the option for suppressing creation of update-SQL work tables.

Table 6-34: Relationships between SQL statements that create work tables and suppressing creation of update-SQL work tables

SQL statement		**Indexes are used**		**Indexes are not used**
		This optimization is applied	**This optimization is not applied**	
SELECT statement	FOR UPDATE[#1]	--	C	--
	FOR UPDATE OF	--	C[#2]	--
	FOR READ ONLY	C	C	C
	ORDER BY	C[#4, #5]	C[#4]	C
	Previous clauses are not specified	--[#5]	--	--
UPDATE statement	Only values are specified for updated values in the SET clause	--	C[#3]	--
	A non-value entity is specified for an updated value in the SET clause	C[#3]	C[#3]	--
DELETE statement		--	C	--

C: Work tables are created.

--: Work tables are not created.

#1: This includes FOR UPDATE clauses that are assumed when data is updated using the cursor of this SELECT statement.

#2: Work tables are not created if the configuration column of the index to be used is not specified in the FOR UPDATE OF column name.

#3: Work tables are not created if the configuration column of the index to be used is not specified in the column to be updated on the left side of the SET clause.

#4: Work tables for the ORDER BY clause are sometimes not created when indexes are used.

#5: A table from which data is being retrieved with a cursor can be updated by another SQL statement. However, if the index that is being used for retrieval with the cursor is updated, the results retrieved with the cursor are not guaranteed.

15. Deriving high-speed search conditions

When this optimization method is specified, HiRDB derives high-speed search conditions.

A high-speed search condition refers to a condition that is derived from a WHERE clause search condition or an ON search condition in a FROM clause by CNF conversion or condition shifting. When a high-speed search condition is derived, the retrieval performance improves because the rows to be retrieved are narrowed down at an early stage. However, generating and executing a high-speed search condition may take a long time, and the intended access path is not always achieved. Therefore, whenever possible, specify the high-speed search condition directly in an SQL statement instead of specifying this optimization option. For details about deriving high-speed search conditions, see *4.5.11 Deriving high-speed search conditions*.

16. Applying key conditions that include scalar operations

When this optimization method is specified and all columns included in the scalar operation specified in a limiting condition are index configuration columns, HiRDB narrows the search by evaluating the condition for each index key value. This condition is evaluated as a key condition.

HiRDB operation when applying key conditions that include scalar operations is specified

When HiRDB uses an index to retrieve data, it evaluates the data in the following sequence:

1. HiRDB narrows the search range of the index (search condition).

2. For the narrowed results obtained in 1., HiRDB evaluates the condition for each key value of the index and narrows the search further (key condition).

3. For all key values evaluated as true in 2., HiRDB use the row identifier (ROWID) to reference the data page and evaluate the condition.

If this optimization method is not specified, conditions that include a scalar operation are evaluated as described in 3. If this optimization method is specified, such conditions are evaluated as described in 2. Consequently, the number of rows for which data page referencing is performed becomes smaller, and the number of input/output operations can be decreased. For details about search conditions and key conditions, see the manual *HiRDB Version 9 Command Reference*.

Notes on applying key conditions that include scalar operations

1. When this optimization method is specified, HiRDB judges that narrowing using indexes is an effective method of data retrieval, and consequently, it becomes easier for indexes to be used. Therefore, do not specify this option unless beneficial results can be expected when indexes in conditions containing scalar operations are used to narrow searches.

2. A condition is not evaluated as a key condition if one of the following applies:

- The condition includes a columns that is not an index configuration column.
- The condition includes a system-defined scalar function.
- The condition includes the system built-in scalar function `IS_USER_CONTAINED_IN_HDS_GROUP`.
- The condition includes function calling.
- The condition includes a repetition column that has an integer subscript.

3. The evaluation of a structured repetition predicate that includes a scalar operation causes an error because such predicates cannot be evaluated without using an index. Therefore, the key condition is applied, even if this optimization method is not specified.

17. Facility for batch acquisition from functions provided by plug-ins

If a function provided by a plug-in is specified in the search conditions and HiRDB uses a plug-in index to retrieve data, HiRDB normally obtains the results returned from that function (row position information and, if necessary, passing inter-function values) one row at a time.

When this optimization method is applied, the number of times the function provided by the plug-in is called can be decreased because the results returned by the function provided by the plug-in can be obtained in batches of multiple rows. Consequently, the retrieval performance also improves. Note that when the facility for batch acquisition from functions provided by plug-ins is applied, HiRDB creates an internal work table.

Even if this optimization method is not specified, HiRDB sometimes unconditionally applies the facility for batch acquisition from functions provided by plug-ins if it determines that data can be retrieved at high-speed if the facility is always applied. The following table describes the cases when the facility for batch acquisition from functions provided by plug-ins is applied.

Type of specified SQL statement		Specification of facility for batch acquisition from functions provided by plug-ins	
		No	Yes
SQL statement that requires a work table for the base table search results#	Function that does not support batch acquisition	NA	NA
	Function that supports batch acquisition	UA	UA
SQL statement that does not require a work table for the base table search results#	Function that does not support batch acquisition	NA	NA
	Function that supports batch acquisition	NA	WTA

Legend:

UA:

The facility for batch acquisition from functions provided by plug-ins is applied unconditionally.

WTA:

A new work table is created, and the facility for batch acquisition from functions provided by plug-ins is applied.

NA:

The facility for batch acquisition from functions provided by plug-ins is not applied.

#: For details about SQL statements that require a work table file, see the manual *HiRDB Version 9 Description*.

Notes on the facility for batch acquisition from functions provided by plug-ins

1. When HiRDB obtains results returned by a function provided by a plug-in, it must create a work table internally. Normally, this optimization method improves the retrieval performance since the time needed to create a work table is usually shorter than the time needed to accept results returned one row at a time. However, sometimes the retrieval performance drops when this optimization method is specified. Therefore, if the effect of the drop in performance is large, do not specify this optimization method.

2. If this optimization method is specified for a retrieval in which a work table is not created, the time when the first FETCH occurs is delayed. This is

because the process that returns results to the client each time a row is fetched changes to a process that returns results to the client after all rows satisfying the search condition of the function provided by the plug-in are fetched and the work table is created. If a drop in the performance of the first `FETCH` process becomes a problem, do not specify this optimization method.

3. When this optimization method is applied, the required memory size increases because the results of the functions provided by plug-ins are obtained in batches of multiple rows. For details about the required memory size, see the *HiRDB Version 9 Installation and Design Guide*.

18. Facility for moving search conditions into derived table

    Normally, a work table for a derived table contains rows that do not satisfy the search condition for the derived table columns. If this optimization method is applied, the HiRDB system first eliminates the rows that do not satisfy the search conditions for the derived table columns, and then creates a work table for the derived table. This enables you to reduce the space used for work tables as well as the number of input/output operations on work tables. By using such an access path, you might be able to use indexes that are more efficient in narrowing down the rows to be retrieved.

    When you install HiRDB version 08-02 or later for the first time, we recommend that you use this function.

    Notes about the facility for moving search conditions into derived table:

    If you upgrade your HiRDB system from a version earlier than 08-02 and use this function, the access path might change. If the access path changes, the selected index might not be able to efficiently narrow down the rows to be retrieved, thereby adversely affecting performance. In such a case, check the access path before and after you use this function to make sure that the optimum index is being used. If the optimum index is not being used, specify the index used in the SQL optimization specification for a used index, or stop using this function.

Notes

1. For a table for which no indexes have been defined, the following optimization methods have no effect, even if they are specified:

   Forced nest-loop-join

   Making multiple SQL objects

   Prioritized nest-loop-join

   Priority of `OR` multiple index use

   Suppressing use of `AND` multiple indexes

Suppressing index use (forced table scan)

Forcing use of multiple indexes

Suppressing creation of update-SQL work tables

Applying key conditions that include scalar operations

2. When the `ASSIGN LIST` statement is used (except in subqueries), HiRDB always uses indexes to retrieve data. Therefore, specifying the following optimization methods has no effect:

   Suppressing use of `AND` multiple indexes

   Suppressing index use (forced table scan)

3. If optimizing information is not retrieved with the optimizing information collection utility (`pdgetcst`), specifying the value for making multiple SQL objects has no effect.

4. If the number of groups to be grouped is large, *rapid grouping processing* may not have any effect. In this case, a value of the required size should be specified by estimating `PDAGGR`. Specifying a value larger than the estimated value will have no beneficial effect. Remember that the use of process-specific memory increases as a larger value is specified.

5. For SQL statements that use all back-end servers to fetch data, even if the value for increasing the target floatable servers (back-end servers for fetching data) is not specified, the back-end servers for fetching data are used as floatable servers.

6. If the optimization option values for priority of `OR` multiple index use and suppressing index use (forced table scan) are specified at the same time, the specification for priority of `OR` multiple index use becomes ineffective.

7. If the optimization option values for suppressing use of `AND` multiple indexes and forcing use of multiple indexes are specified at the same time, multiple index use is suppressed for `AND` sections and is forced for `OR` sections.

8. If the optimization option values for forcing use of multiple indexes and suppressing index use (forced table scan) are specified at the same time, the specification for forcing use of multiple indexes becomes ineffective.

9. SQL optimization option values that are stored in the `SQL_ROUTINES` dictionary table are converted to decimal format (sum of unsigned integers used to specify individual optimization methods).

10. If *Suppressing creation of update-SQL work tables* is specified and a search with a `FOR UPDATE` clause specification is performed, the same row may be retrieved several times if the configuration column of an index used in the search is updated for a value that satisfies the search condition.

    Example:

[Index definition]

```
CREATE INDEX X1 ON T1(C1)
```

[Cursor definition]

```
DECLARE CR1 CURSOR FOR SELECT C1 FROM T1 WHERE C1>0 FOR
UPDATE
```

When this cursor is used and the next `FETCH` and `UPDATE` statements are repeated, the row that was updated to `C1=10` is retrieved several times.

```
FETCH CR1 INTO :XX
UPDATE T1 SET C1=10 WHERE CURRENT OF CR1
```

Correction methods:

- Change the search condition so that the update value in the `UPDATE` statement does not satisfy the search condition of the search.

  Example:

  Change `WHERE C1>0` to `WHERE C1>0 AND C1< >10`.
- Do not specify the optimization option value for suppressing creation of update-SQL work tables for UAPs that have problems with the same row being retrieved more than once. For a stored routine, redefine the routine instead of specifying the optimization option value when defining the routine.
- Delete the affected column from the configuration columns of the indexes to be used in the search. However, if an index configuration column is deleted, the search performance may become worse if that column significantly narrows the search scope with the search condition. Also, if part of an index is deleted, the number of index key duplications increases, and lock wait and deadlock may occur frequently. This correction method is therefore not highly recommended. If you plan to employ this correction method, be sure to verify the effects thoroughly.

The same line may also be updated several times if an index configuration file to be used in an `UPDATE` statement is specified as the column name of the `SET` clause in that `UPDATE` statement, and a value that satisfies the search condition of the `WHERE` clause is specified as the update value.

(89) PDADDITIONALOPTLVL=SQL-extension-optimizing-option[,SQL-extension-optimizing-option]...

~ <identifier or unsigned integer>

This environment variable specifies optimization methods for determining the most efficient access path by taking the database status into consideration.

The SQL extension optimizing methods can be specified with identifiers (character strings) or numbers.

Specifying the SQL extension optimizing methods with identifiers

`PDADDITIONALOPTLVL="`*identifier*`"[,"`*identifier*`"]...`

Examples

- Applying *application of optimizing mode 2 based on cost* and *hash join, subquery hash execution*:

  ```
  PDADDITIONALOPTLVL="COST_BASE_2","APPLY_HASH_JOIN"
  ```

- Applying no optimization method:

  ```
  PDADDITIONALOPTLVL="NONE"
  ```

Rules

1. Specify at least one identifier.
2. When specifying two or more identifiers, separate them with commas.
3. For details about the information (optimization methods) that can be specified with identifiers, see *Specification values for the SQL extension optimizing option*.
4. If no optimization is to be applied, specify `NONE` as the identifier.
5. The identifiers can be specified with uppercase and lowercase characters.
6. Even if the same identifier is specified more than once, HiRDB recognizes only one specification. However, try not to specify the same identifier more than once.
7. The character string specified for `"`*identifier*`"[,"`*identifier*`"]...` can have up to 575 bytes.

Specifying the SQL extension optimizing methods with numbers

`PDADDITIONALOPTLVL=`*unsigned-integer*`[,`*unsigned-integer*`]...`

Examples

- Applying *application of optimizing mode 2 based on cost* and *hash join, subquery hash execution*:

  ```
  PDADDITIONALOPTLVL=1,2
  ```

- Applying no optimization method:

  ```
  PDADDITIONALOPTLVL=0
  ```

Rules

1. Specify at least one unsigned integer.

2. When specifying two or more unsigned integers, separate them with commas.
3. For details about the information (optimization methods) that can be specified with unsigned integers, see *Specification values for the SQL extension optimizing option*.
4. If no optimization is to be applied, specify `0` as the identifier.
5. Even if the same unsigned integer is specified more than once, HiRDB recognizes only one specification. However, try not to specify the same unsigned integer more than once.
6. The character string specified for `"`*unsigned-integer*`"[,"`*unsigned-integer*`"]...` can have up to 575 bytes.

Relationship to the system definition

We recommend that you specify both the following identifiers when you omit the `pd_additional_optimize_level` operand in the system definition:

`COST_BASE_2` and `APPLY_JOIN_COND_FOR_VALUE_EXP`

When this environment variable is omitted, the value specified in the `pd_additional_optimize_level` operand of the system definition is assumed. For details about the `pd_additional_optimize_level` operand, see the manual *HiRDB Version 9 System Definition*.

Relationship with SQL

The SQL extension optimizing option for an SQL statement in a stored procedure is determined by the specifications for `CREATE PROCEDURE`, `CREATE TYPE`, `ALTER PROCEDURE`, or `ALTER ROUTINE`, and is not affected by the `PDADDITIONALOPTLVL` specification.

If SQL optimization specifications are specified in SQL statements, those specifications have priority over the SQL extension optimizing options. For details about SQL optimization specifications, see the manual *HiRDB Version 9 SQL Reference*.

Specification values for the SQL extension optimizing option

The following table shows the specification values for the SQL extension optimizing option.

Table 6-35: Specification values of the SQL extension optimizing option

Number	Optimization method	Specification value	
		Identifier	Unsigned integer
1	Application of optimizing mode 2 based on cost	`"COST_BASE_2"`	`1`
2	Hash join, subquery hash execution	`"APPLY_HASH_JOIN"`	`2`
3	Facility for applying join conditions including value expression	`"APPLY_JOIN_COND_FOR_VALUE_EXP"`	`32`

Note

Item numbers 2 to 3 become effective when the value for application of optimizing mode 2 based on cost is specified.

Recommended specification values

- When HiRDB is installed for the first time, Hitachi recommends that you use optimizing mode 2 based on cost. When you use optimizing mode 2 based on cost to improve the accuracy of optimization, execute the optimizing information collection utility as needed. If the database scope (number of table rows) differs between the test environment and actual environment, the access path may change in the test environment and actual environment when the optimizing information collection utility is executed. In the test environment, specify the numbers of table rows (`NROWS`) being used in the actual environment in the optimizing information parameter file, specify the`-s` option in the optimizing information collection utility, and execute the utility.
- If you are upgrading HiRDB from a version before version 06-00, investigate whether optimizing mode 2 based on cost should be used. If you are setting up the same operating environment that was used before the version upgrade, do not use optimizing mode 2 based on cost. However, some of the SQL statements that are supported from version 06-00 always use optimizing mode 2 based on costs in optimization processing.
- If hash join is not being used, *hash join, subquery hash execution* does not need to be specified.

Explanation of optimization methods

1.Application of optimizing mode 2 based on cost

This optimization method uses optimizing mode 2 based on cost to execute optimization processing. For details about optimizing mode 2 based on cost, see *4.5.1 SQL optimizing modes*.

2.Hash join, subquery hash execution

When a join search is executed, this optimization method applies hash join to optimize the search process. If the search involves a subquery, hashing is used to process the subquery. When deciding whether or not to apply this optimization method, consider the join methods, the execution methods for subqueries with no external references, and the execution methods for subqueries with external references. For details about these methods, see *4.5.6 Join methods*, *4.5.8 Execution of subqueries with no external references*, and *4.5.9 Execution of subqueries with external references*.

When this optimization method is applied, the system definitions must be specified beforehand. For details about preparing for application of hash join and subquery hash execution, see *4.5.10 Preparing for application of hash join and subquery hash execution*.

3. Facility for applying join conditions including value expression

If this function is specified, a join condition is created for conditions including value expressions.

Example:

This example creates a join condition for the following expression:

substr(t1.c1,1,100)=t2.c1

If there are only join conditions including value expressions, this function changes the access path from direct product to nest-loop join, hash join, or merge join, so that high-speed SQL execution can be expected.

Notes

When you install HiRDB version 08-04 or later for the first time, we recommend that you use this function.

If you upgrade your HiRDB system from a version earlier than 08-04 and use this function, the number of join conditions increases and the join sequence and join method might change. If the join sequence and join method change, the selected join sequence might not provide effective narrowing, or the selected join method might exhibit poor performance, thereby having adverse effects on overall performance. Furthermore, if the selected join sequence cannot narrow down the rows efficiently, the intermediate join results cannot be placed in the work table and an error might result. Therefore, check the access path before and after you apply this function to make sure that this function will not affect performance negatively or cause errors. If performance is affected adversely or errors occur due to application of this function, stop using the function or specify the join sequence by using `INNER` or `LEFT OUTER` in the join tables and specify the optimum join method by using an SQL optimization specification.

(90) PDHASHTBLSIZE=hash-table-size-when-hash-join-subquery-hash-execution-is-applied

For 32-bit mode

~ <unsigned integer> ((128-524288)) (kilobytes)

For 64-bit mode

~ <unsigned integer> ((128-2097152)) (kilobytes)

This environment variable specifies the hash table size when the value for hash join, hash subquery execution is specified in the SQL extension optimizing options.

Specify a value that is a multiple of 128. If the specified value is not a multiple of 128, the value is rounded up to the next multiple of 128.

When the server is set to 32-bit mode, an upper limit of 524,288 is assumed when a value from 524,289 to 2,097,152 is specified.

Specification value reference

See *4.5.10 Preparing for application of hash join and subquery hash execution*.

Relationship to the system definition

If this environment variable is omitted, the HiRDB uses the value that was specified in the `pd_hash_table_size` operand of the system definition.

(91) PDDFLNVAL={USE|NOUSE}

This environment variable specifies whether the default value is to be set into an embedded variable if the table data fetched into an embedded variable is a null value.

`USE`

Use the default value setting facility for null values.

`NOUSE`

Do not use the default value setting facility for null values.

For details about the default value setting facility for null values, see the *HiRDB Version 9 SQL Reference* manual.

For an internal Type 4 JDBC driver that is run with a Java stored procedure, use the same `PDDFLNVAL` specification as for the program that calls the Java stored procedure.

(92) PDAGGR=group-count-resulting-from-grouping

For 32-bit mode

~ <unsigned integer> ((0-30000000)) <<1024>>

For 64-bit mode

~ <unsigned integer> ((0-2147483647)) <<1024>>

This environment variable specifies the maximum number of groups allowed in each server so that the memory size used in `GROUP BY` processing can be determined. This environment variable becomes effective when rapid grouping processing is specified as the SQL optimization option.

Estimation method

- When at least 1,024 groups will be created or when the expected performance cannot be achieved:

  Specify a large value for this environment variable; however, consider the balance with the required memory size, and increase the specified value gradually.

  If a memory shortage occurs when 1,024 is specified, specify a value according to the amount of memory available.

- When fewer than 1,024 groups will be created or if a memory shortage occurs:

  Specify a small value for this environment variable. If a value of the required size cannot be specified because the amount of required memory is too large, specify the largest value that can be specified, even if the value is less than the number of groups.

Note

If the specified value is too large, a memory shortage may occur. If the number of groups created exceeds the specified value, processing may become slow because the allocated memory size is insufficient. For details about the formula for calculating the required memory size used by the rapid grouping facility, see the *HiRDB Version 9 Installation and Design Guide*.

(93) PDCMMTBFDDL={YES|NO}

When a definition SQL is to be executed in a transaction that is executing a data manipulation SQL, this environment variable specifies whether the transaction is to be committed automatically before the definition SQL is executed. When the transaction is automatically committed before the definition SQL is executed, the open holdable cursors are closed, and the results of the preprocessed SQL statements become invalid.

`YES`

Automatically commit the transaction that executes the data manipulation SQL before executing the definition. When this value is specified, the open holdable cursors are closed, and the results of the preprocessed SQL statements become invalid.

NO

Execute the definition SQL after explicitly committing the transaction executing the data manipulation SQL.

(94) PDPRPCRCLS={YES|NO}

This environment variable specifies whether an open cursor is to be closed automatically if a PREPARE statement reuses the SQL identifier that is using that open cursor.

PDPRPCRCLS becomes effective if the -Xe option is not specified during preprocessing. For details about preprocessing, see *8.2 Preprocessing*.

YES

Close the open cursor automatically.

NO

Do not close the open cursor automatically.

(95) PDAUTOCONNECT={ON|OFF}

This environment variable specifies whether the HiRDB client is to connect automatically with the HiRDB server if an SQL statement is executed while the client is not connected to the server.

ON

Connect to the HiRDB server automatically and then execute the SQL statement.

OFF

Do not connect to the HiRDB server automatically. In this case, the SQL statement generates an error (SQLCODE=-563).

If the SET SESSION AUTHORIZATION statement is executed while the HiRDB client is not connected to the HiRDB server, an error (SQLCODE=-563) occurs regardless of the PDAUTOCONNECT specification.

When you create a UAP, Hitachi recommends that you specify OFF in PDAUTOCONNECT because the HiRDB client must be able to determine whether it is properly connected to HiRDB.

(96) PDDDLDEAPRPEXE={YES|NO}

If a transaction that is already running (called the *preceding transaction*) is already making normal use of a resource (such as a table, stored routine, or abstract data type) that is to be manipulated by a transaction that executes a definition SQL statement (called the *definition transaction*), the definition transaction waits for the preceding transaction to be completed. If you specify YES in this environment variable, the

HiRDB system ignores the preceding transaction's preprocessing results and executes the definition transaction.

Reference note:

You can specify the timing of executing a definition transaction by specifying the following environment variables:

1. PDDDLDEAPRPEXE

   Ignores the preceding transaction's preprocessing results and executes the definition transaction.

2. PDDDLDEAPRP

   Executes the definition transaction after the preceding transaction's holdable cursor has closed and the transaction including that holdable cursor has been completed. When the definition transaction is executed, the holdable cursor's preprocessing results are ignored.

3. PDLCKWAITTIME

   Enables you to specify the lock-release wait time. If you use this environment variable together with PDDDLDEAPRP and PDLCKWAITTIME, you can prevent the definition transaction from resulting in a timeout error before the preceding transaction is completed.

The following table shows whether the preceding transaction's preprocessing results are ignored when this environment variable is specified together with PDDDLDEAPRPEXE and PDDDLDEAPRP.

Client environment definition		**Preprocessing results used by holdable cursor**	**Preprocessing results that are not used by holdable cursor**
PDDDLDEAPRPEXE	**PDDDLDEAPRP**		
YES	YES	Y	Y
	NO	Y	Y
NO	YES	AP	N
	NO	N	N

Legend:

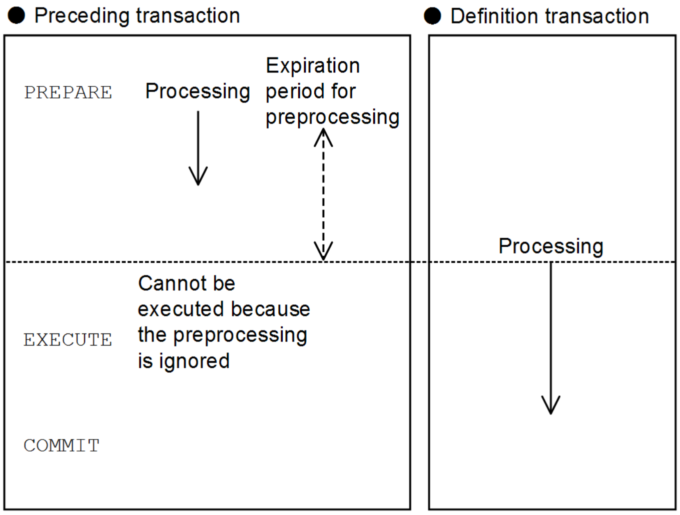
Y: Can be ignored.

AP: Can be ignored after the preceding transaction is completed.

N: Cannot be ignored.

`YES`

Ignore the preceding transaction's preprocessing results and execute the definition transaction. The following shows an example when `YES` is specified:


● Preceding transaction
● Definition transaction
PREPARE
Processing
Expiration period for preprocessing
Processing
EXECUTE
Cannot be executed because the preprocessing is ignored
COMMIT

Explanation:

The definition transaction does not wait for completion of the preceding transaction.

When the definition transaction is executed, the preceding transaction's preprocessing results are ignored and the preceding transaction can no longer be executed.

Notes

You can ignore the preprocessing results by specifying `YES` in this environment variable only when the definition transaction performs the following operations (except when the preceding transaction is executing preprocessing):

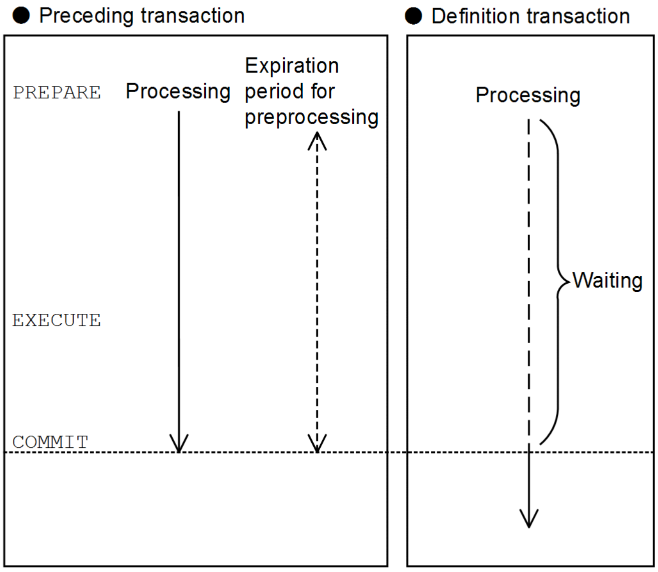
- Executes a definition SQL statement
- Closes the data dictionary LOB RDAREA for storing stored routines' SQL objects
- Any of the following operations performed by the database structure modification utility (`pdmod`):
  - Deleting a data dictionary RDAREA for storing database state analyzed tables (`SQL_DB_STATE_ANALYZED`) and database management tables (`SQL_DB_MANAGEMENT`) by using the `remove rdarea` statement (for deleting RDAREAs)

- Changing the reference privilege or mixed character string data usage setting by using the `alter system` statement (for changing the data dictionary table attribute definitions)
- Renaming an RDAREA by using the `alter rdarea` statement (for changing RDAREA attributes)

Note that if the preceding transaction is executing a definition SQL statement (including the rebalancing utility), a definition transaction that uses the table, procedure, or function whose definition is being changed will result in an error.

`NO`

Do not execute a definition transaction prior to completion of the preceding transaction. The following shows an example when `NO` is specified:



Explanation:

The definition transaction waits for completion of the preceding transaction regardless of the SQL statement being executed by the preceding transaction.

The definition transaction is executed after the preceding transaction has been completed.

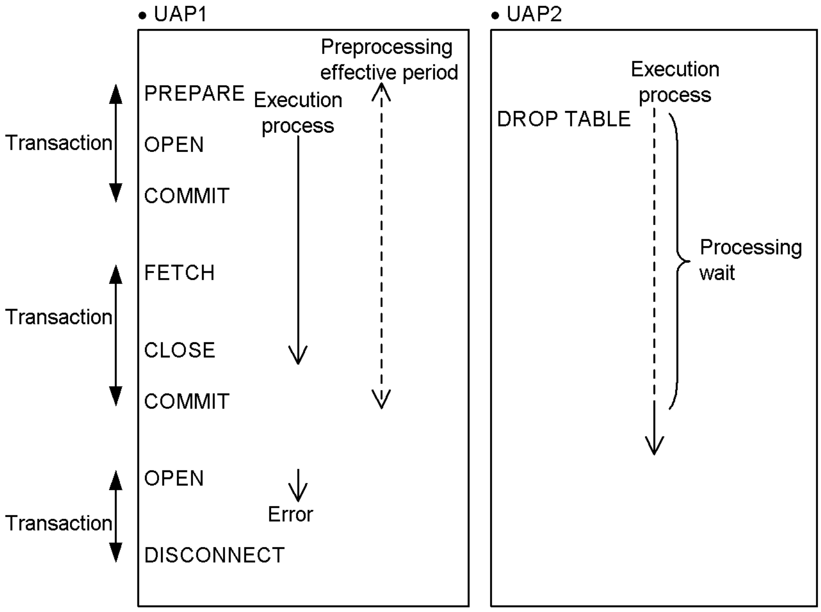
(97) PDDDLDEAPRP={YES|NO}

This environment variable specifies whether the definition information of a table being used by a closed holdable cursor can be changed by another UAP between transactions. When a definition SQL is executed, the preprocessing of the holdable cursor becomes invalid.

YES

Allow another UAP to change the definition information of the table between transactions of the UAP using the holdable cursor.

The following figure shows an example of the processing when YES is specified.



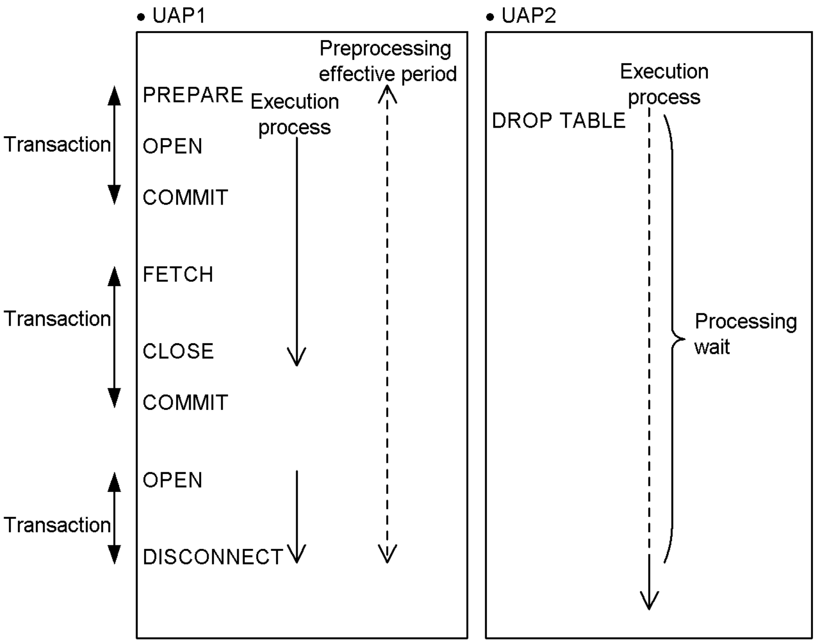
Explanation

The definition SQL executed by UAP2 can be executed after the holdable cursor of UAP1 is closed and the transaction containing that holdable cursor is completed. Also, if the holdable cursor of UAP1 is reopened, error SQLCODE=-1512 occurs (the processing becomes invalid).

NO

Do not allow another UAP to change the definition information of the table between transactions of the UAP using the holdable cursor.

The following figure shows an example of the processing when NO is specified.



Explanation

The definition SQL executed by UAP2 can be executed after DISCONNECT processing of UAP1.

Relationship with other environment variables

You can specify a lock-release wait time in PDLCKWAITTIME. By combining PDDDLDEAPRP and PDLCKWAITTIME, you can prevent a definition SQL statement from resulting in a timeout error before the transaction is completed.

(98) PDLCKWAITTIME=lock-release-wait-time

~<unsigned integer>((0 to 65535)) <<pd_lck_wait_timeout value in system definition>> (seconds)

This environment variable specifies the maximum amount of time the HiRDB client is to monitor a lock request for the lock to be released, beginning when the lock request is placed on wait status. If the lock is not released within the specified time, the SQL statement returns an error. If 0 is specified, the HiRDB client waits until the lock is released without monitoring the wait status.

(99) PDCURSORLVL={0|1|2}

This environment variable specifies the timing for sending an open/close cursor request from a HiRDB client to the HiRDB server when performing a search using the cursor. By specifying this environment variable, the request is not sent to the HiRDB server when the open cursor request is received from the application but rather the open cursor is requested when data is fetched for the first time. Also, when it is detected that there is no data to be searched (`SQLCODE=100`), the cursor is closed. This environment variable reduces the communication overhead.

`0`:

The HiRDB client requests execution of cursor open/close to the HiRDB server when it receives a request from the application.

`1`:

When there is no data to be searched, the HiRDB server closes the cursor when it returns `SQLCODE=100`, without a request from the HiRDB client. If the HiRDB client has already detected `SQLCODE=100` when it receives a close cursor request from the application, the HiRDB client does not send the close cursor request to the HiRDB server. The close cursor request is sent only when `SQLCODE=100` has not been detected.

For an open cursor request, the operation is the same as when 0 is specified.

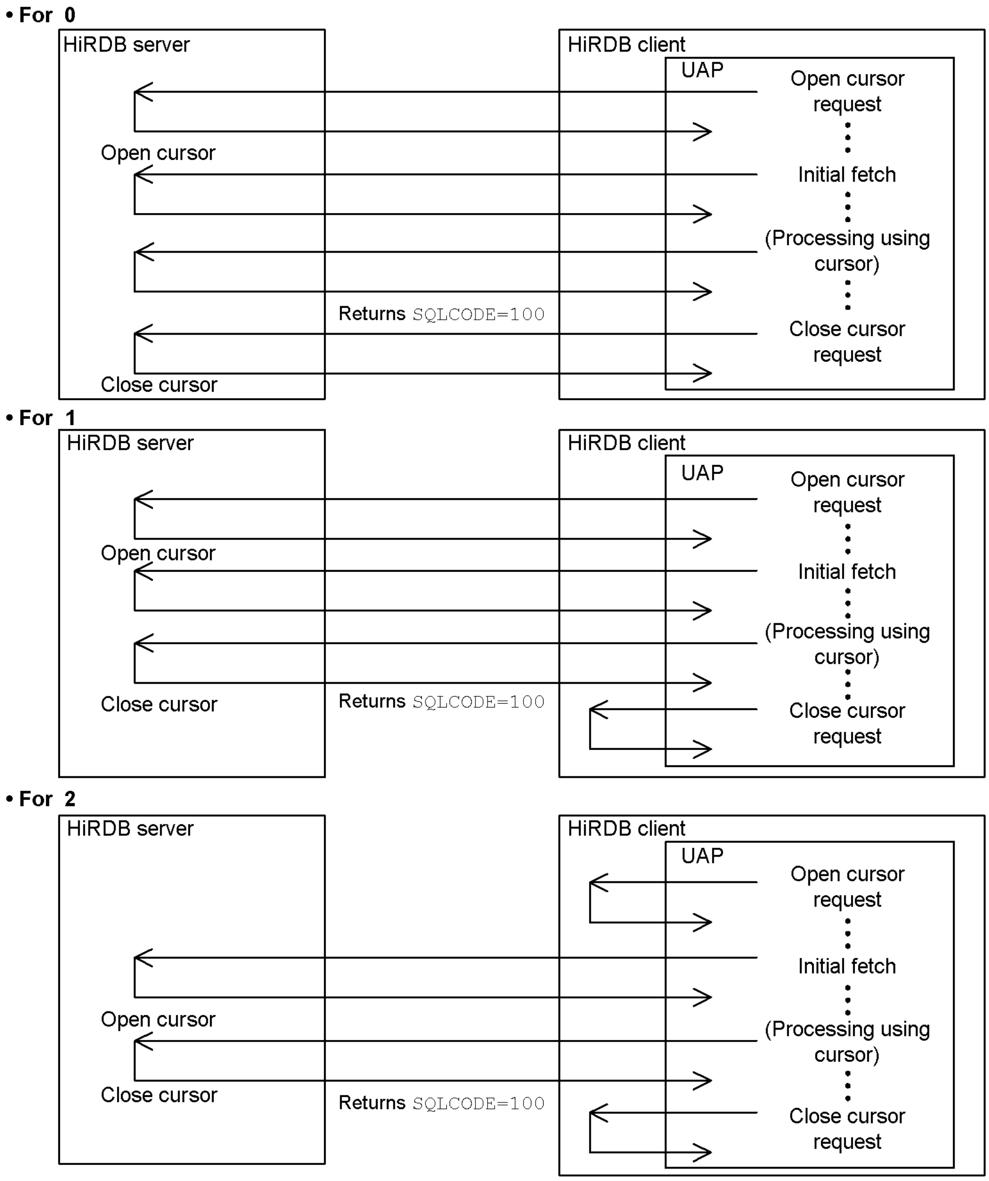
`2`:

When the client receives an open cursor request from the application, it does not request that the HiRDB server execute it, but requests opening of the cursor at the same time it sends the initial request to fetch data.

For the close cursor request, the operation is the same as when 1 is specified.

The following figure provides an overview of the processing for each setting.

Figure 6-4: Overview of processing for each setting of `PDCURSORLVL`



Notes

- Even when `1` or `2` is specified for this environment variable, if the client receives a request to close the cursor for a results-set returned from the procedure, the client requests that the HiRDB server execute the request.

- When `1` or `2` is specified for this environment variable, a cursor close is added to the number of SQL executions in the UAP statistical information, but the cursor close is not output to the SQL statistical information. Also, when `2` is specified for this environment variable, a cursor open is added to the number of SQL executions in the UAP statistical information, but the cursor open is not output to the SQL statistical information.
- When `1` or `2` is specified for this environment variable, the open/close cursor operation code is output to the SQL trace. If the `FETCH` statement is used to open or close the cursor, SQL statistical information, access path information, and SQL runtime interim results of the open/close cursor are output to the `FETCH` side.
- In the case of a HiRDB/Parallel Server, the first data fetch may take a long time if there is a long delay from when the open cursor is executed until the first data is fetched.
- When `2` is specified for this environment variable, even though the `PREPARE` statement is executed again on a cursor that is open before the initial fetch call, an error does not occur because the open cursor request is not sent to the HiRDB server. When the `PREPARE` statement is executed again, the cursor opening needs to be executed again because the `PREPARE` statement information is used as cursor information.
- When `1` or `2` is specified for this environment variable, even though the `PREPARE` statement or `OPEN` statement is executed without executing the `CLOSE` statement after `SQLCODE=100` is detected, an error does not occur because the cursor is already closed. Also, when the `FETCH` statement is executed following detection of `SQLCODE=100`, `SQLCODE=-501` (which indicates that the cursor is not open) is returned without producing a no-data-to-be-searched situation.

(100) PDDELRSVWDFILE=SQL-reserved-word-deletion-file-name

~ <identifier> ((up to 8 characters))

This environment variable specifies the name of the SQL reserved word deletion file when the SQL reserved word deletion facility is used. The keywords to be deleted from the SQL reserved words are specified in the SQL reserved word deletion file.

Relationship to the system definition

When `PDDELRSVWDFILE` is specified, an SQL reserved word deletion file must be specified in the `pd_delete_reserved_word_file` operand of the system definition. For details about the SQL reserved word deletion file, see the manual *HiRDB Version 9 System Definition*.

Note

For a Windows edition HiRDB, the SQL reserved word deletion file name is not case sensitive. Note. therefore, that files having the same name except for case differences are treated as the same file.

(101) PDHJHASHINGMODE={TYPE1|TYPE2}

This environment variable specifies the hashing method when Apply hash join, subquery hash execution is selected as the SQL extension optimizing option.

`TYPE1`

This specification preserves the HiRDB performance found in versions before version 07-02.

`TYPE2`

Hashing is performed more uniformly compared to `TYPE1`.

Specification value guidelines

- Normally specify `TYPE2`. However, if the hashing method does not distribute the records uniformly because of the data in the columns specified in the join condition, specify `TYPE1`.
- `TYPE1` is the HiRDB hashing method found in versions before version 07-02. If `TYPE1` is specified after the HiRDB version is upgraded and the expected performance is not achieved, specify `TYPE2`.

Relationship to the system definition

When this environment variable is omitted, the specification of the `pd_hashjoin_hashing_mode` operand in the system definition is assumed.

(102) PDCALCMDWAITTIME=maximum-wait-time-for-CAL-COMMAND-statemen t

~ <unsigned integer>>((0 to 65535)) <<`PDCWAITTIME` specification value>> (seconds)

This environment variable specifies the maximum amount of time the HiRDB client is to wait for termination of a command executed by the `CALL COMMAND` statement or a utility, beginning when its execution starts. If no response is received from the server within the specified time, an error is returned to the UAP and the server's process is canceled. If you specify `0`, the HiRDB client will wait until a response is received from the HiRDB server.

(103) PDSTANDARDSQLSTATE={YES|NO}

This environment variable specifies whether the details of the `SQLSTATE` value are to be displayed.

YES

Display the details of the `SQLSTATE` value.

NO

Do not display the details of the SQLSTATE value.

Relationship to the system definition

If this environment variable is omitted, the pd_standard_sqlstate operand value in the system common definition is assumed. However, if an error occurs before connection establishment with the server is completed, HiRDB assumes that NO is specified in this environment variable.

For details about SQLSTATE, see the manual *HiRDB Version 9 Messages*.

If a Java stored procedure uses a type 4 JDBC driver, whether the details of the SQLSTATE value returned to the Java stored procedure are to be set depends on the combination of the client's (program that called the Java stored procedure) PDSTANDARDSQLSTATE specification, pd_standard_sqlstate in the system common definition, and PDSTANDARDSQLSTATE in the Java stored procedure. The following table shows the combinations of these specifications:

Value of client's PDSTANDARDSQL STATE environment variable	Value of pd_standard_sqlstat e in the system common definition	Value of PDSTANDARDSQLSTAT E in Java stored procedure	Whether details of SQLSTATE are to be set
YES	y, n, or omitted	YES	Details are set
Omitted	y	YES	
NO	y, n, or omitted	NO or omitted	Details are not set
Omitted	n or omitted	NO or omitted	

The value of SQLSTATE returned from the ODBC driver follows the ODBC standards regardless of the pd_standard_sqlstate specification in the system common definition.

(104) PDBLKF=block-transfer-row-count

~ <unsigned integer> ((1-4096)) <<1>>

This environment variable specifies the number of rows to be sent in one transfer when the server transfers retrieval results to the client.

Note that the actual number of rows that are sent changes according to the specification value of the PDBLKBUFFSIZE client environment definition. For details about the number of rows to be sent, see *4.7(4) Number of rows transferred in one transmission*.

Specifying a large value reduces the communication overhead and shortens the retrieval time, but much more memory becomes necessary. Therefore, be sure to consider the balance of retrieval time and memory.

For details about the calculation expression for memory required in the server, see *Formula for the size of memory required during block transfer or array FETCH* in the manual *HiRDB Version 9 Installation and Design Guide*. The calculation expression for memory required in the client is shown below.

Memory calculation expression (kilobytes)

$$= \uparrow(600 + 19 \times \textit{retrieval-column-count} + (7 \times \textit{retrieval-column-count} + \Sigma\ \textit{defined-column-lengths}^{\#}) \times \textit{PDBLKF-value}) \div 4096 \uparrow \times 4$$

#: The unit is bytes.

(105) PDBINARYBLKF={YES|NO}

This environment variable specifies whether the block transfer facility is to be applied when a table having a `BINARY`-type selection expression with a defined length of 32,001 bytes or more is searched. For details about the block transfer facility, see *4.7 Block transfer facility*.

`YES`

Apply the block transfer facility.

`NO`

Do not apply the block transfer facility.

When this value is specified, the data is transferred one row at a time even if the value specified in the `PDBLKF` client environment definition is `2` or higher and the value specified in `PDBLKBUFFSIZE` is `1` or higher.

(106) PDBLKBUFFSIZE=communication-buffer-size

~ <unsigned integer> ((0-2000000)) <<0>> (kilobytes)

This environment variable specifies the size of the server-client communication buffer used by the block transfer facility.

If `0` is specified, HiRDB calculates the communication buffer size (in units of bytes) from the value of the `PDBLKF` client environment definition and the maximum length of one row.

The value specified in `PDBLKBUFFSIZE` affects the following values for buffer size and number of rows:

- Size of the server-client communication buffer used for search result transfer
- Number of search result rows that a single server or a front-end server sends to the client during one communication

(107) PDBLKFUPD={YES|NO}

This environment variable specifies whether the block transfer facility is to be suppressed when an extended cursor is used to retrieve a query with FOR UPDATE specified. When you specify this environment variable, you can perform retrieval using an extended cursor for a query with FOR UPDATE specified even when the block transfer facility is used.

YES

Suppress the block transfer facility.

NO

Do not suppress the block transfer facility.

Application standard

In general, you must either specify YES in this environment variable or omit this operand.

If you specify NO, the results of update processing using an extended cursor might differ from the results obtained when YES is specified. The following shows an example.

Example:

This example searches a table (T1) containing 50 rows of data by using an extended cursor while block transfer is enabled (PDBLKF=100), and then uses the extended cursor to update the rows.

```
PREPARE  :SQL1 FROM 'SELECT * FROM T1 FOR UPDATE'   1
ALLOCATE GLOBAL  :CR1 CURSOR FOR :SQL1              2
OPEN :CR1                                           3
FETCH :CR1 INTO :C1;                                4
UPDATE SET C1=999 WHERE CURRENT OF :CR1             5
```

The row to be updated in line 5 varies according to the value specified in this environment variable as described below:

- When YES is specified

  The row at the cursor location that was fetched in line 4 is updated.

- When NO is specified

  Because the cursor position in line 4 has moved by the number of blocks, the row at the resulting cursor position is updated. If there is no row at the resulting cursor position, an error results.

Notes

If you enable this function, note the following:

- For a UAP written in C or C++ language, you must link a client library version 07-03 or later.
- For a UAP written in COBOL, you must use a preprocessor version 07-03 or later to create a post-source program.

(108) PDBLKFERRBREAK={YES|NO}

This environment variable specifies when an error is to be returned to the UAP after implicit rollback occurs while multiple rows are being acquired by the block transfer facility.

If an error occurs while multiple rows are being acquired from the HiRDB server, the HiRDB client returns an error to the UAP at the actual row where the error occurred. Depending on the nature of the error, implicit rollback might occur and the transaction might no longer be valid. In such a case, if another SQL statement is issued while the retrieval results are being acquired, the obtained results might be invalid. By specifying this environment variable, you can return an error when the UAP acquires the first row of the rows in the retrieval results held by the HiRDB client.

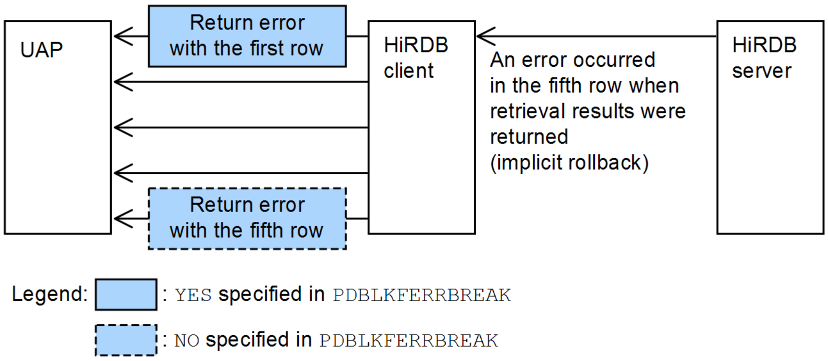
YES

Return an error when the first row of the rows in the retrieval results held by the HiRDB client is acquired.

NO

Return an error when the actual row resulting in the error is acquired.

The following figure illustrates the timing of acquiring an implicit rollback error:


UAP
Return error with the first row
HiRDB client
An error occurred in the fifth row when retrieval results were returned (implicit rollback)
HiRDB server
Return error with the fifth row
Legend:
: YES specified in PDBLKFERRBREAK
: NO specified in PDBLKFERRBREAK

(109) PDNODELAYACK={YES|NO}

This environment variable is limited to the AIX version.

This environment variable specifies whether immediate acknowledgment is to be used for communication between the HiRDB server machine and the HiRDB client machine. For details about using immediate acknowledgment for HiRDB communication, see the *HiRDB Version 9 Installation and Design Guide*.

YES

Use immediate acknowledgment.

NO

Do not use immediate acknowledgment.

Notes

- This environment variable is not valid when the HiRDB server to be connected is on the same machine.
- If you use the tcp_nodelayack OS parameter to specify sending of an immediate acknowledgement, the capability to delay acknowledgment sending is suppressed throughout the entire system. In such a case, immediate acknowledgment is used in the entire system regardless of the setting of this environment variable.

Relationship with system definition

- When the HiRDB server is another server machine in an AIX version environment, immediate acknowledgment can also be used for HiRDB servers. To use immediate acknowledgment between HiRDB servers, set Y for the pd_ipc_tcp_nodelayack operand in the system common definition.

(110) PDBINDRETRYCOUNT=bind-system-call-retries-count

~ <integer> ((-1 to 1000))<<10>>

This environment variable specifies the number of retries that can be attempted when EADDRINUSE is returned for a bind system call in a UNIX domain.

If the client and server are on the same host, the HiRDB client prepares for UNIX domain communication by issuing a bind system call to assign a named file to a socket. In Solaris 9 or later, if SQL DISCONNECT and SQL CONNECT are executed within a brief interval, the bind system call might return EADDRINUSE. You can avoid this by re-issuing the system call.

If you specify -1, the HiRDB client retries until EADDRINUSE is no longer returned. If you specify 0, the HiRDB client returns an error to the UAP without retrying.

Specification value guidelines

If the KFPZ02444-E message has been output to the client error log file due to the network failure indicated in the KFPA11723-E message (reason=NETWORK) and the message displays func=bind and errno=EADDRINUSE with count and interval, increase the value of this environment variable as well as the value of PDBINDRETRYINTERVAL.

(111) PDBINDRETRYINTERVAL=bind-system-call-retry-interval

~ <unsigned integer> ((0 to 1000)) <<0>> (milliseconds)

This environment variable specifies the retry interval when EADDRINUSE is returned for a bind system call in a UNIX domain. If you specify 0, the HiRDB client retries immediately.

(112) PDCLTSIGPIPE={CATCH|IGNORE}

This environment variable specifies whether the HiRDB client's signal handler is to be set for the SIGPIPE signal.

This environment variable is applicable only to the UNIX edition multithread-capable client libraries. For details about the multithread-capable libraries, see *6.4.1 Directories and files for UNIX clients*.

CATCH

Set the HiRDB client's signal handler in the SIGPIPE signal.

IGNORE

Set SIG_IGN (ignore) in the SIGPIPE signal.

The HiRDB client sets a signal handler in the SIGPIPE signal. If you execute a UAP using the multi-connection facility in a multi-thread environment, the signal handler set by the HiRDB client might be used as is. If a UAP execution process unloads the HiRDB client libraries in this status, the process will terminate abnormally when SIGPIPE occurs. If you use this environment variable to set SIG_IGN (ignore) in the SIGPIPE signal, you can avoid setting the HiRDB client's signal handler while a UAP execution process is running.

Notes

In a multi-thread environment, if the HiRDB client libraries might be unloaded by a process of the UAP (such as Cosminexus or TP1/EE) using COBOL, specify SIG_IGN (ignore) for the SIGPIPE signal. If you do not specify SIG_IGN (ignore), you must make sure that the HiRDB client libraries will not be unloaded.

(113) PDDBACCS=generation-number-of-RDAREA-to-be-accessed

~ <unsigned integer> ((0-10))

If the inner replica facility is being used and an RDAREA that is not the current RDAREA in the inner replica group is to be accessed, this environment variable specifies the generation number of that RDAREA. The generation number of the original RDAREA is 0. If you omit this environment variable, the value of PDDBACCS in the UAP environment definition is assumed.

PDDBACCS is applied to all inner replica groups defined in HiRDB. If a replication RDAREA of the generation specified in PDDBACCS is not defined, the current RDAREA in the target inner replica group is processed. Therefore when setting up a test environment that uses replica RDAREAs, you must check that replica RDAREAs of the specified generation are defined for all RDAREAs to be accessed. This is so that an RDAREA for actual operation is not accessed by mistake.

(114) PDDBORGUAP={YES|NO}

This environment variable specifies whether to execute a UAP on the original RDAREA during online processing in a replica RDAREA.

YES

Execute the UAP on the original RDAREA being held for online reorganization.

NO

Do not execute the UAP on the original RDAREA being held for online reorganization.

(115) PDSPACELVL={0|1|3}

This environment variable specifies the space conversion level for data storage, comparison, and search processing. Space conversion is not executed when a definition SQL is executed.

0

Do not convert spaces.

1

Convert spaces in data for literals, embedded variables, and ? parameters of data manipulation SQL as follows:

- If a character string literal is determined to be a national character string literal, two single-byte space bytes are converted to a double-byte space character. If one single-byte space character appears alone, it is not converted.
- For a mixed character string literal, one double-byte space character is converted to two single-byte space characters.
- During data storage to a national character string-type column or comparison with a national character string-type value expression, two single-byte space bytes in embedded variables and ? parameters are converted to one double-byte space character. If one single-byte space character appears alone, it is not converted.
- During data storage to a mixed character string-type column or comparison with a mixed character string-type value expression, double-byte space

characters in embedded variables and `?` parameters are converted to two single-byte space characters.

`3`

In addition to the conversions for space conversion level 1, convert each double-byte space character to two single-byte space characters when data in a national character string-type value expression is searched.

Relationship to the system definition

If this environment variable is omitted, the HiRDB system uses the value that was specified in the `pd_space_level` operand of the system common definition.

Notes

1. If the space conversion level is changed, the UAP results obtained before and after the change may be different. To obtain the same UAP results, do not change the space conversion level.
2. If space conversion level 3 is specified and the data is sorted, the expected results may not be obtained because HiRDB applies space conversion to the sort results.
3. When data is stored to a cluster key column, a unique error may occur as a result of space conversion. If this occurs, store the data without applying space conversion, or make all spaces in the existing database uniform (apply space conversion with the database reorganization utility).
4. Space conversion of a national character string is executed in two-byte units from the beginning of the string.
5. If space conversion level `1` or `3` is specified and the UAP uses the hash function for table partitioning to determine the storage RDAREA of a hash-partitioned table, the space conversion level must also be specified as an argument of the hash function for table partitioning. Otherwise, the results of the hash function for table partitioning may become invalid. For details about the hash function for table partitioning, see *H.1 Hash function for table partitioning*.
6. If space conversion level `1` or `3` is specified and the UAP executes key range partitioning on a key range-partitioned table that has a national character string-type or mixed character string-type column in the partitioning key, the partitioning key value must be converted with the space conversion function. Otherwise, the results of key range partitioning may become invalid. For details about the space conversion function, see *H.2 Space conversion function*.

(116) PDCLTRDNODE=XDM/RD-E2-database-identifier

~ <identifier>

This environment variable specifies the identifier of the XDM/RD E2 database to be connected when the XDM/RD E2 connection facility is used. The database identifier refers to the RD node name specified in the XDM subsystem definitions.

(117) PDTP1SERVICE={YES|NO}

This environment variable specifies whether OpenTP1 service names are to be reported to XDM/RD E2 when the XDM/RD E2 connection facility is used.

`PDTP1SERVICE` cannot be specified if HiRDB client library `cltdll.dll` is being used in the Windows edition. This environment variable can be specified if the HiRDB client is relinked to another HiRDB client library (for example, `pdcltm32.dll`).

`YES`

Report OpenTP1 service names to XDM/RD E2.

When OpenTP1 service names are reported to XDM/RD E2, the XDM/RD E2 statistical information can be analyzed for each service. This function is supported only if the XDM/RD E2 version is 09-01 or later.

When OpenTP1 is not used, or if the service is not an OpenTP1 service (for example, if the service is SUP), the service name is not reported even when `YES` is specified.

`NO`

Do not report OpenTP1 service names.

(118) PDRDCLTCODE={SJIS|UTF-8}

This environment variable is valid for Windows clients. For UNIX clients, this environment variable is invalid even when specified.

This environment variable specifies the character code classification used by the client when the XDM/RD E2 connection facility is used.

`SJIS`

Use shift JIS kanji codes.

`UTF-8`

Use Unicode (UTF-8) codes. When UTF-8 is specified, specify `NOUSE` in the `PDCLTCNVMODE` client environment definition, or omit `PDCLTCNVMODE`.

Rules when UTF-8 is specified

1. Unicode (UTF-8) codes can be used in input/output data handled by embedded variables and data handled by the `?` parameter.
2. Only ASCII codes can be specified in SQL statements specified in a UAP. To specify a non-ASCII character (kanji, single-byte katakana, or Gaiji character) in an SQL statement, use the `PREPARE` or `EXECUTE IMMEDIATE`

statement and specify the SQL statement in an embedded variable.

3. Error messages returned from XDM/RD E2 and stored in the SQL Communications Area, column names stored in the Column Name Descriptor Area (SQLCNDA), and data type names stored in the Type Name Descriptor Area (SQLTNDA) use Unicode (UTF-8) codes. For this reason, if characters other than ASCII codes are contained in these values and are output as Shift JIS kanji codes, they may not be displayed correctly.
4. When the XDM/RD E2 side converts character codes from Unicode (UTF-8) to EBCDIK or KEIS, or EBCDIK or KEIS to Unicode (UTF-8), the data length may be changed. Therefore, pay attention to the definition length of embedded variables.

(119) PDCNSTRNTNAME={LEADING|TRAILING}

This environment variable specifies the position of the constraint name definition when a referential or check constraint is defined.

`LEADING`

Specify the constraint name definition before the constraint definition.

`TRAILING`

Specify the constraint name definition after the constraint definition.

Relationship to the system definition

If this environment variable is omitted, the value of the `pd_constraint_name` operand in the system definition is assumed.

(120) PDBESCONHOLD={YES|NO}

This environment variable can be specified when HiRDB/Parallel Server is used.

This environment variable specifies whether the BES connection holding facility is to be used. For details about the BES connection holding facility, see the *HiRDB Version 9 System Operation Guide*.

`YES`

Use the BES connection holding facility.

`NO`

Do not use the BES connection holding facility.

Relationship to the system definition

If this environment variable is omitted, the value of the `pd_bes_connection_hold` operand in the system definition is assumed.

(121) PDBESCONHTI=BES-connection-holding-period

~ <unsigned integer> ((0-3600)) (seconds)

This environment variable specifies the BES connection holding period when the BES connection holding facility is used.

When the BES connection holding facility is used, the back-end server monitors the time elapsed from when a transaction ends until the next transaction is executed. If the time until the next transaction is executed falls within the `PDBESCONHTI` specification value, the back-end server continues the BES connection holding facility. If the time exceeds the `PDBESCONHTI` specification value, the back-end server disconnects the connection with the front-end-server.

Notes

1. If `0` is specified, the back-end server does not monitor the time. The connection between the front-end server and the back-end server is disconnected only when the connection between the front-end server and the client is disconnected, such as when the `DISCONNECT` (`xa_close` when the XA library is used) SQL statement is executed or the time specified by the `PDCWAITTIME` client environment definition is exceeded.
2. `PDBESCONHTI` becomes valid when `YES` is specified in `PDBESCONHOLD`.

(122) PDODBSTATCACHE={<u>0</u>|1}

This environment variable specifies whether the column information or index information collected the first time an ODBC function (`SQLColumns()` or `SQLStatistics()`) is issued is to be cached.

`0`

Do not cache the information.

Column information or index information is collected by accessing the server each time the `SQLColumns()` or `SQLStatistics()` function is called.

`1`

Cache the column information or index information collected the first time the function is called.

However, the cache is not refreshed while the server is connected. Thus, if the table definition is altered while the server is connected, column information or index information that is different from the actual definition is returned. Therefore, the server connection must be terminated first.

Benefits

When the `SQLColumns()` and `SQLStatistics()` functions are called repeatedly with the same parameters, the number of communications with the server can be reduced by returning the retrieval results stored in the cache to the UAP.

Notes

To determine whether specifying this option will be effective, collect an ODBC trace and investigate whether the `SQLColumns()` or `SQLStatistics()` function is issued repeatedly with the same parameters during the same connection.

The number of rows that can be cached is x:

`SQLColumns()`

Approximately 60,000/(50 + table-owner-name-length + table-name-length + column-name-length + comment-length) rows

`SQLStatistics()`

Approximately 60,000/(50 + table-owner-name-length + table-name-length + index-name-length + column-name-length) rows

(123) PDODBESCAPE={0|<u>1</u>}

This environment variable specifies whether the `&` ESCAPE character is to be specified for the pattern character in a retrieval using a cataloging ODBC function (`SQLTables()`, `SQLColumns()`, etc.).

`0`

Do not specify the `&` ESCAPE character for the pattern character.

`1`

Specify the `&` ESCAPE character for the pattern character.

Notes

1. If the column attribute of the dictionary table is `CHAR` (`dictionary datatype mchar nouse` specified by the database initialization utility) and a double-byte character containing code `0x26` is used in the table name and the column name, `0` should specified in this option. If `1` is specified and the HiRDB system is accessed through ODBC, some tables and columns may not be recognized.
2. If an underscore (_) is used in the identifier of a table name, `1` should be specified in this option. If `0` is specified, some ODBC-compatible software programs may not be able to access the identifier that uses the underscore.

(124) PDGDATAOPT={YES|<u>NO</u>}

This environment variable specifies whether the `SQLGetData` function of the ODBC functions is to repeatedly retrieve data for columns from which data has already been retrieved.

Normally, when data retrieval is repeated for a column after data has already been retrieved from that column, SQL_NO_DATA is returned as the return value.

YES

The SQLGetData function can retrieve data repeatedly for columns from which data has already been retrieved.

NO

When the SQLGetData function goes to retrieve data for columns for which data has already been retrieved, SQL_NO_DATA is returned as the return value.

Application standard

This environment variable is specified when data is to be retrieved repeatedly for the same column. For example, this environment variable is specified when a host UAP that expects SQL_SUCCESS for repeated data retrieval executions is used.

Note

When Internet Banking Server is used, set PDGDATAOPT=YES in the HiRDB.ini file of the HiRDB client. If this specification is not set, the following problem may occur after the customer information management utility or transaction history management utility of Internet Banking Server is used to log into HiRDB. After a function selection key, such as **Register Customer**, **Update Customer Information**, or **Reference Customer Information** is pressed, screen operations other than the **Return** button may become disabled.

(125) PDODBLOCATOR={YES|NO}

This environment variable specifies whether the locator facility is to be used to partition and retrieve data when a database access tool is used to retrieve BLOB-type or BINARY-type column data. The database access tools are the ODBC driver, the OLE DB provider, and the HiRDB.NET data provider.

YES

Use the locator facility to partition and retrieve data when a database access tool is used to retrieve BLOB-type or BINARY-type column data.

NO

Do not use the locator facility when a database access tool is used to retrieve BLOB-type or BINARY-type column data.

Application standard

If NO is specified (NO is also the assumed value when this environment variable is omitted), the database access tool allocates a data reception area that has the defined length of the column. The HiRDB client also requires a data reception area that has the defined length of the column.

Since a memory shortage may occur during execution if the defined length of the column is large, specify YES to avoid running out of memory. Note that when YES is specified, the number of communications with the HiRDB server increases by the number of partition acquisitions.

(126) PDODBSPLITSIZE=partition-acquisition-size

~ <unsigned integer> ((4-2097152)) <<100>> (kilobytes)

This environment variable specifies the partition acquisition size when PDODBLOCATOR=YES is specified.

Specification value guideline

Consider the distribution of the actual data length, and specify a value that reduces the number of partition acquisitions but does not trigger a memory shortage.

(127) PDODBCWRNSKIP={YES|NO}

This environment variable specifies whether warnings should be skipped for ODBC connections. For non-ODBC connections, this environment variable is ignored, if specified.

YES

The ODBC driver returns SQL_SUCCESS as the SQLFetch() return value even if SQLFetch() processing is prolonged and SQLWARN is set.

NO

The ODBC driver returns SQL_SUCCESS_WITH_INFO as the SQLFetch() return value if SQLFetch() processing is prolonged and SQLWARN is set.

Application standard

When SQLWARN is set in the SQL Communications Area of HiRDB during retrieval processing, the ODBC driver returns SQL_SUCCESS_WITH_INFO as the SQLFetch() return value. However, depending on the higher-level application# that calls the ODBC driver, retrieval processing may be terminated by the SQL_SUCCESS_WITH_INFO return value. If YES is specified in this environment variable, SQL_SUCCESS is returned as the return value even if SQLWARN is set to the SQL Communications Area during retrieval processing, and retrieval processing can continue.

#: For example, if ADO.Net is used and connected to HiRDB through an ODBC connection, retrieval processing may be terminated by the SQL_SUCCESS_WITH_INFO return value.

(128) PDJETCOMPATIBLE={YES|NO}

This environment variable specifies whether the ODBC 3.5 driver is to be operated in a mode compatible with Microsoft Access, rather than on the basis of the ODBC 3.5 specification.

`YES`

The ODBC 3.5 driver is to operate in a mode compatible with Microsoft Access.

`NO`

The ODBC 3.5 driver is to operate as specified.

Application standard

Specify this environment variable when you use Microsoft Access to access HiRDB. When this variable is not specified, `#Delete` may be displayed as the search result, or inserted data may be converted incorrectly. If such events occur with other Microsoft products or interfaces, you might be able to prevent them by specifying this environment variable.

(129) PDPLGIXMK={YES|NO}

This environment variable specifies whether delayed batch creation of plug-in indexes is to be used. For details about delayed batch creation of plug-in indexes, see the *HiRDB Version 9 System Operation Guide*.

`YES`

Use delayed batch creation of plug-in indexes.

`NO`

Do not use delayed batch creation of plug-in indexes.

(130) PDPLUGINNSUB

For details, see the manual for the target plug-in.

(131) PDPLGPFSZ=initial-size-of-delayed-batch-creation-index-information-file

~ <unsigned integer> ((1-1048574000)) <<8192>> (kilobytes)

This environment variable specifies the initial size of the index information file for delayed batch creation of plug-in indexes. This specification is effective when the index information file is to be created in the HiRDB file system area.

When this environment variable is specified, `PDPLGIXMK=YES` should also be specified.

(132) PDPLGPFSZEXP=extension-value-of-delayed-batch-creation-index-information-file

~ <unsigned integer> ((1-1048573000)) <<8192>> (kilobytes)

This environment variable specifies the extension size of the index information file for delayed batch creation of plug-in indexes. When the index information file becomes full, the file is extended by the value specified in this environment variable. This specification is effective when the index information file is to be created in the HiRDB file system area.

When this environment variable is specified, `PDPLGIXMK=YES` should also be specified.

(133) PDJDBFILEDIR=exception-trace-log-file-storage-directory

~<path name> PDCLTPATH setting

This environment variable specifies the Exception trace log file storage directory with the Type4 JDBC driver. To specify the file storage directory, specify the absolute path of the directory (maximum of 256 bytes). This environment variable can be specified only when the Type4 JDBC driver is used.

For details about the Exception trace log, see *18.15 Exception trace log*. For other details, see system property `HiRDB_for_Java_FileDIR` in *18.15.1(2)(b) Setting system properties*.

(134) PDJDBFILEOUTNUM=number-of-outputs-to-exception-trace-log-file

~<unsigned integer>((1-50)) (5)

This environment variable specifies the number of outputs to the Exception trace log file with the Type4 JDBC driver. This environment variable can be specified only when the Type4 JDBC driver is used.

For details about the Exception trace log, see *18.15 Exception trace log*. For other details, see system property `HiRDB_for_Java_FileOutNUM` in *18.15.1(2)(b) Setting system properties*.

(135) PDJDBONMEMNUM=number-of-acquired-information-items-in-exception-trace-log-memory

~<unsigned integer>((500-10000)) (1000)

This environment variable specifies the number of acquired information items in the Exception trace log memory. This environment variable can be specified only when the Type4 JDBC driver is used.

For details about the Exception trace log, see *18.15 Exception trace log*. For other details, see system property `HiRDB_for_Java_OnMemNUM` in *18.15.1(2)(b) Setting system properties*.

(136) PDJDBTRACELEVEL=trace-acquisition-level-of-exception-trace-log

~<unsigned integer>((0~5)) (1)

This environment variable specifies the trace acquisition level of the Exception trace log with the Type4 JDBC driver. If 0 is specified, the Exception trace log is not acquired. This environment variable can be specified only when the Type4 JDBC driver is used.

For details about the Exception trace log, see *18.15 Exception trace log*. For other details, see system property HiRDB_for_Java_TraceLevel in *18.15.1(2)(b) Setting system properties*.

(137) PDXDSHOST=XDS-host-name[,secondary-XDS-host-name]

~ <identifier> ((maximum 511 bytes)) <<value of PDFESHOST>>

This client environment definition is applicable only to XDS clients.

This environment variable specifies the host name of the XDS to be connected. You can also specify a domain name or IP address for the host name. Express an IP address as decimal numbers using a period as the delimiter between bytes.

The value to be specified depends on whether the IP address is to be inherited after the systems have been switched when the dual memory database function is used.

For system switchover that inherits IP addresses

Specify the primary host name in PDXDSHOST in the client environment definition.

For system switchover that does not inherit IP addresses

Specify both primary and secondary host names in PDXDSHOST in the client environment definition. If you specify only the primary host name, you must change the primary host name in this operand after system switchover has occurred.

If you omit this environment variable, the HiRDB client uses the host name specified in PDFESHOST. Note that the port number of the front-end server specified in PDFESHOST is not used. The name of the host to be connected depends on the specification of PDXDSHOST and PDFESHOST. The following table shows how the connection-target host name is determined.

Table 6-36: How to determine the connection-target host name

PDXDSHOST	PDFESHOST	Connection-target host name
Y	Y	PDXDSHOST value
	N	
N	Y	PDFESHOST value
	N	No host name specified

Legend:

Y: Specified

N: Omitted

(138) PDXDSPORT=XDS-port-number

~ <unsigned integer> ((5001 to 65535)) <<value of PDSERVICEPORT>>

This client environment definition is applicable only to XDS clients.

This environment variable specifies the port number of the XDS to be connected. This must be the port number specified in the -p option in the pdqmyrecvdef operand in the XDS server definition. For details about the pdqmyrecvdef operand, see the *HiRDB Version 9 Memory Database Installation and Operation Guide*.

If you omit this environment variable, the HiRDB client uses the first high-speed connection port number specified in PDSERVICEPORT.

The connection-target port number depends on the port number specification in PDXDSPORT, PDSERVICEPORT, and PDFESHOST. The following table shows how the connection-target port number is determined.

Table 6-37: How to determine the connection-target port number

PDXDSPORT	PDSERVICEPORT	PDFESHOST	Connection-target port number
Y	Y	Y	PDXDSPORT value
		N	
	N	Y	
		N	
N	Y	Y	PDSERVICEPORT value
		N	
	N	Y	No port number specified
		N	

Legend:

Y: Specified

N: Omitted

(139) PDXVWOPT=output-of-access-path

~ <unsigned integer> ((0 to 1)) <<0>>

This client environment definition is applicable only to XDS clients.

This environment variable specifies whether the access path of the SQL statement used to access a table to be expanded to the memory database is to be output.

`0`

Do not output the access path.

`1`

Output the access path of a preprocessed SQL statement if the SQL statement was preprocessed in XDS. Note that if the SQL statement was found in the SQL pool, the access path will not be output.

6.6.5 Environment variables and connection types for HiRDB servers

The following table shows the relationships among environment variables and connection types for connecting with the HiRDB server.

Table 6-38: Relationships between environment variables and connection types

Environment variable	HiRDB/Single Server		HiRDB/Parallel Server				
			Single front-end server		Multiple front-end servers		
	Normal	High-speed	Normal	High-speed	Normal	Connection with specific front-end server	
						FES-host direct	High-speed
`PDHOST`	S	S	S	S	S	S	S
`PDFESHOST`	--	--	--	S#	--	S	S#
`PDNAMEPORT`	S	S	S	S	S	S	S
`PDSERVICEPORT`	--	S	--	S#	--	--	S#
`PDSERVICEGRP`	--	S	--	S#	--	S	S#
`PDFESGRP`	--	--	--	S#	--	--	S#
`PDSRVTYPE`	--	△	--	△	--	--	△

S: Must be specified.

△: Must be specified if the HiRDB server is the Linux or Windows edition.

--: Does not have to be specified.

Notes

1. The connection format in which all required environment variables are specified is selected in the priority order of high-speed connection, FES-host direct connection, and normal connection. Unnecessary environment variables are not used.
2. The following relationships apply to the connection time to the HiRDB server and the number of TCP ports used during connection:

   Normal connection time > FES-host direct connection time > high-speed connection time

A high-speed connection is recommended if you want to shorten the connection time. A normal connection is recommended in order to use the connected front-end server efficiently.

If you want to reduce the number of TCP ports required when a normal connection is used, specify `1` in `PDTCPCONOPT` in the client environment definition. For details about `PDTCPCONOPT`, see *6.6.4 Environment definition information*.

#

There are two ways to specify high-speed connection:

- Specify `PDFESHOST`, `PDSERVICEGRP`, and `PDSERVICEPORT`
- Specify `PDFESGRP`

If you specify only one host name in `PDFESHOST` or `PDFESGRP`, the two methods above are treated as being the same. If you specify multiple host names, the specification method depends on the conditions shown in the following table:

No.	Condition for specifying multiple host names		Environment variables to be specified
	System switchover facility	Number of FESs	
1	The system switchover facility is used on the unit where the FES is located and the IP address of the primary system used for connection from the client is different from the IP address of the secondary system.	1	`PDFESHOST`, `PDSERVICEGRP`, and `PDSERVICEPORT`
2		Multiple	`PDFESHOST`, `PDSERVICEGRP`, and `PDSERVICEPORT`
3	Other than the above	Multiple	`PDFESGRP`

In No. 2, there are multiple FESs (multiple front-end servers), but you must specify PDFESHOST, PDSERVICEGRP, and PDSERVICEPORT and use only one FES as the connection target. If multiple FESs are specified in PDFESGRP and an error (KFPA11932-E) occurs in the running system due to too many connected users, the following problems might occur:

- An unnecessary attempt is made to connect to the standby system.
- If a connection error occurs in all FES groups and the FES group for which the last connection attempt was made is a standby system, information about errors only in the standby system is returned to the UAP, making it impossible to detect an error that might be caused by too many connected users in the running system.

6.7 Registering an environment variable group

The environment variables of a client can be registered as a group. When the environment variables of each client are registered, the environment variables can be changed for each connection. This operation is therefore convenient when the environment variables must be changed for each connection.

The environment variables are registered to a normal file in the UNIX environment and to a registry or a file in the Windows environment. Information about the registered environment variables is obtained during connection to the HiRDB server.

When an open character string is specified while a UAP that uses an X/Open-compliant API under OLTP is used as the client, the environment variables of the environment variables group specified in the open character string have priority over environment variables specified according to *6.6.2 Specifications for using a UAP under OLTP as the client*. For details about open character strings, see the *HiRDB Version 9 Installation and Design Guide*.

6.7.1 Registering an environment variable group in a UNIX environment

When registering environment variables to a normal file, use the following rules:

- Specify one environment variable per line.
- Specify the environment variable with the following format:

  client-environment-variable=specification-value
- When specifying a comment, specify a slash and asterisk (`/*`) before the comment and an asterisk and slash (`*/`) after the comment. Comments cannot be embedded. Do not use line breaks in comments.
- If the same environment variable is specified more than once, the final specification becomes effective.
- A specified value that includes a space must be enclosed in double quotation marks (`"`). If such a value is not enclosed in double quotation marks, the space will be deleted.
- `[HIRDB]` can be specified in the first line.

An example of how environment variables are set to a normal file (`/HiRDB_P/Client/HiRDB.ini`) is shown as follows.

Example

```
[HIRDB]
PDCLTPATH=trace-file-storage-directory
PDHOST=system-manager-host-name
```

PDUSER="*authorization-identifier*"/"*password*"
PDNAMEPORT=*port-number-of-system-manager-process*
PDCLTAPNAME=*identification-name-of-UAP-to-be-executed*

Note

With this method, client environment definitions that begin with PDJDB become invalid except in the case of UAPs that use the Type4 JDBC driver.

6.7.2 Registering an environment variable group in a Windows environment (registry registration)

Use the tool for registering HiRDB client environment variables to register environment variables in the registry.

To use the tool for registering HiRDB client environment variables, execute *xxxx*\UTL\pdcltadm.exe (*xxxx* is %PDDIR%\client in the HiRDB server and the HiRDB client installation directory in the HiRDB client).

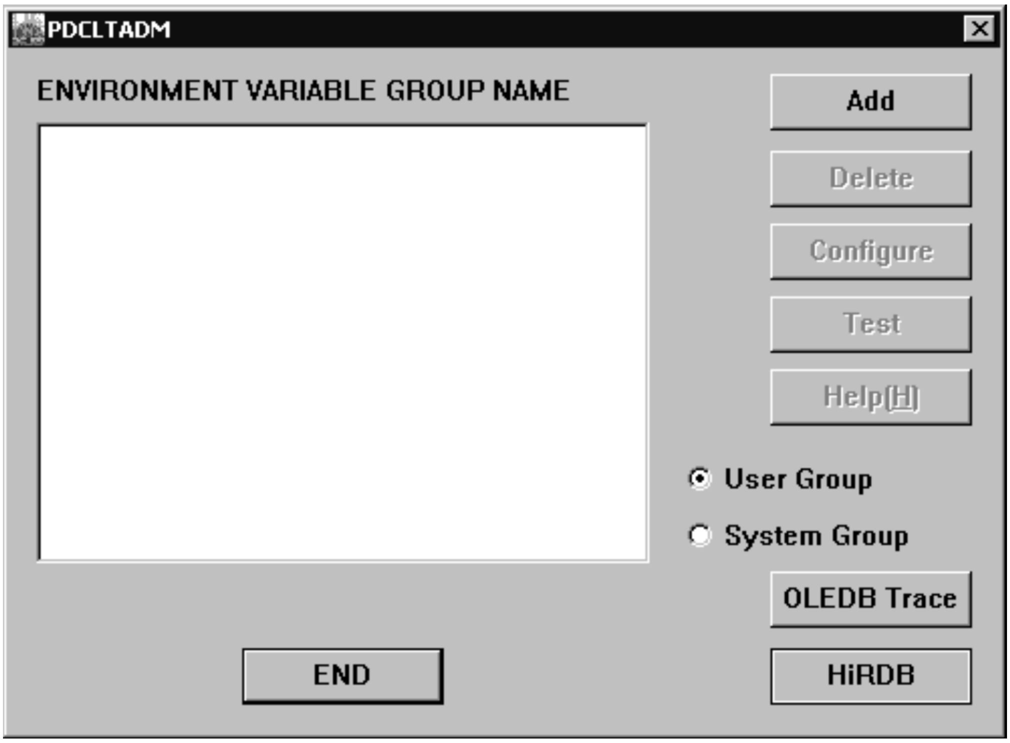
The rest of this section describes procedures for registering environment variables in the registry with the tool for registering HiRDB client environment variables.

In an OLE DB connection, the environment variables registered with the tool for registering HiRDB client environment variables have priority over the user environment variables and the specifications in HIRDB.INI.

Client environment definitions set using this method become invalid when the Type4 JDBC driver is used. Client environment definitions that begin with PDJDB become invalid if this method is used.

(1) Starting the tool for registering HiRDB client environment variables

Execute *xxxx*\UTL\pdcltadm.exe. The **Tool for Registering HiRDB Client Variables** dialog box is displayed.



● User Group

○ System Group

Explanation

User Group

Select this item to add, delete, or modify an environment variable group for a user. This information is registered in `HKEY_CURRENT_USER`.

System Group

Select this item to add, delete, or modify an environment variable group for a computer. This information is registered in `HKEY_LOCAL_MACHINE`.

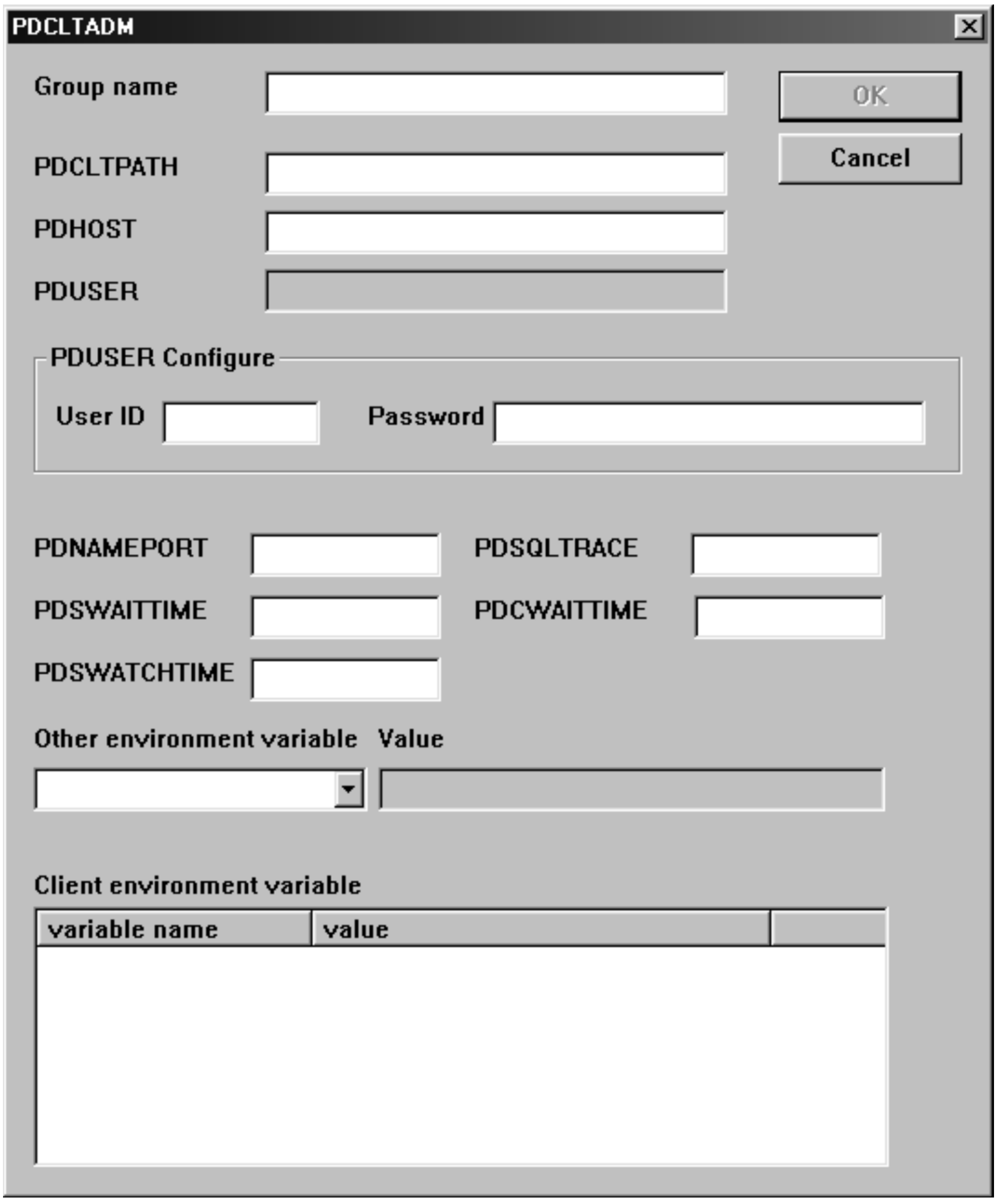
Select either **User Group** or **System Group**, and click the **Add** button.

Note

When a UAP that uses an X/Open-compliant API under OLTP is used as a client, select **System Group** so that you can use the tool for registering HiRDB client environment variables to register the environment variables group whose name is specified in the open character string. For details about open character strings, see the *HiRDB Version 9 Installation and Design Guide*.

(2) Registering an environment variable group

The **HiRDB Client Environment Variable Setup** dialog box is displayed.



Explanation

Group Name

Specify the group name with up to 30 characters.

Environment variable fields

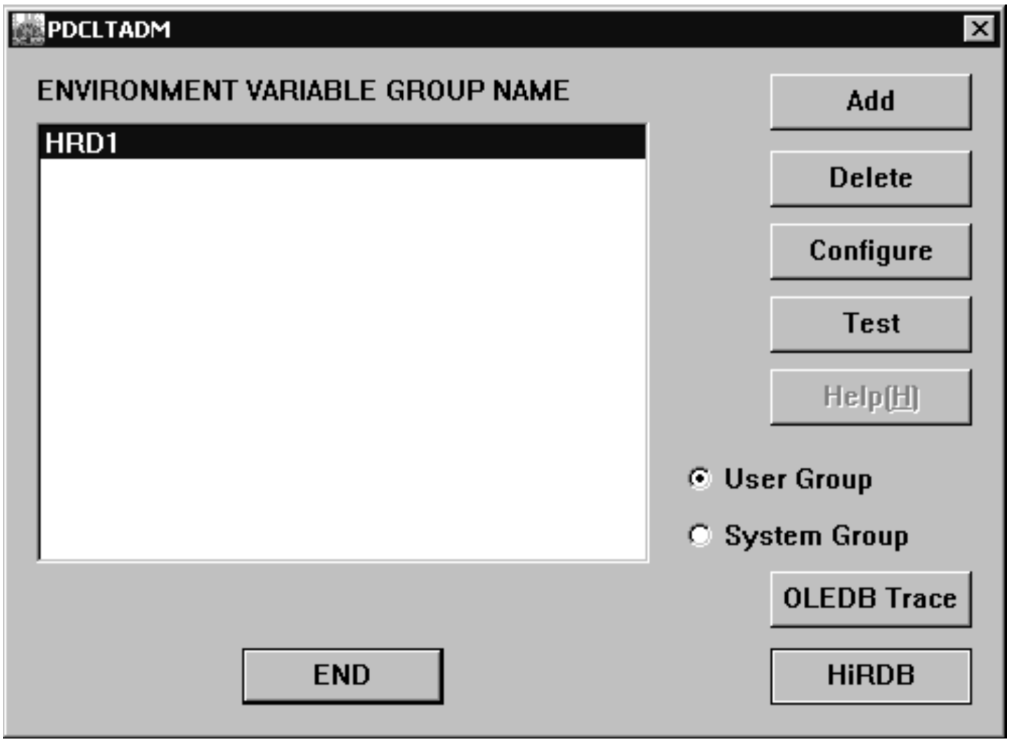
Specify a setting for each environment variable. For details about each environment variable, see *6.6.4 Environment definition information*.

After completing the setup, click the **OK** button. When the **OK** button is clicked, the client environment variable settings are registered in the registry, and the **Tool for Registering HiRDB Client Variables** dialog box is displayed again.

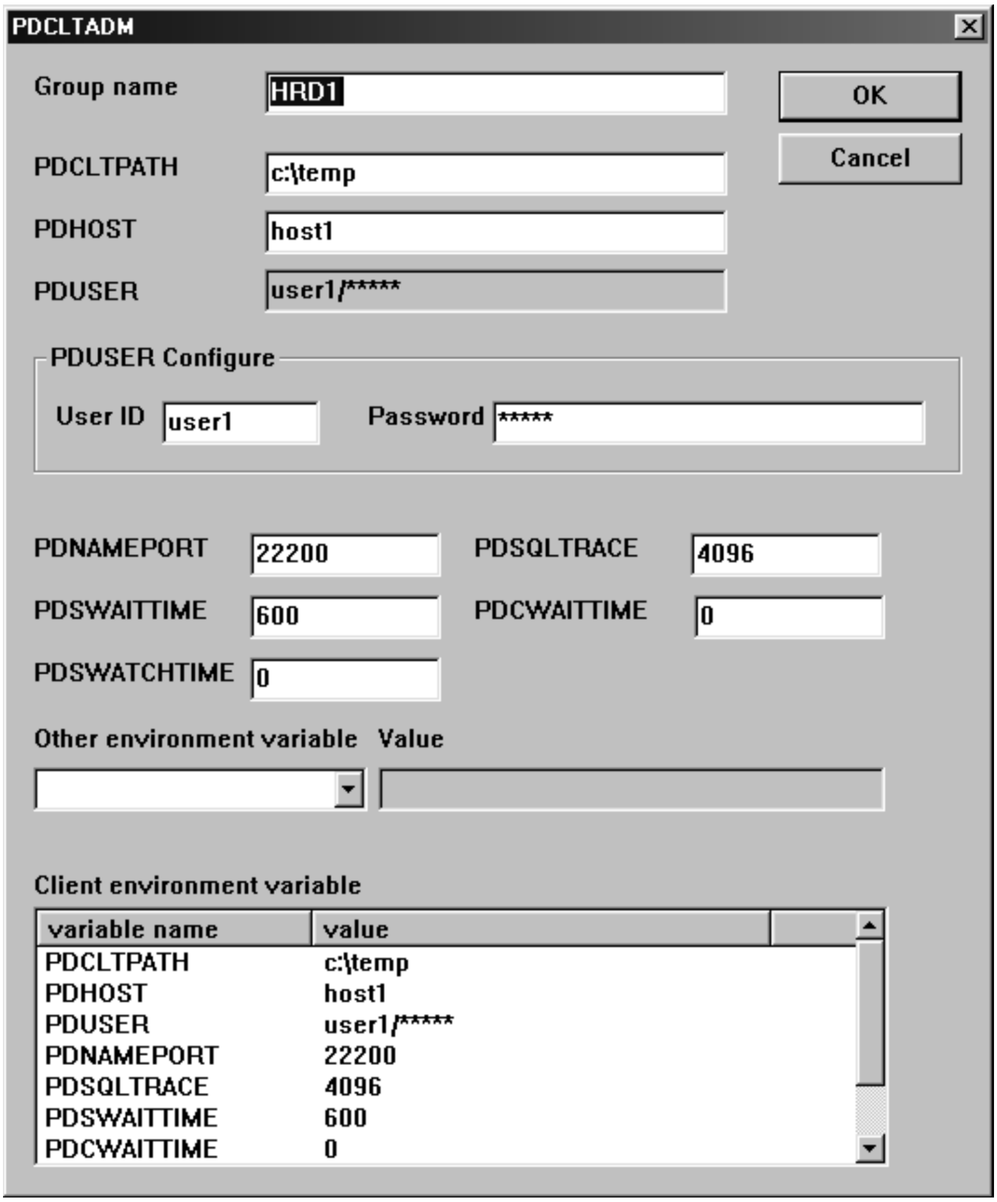
Click the **Cancel** button to cancel the client environment variable settings and return to the **Tool for Registering HiRDB Client Variables** dialog box.

(3) Changing the settings of an environment variable group

After one or more environment variable groups are registered, a list of the registered environment variable group names is displayed, as shown in the next dialog box.



When an environment variable group name in the list is selected, the **Delete**, **Configure**, and **Test** buttons become enabled and can be clicked. Click the **Configure** button or double-click an environment variable group name in the list. The following dialog box is displayed:



Group name: HRD1

PDCLTPATH: c:\temp

PDHOST: host1

PDUSER: user1/*****

User ID: user1

Password: *****

PDNAMEPORT: 22200

PDSQLTRACE: 4096

PDSWAITTIME: 600

PDCWAITTIME: 0

PDSWATCHTIME: 0

Client environment variable

variable name	value
PDCLTPATH	c:\temp
PDHOST	host1
PDUSER	user1/*****
PDNAMEPORT	22200
PDSQLTRACE	4096
PDSWAITTIME	600
PDCWAITTIME	0

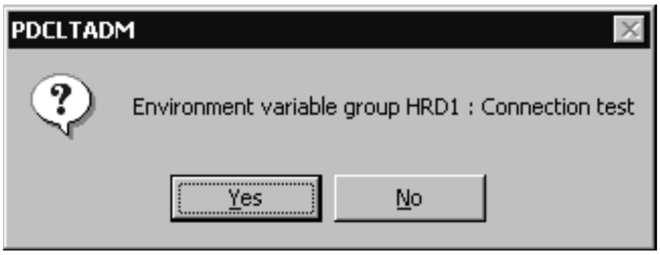
Change the environment variable settings, and click the **OK** button. When the **OK** button is clicked, the new client environment variable settings are registered in the registry, and the **Tool for Registering HiRDB Client Variables** dialog box is displayed again.

Click the **Cancel** button to cancel the client environment variable settings and return to the **Tool for Registering HiRDB Client Variables** dialog box.

(4) Checking the HiRDB connection with a registered environment variable group

You can use a registered environment variable group to check whether a connection to HiRDB can be established for that group.

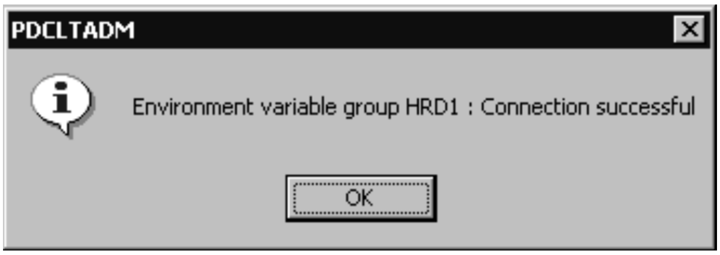
From the environment variable group name list in the **Tool for Registering HiRDB Client Variables** dialog box, select the environment variable group name for which connection to HiRDB is to be checked, and click the **Connect** button. The following dialog box is displayed:


PDCLTADM

Environment variable group HRD1 : Connection test

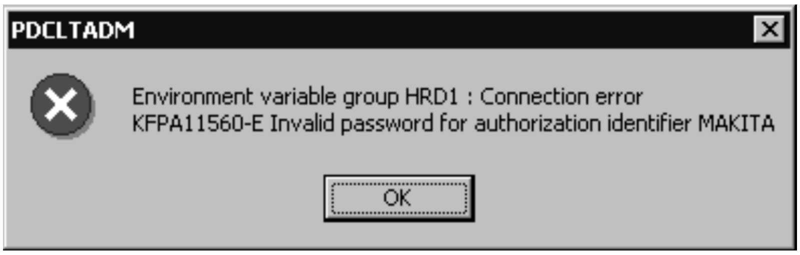
Click **Yes**.

If connection to HiRDB is successful, the following dialog box is displayed:


PDCLTADM

Environment variable group HRD1 : Connection successful

If connection to HiRDB fails, the following dialog box is displayed. If the error is due to the contents of an environment variable, change the settings of the environment variable group.

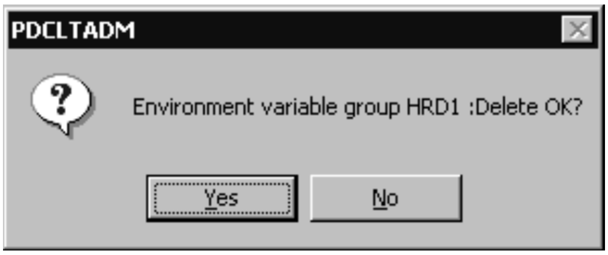

PDCLTADM

Environment variable group HRD1 : Connection error
KFPA11560-E Invalid password for authorization identifier MAKITA

(5) Deleting an environment variable group

From the environment variable group name list in the **Tool for Registering HiRDB Client Variables** dialog box, select the environment variable group name to be deleted, and click the **Delete** button.

The following dialog box is displayed:



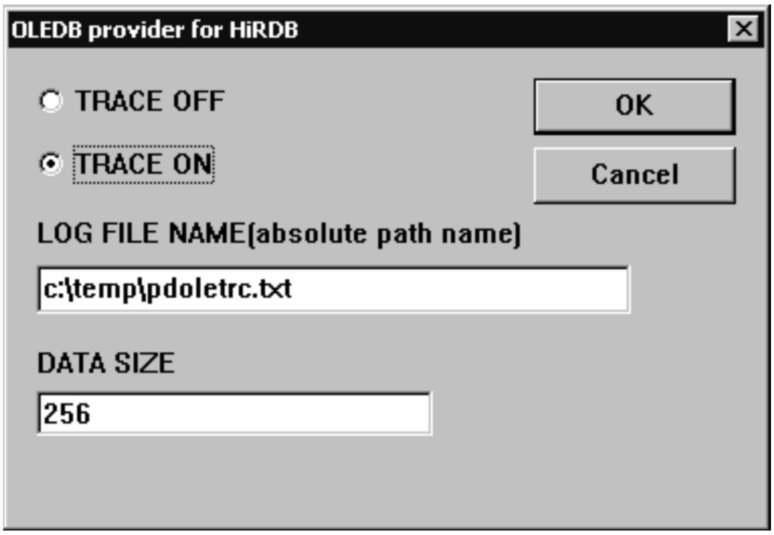
PDCLTADM

Environment variable group HRD1 :Delete OK?

(6) Setting an OLE DB provider trace

The OLE DB provider trace is for troubleshooting only. Do not set this trace for any other investigation. Note that when a trace is performed, the performance of other operations may drop dramatically.

To set an OLE DB provider trace when the HiRDB client is connected to an OLE DB, open the **Tool for Registering HiRDB Client Variables** dialog box, and click the **OLE DB Trace** button. The following dialog box is displayed:



OLEDB provider for HiRDB

○ TRACE OFF

● TRACE ON

LOG FILE NAME(absolute path name)

c:\temp\pdoletrc.txt

DATA SIZE

256

To perform a trace, select **TRACE ON** and then click the **OK** button. Note that the trace continues until you select **TRACE OFF** and click the **OK** button.

For the log file name, always specify the absolute path name of the file.

For the data size, specify the output size, in bytes, of the `void*`-type data dump.

(a) Environment variables that become invalid when the multi-connection facility is used

When the multi-connection facility is used, the following environment variables cannot be set for each connection destination. These environment variables become invalid even if they are registered to a normal file or registry and specified in each connection destination.

- `HiRDB_PDHOST`
- `HiRDB_PDNAMEPORT`
- `HiRDB_PDTMID`
- `HiRDB_PDXAMODE`
- `PDTMID`
- `PDXAMODE`
- `PDTXACANUM`

6.7.3 Registering an environment variable group in a Windows environment (file registration)

The client environment definitions can be set to a file and the environment variable definitions can then be obtained from the file during HiRDB server connection.

To register an environment variable group to the file, you must specify `[HIRDB]` in the first line.

An example of setting an environment variable group to a file (`c:\HiRDB_P\Client\HiRDB.ini`) is shown below.

Example

```
[HIRDB]
PDCLTPATH=trace-file-storage-directory
PDHOST=system-manager-host-name
PDUSER="authorization-identifier"/"password"
PDNAMEPORT=port-number-of-system-manager-process
PDCLTAPNAME=identification-name-of-UAP-to-be-executed
```

Note

For a UAP using the Type4 JDBC driver, follow the rules described in *6.7.1 Registering an environment variable group in a UNIX environment*.

Chapter
7. UAP Creation

This chapter explains how to embed SQL statements in a UAP written in C, C++, COBOL, or OOCOBOL.

This chapter contains the following sections:

7.1 Overview
7.2 Writing a UAP in C
7.3 Writing a UAP in COBOL
7.4 Writing a UAP in C++
7.5 Writing a UAP in OOCOBOL
7.6 Creating a UAP in 64-bit mode

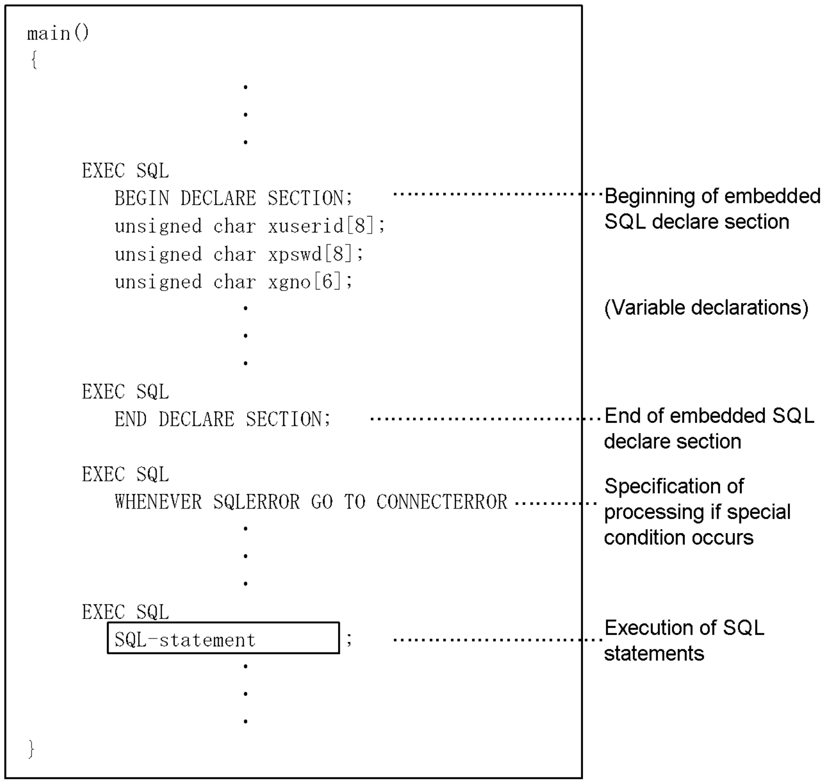
7.1 Overview

To create an embedded SQL UAP, embed SQL statements into a source program written in the C or COBOL language. This section explains the basic configuration of and rules for writing UAPs in which SQL statements will be embedded.

7.1.1 UAP basic configuration

The following is an example of the basic configuration of an embedded SQL UAP written in C.

Figure 7-1: Example of the basic configuration of an embedded SQL UAP



```
main()
{
                   .
                   .
                   .
     EXEC SQL
        BEGIN DECLARE SECTION;
        unsigned char xuserid[8];
        unsigned char xpswd[8];
        unsigned char xgno[6];
                   .
                   .
                   .
     EXEC SQL
        END DECLARE SECTION;
     EXEC SQL
        WHENEVER SQLERROR GO TO CONNECTERROR
                   .
                   .
                   .
     EXEC SQL
        SQL-statement     ;
                   .
                   .
                   .
}
```

Beginning of embedded SQL declare section
(Variable declarations)
End of embedded SQL declare section
Specification of processing if special condition occurs
Execution of SQL statements

7.1.2 UAP configuration elements

An embedded SQL UAP consists of the following four principal elements:

- Declaration of embedded variables and indicator variables
- Declaration of SQL Communications Areas

- Specification of operations to be performed when unexpected events occur
- SQL statements to be executed

(1) Declaration of embedded variables and indicator variables

Embedded variables and indicator variables to be used in the SQL statement must be declared. For details, see the *HiRDB Version 9 SQL Reference* manual.

(2) Declaration of SQL Communications Areas

Areas for receiving information (return codes) returned from HiRDB must be declared. The SQL Communications Areas need not be described in the UAP because they are expanded automatically within the source program when the UAP is preprocessed (for details, see *A. SQL Communications Area*).

(3) Specification of operations to be performed when unexpected events occur

`WHENEVER` statements should be specified to set the operations the UAP must perform for the various return codes returned by HiRDB after SQL statement execution.

Even when no `WHENEVER` statements are specified, it is possible to specify operations to be performed when unexpected events occur, providing that the return codes are identified directly after SQL statements execution. For details about how to specify the `WHENEVER` statement and about return code identification, see *3.6 SQL error identification and corrective measures*.

(4) SQL statement execution

The SQL statement to be executed must be specified. For details about the coding rules for C, see *7.2.1 Coding rules*. For details about the coding rules for COBOL, see *7.3.1 Coding rules*.

7.2 Writing a UAP in C

This section explains, by way of examples, the coding rules for embedding SQL statements in UAPs written in C.

7.2.1 Coding rules

When a UAP is created, the labelling rules, SQL coding rules, and SQL syntax rules must be followed.

(1) Labeling rules

Labels must be assigned according to the C language rules. These types of labels cannot be used:

- Labels that begin with uppercase `SQL`
- Labels that begin with lowercase `p_`
- Labels that begin with lowercase `pd`
- Labels that begin with uppercase `PD`

For naming embedded variables, indicator variables, and branching destination labels, the labeling and the C language rules must be followed.

(2) SQL coding rules

1. Each SQL statement must be preceded by the SQL prefix (`EXEC SQL`) and followed by the SQL terminator (`;`).

   Valid example:
   `EXEC SQL` *SQL-statement*`;`

2. The C language macro function cannot be used for an embedded SQL statement or any part of it.

   Invalid example:
   ```
   #define X USER.MEMBER
   EXEC SQL
     SELECT NAME INTO  MANNAME FROM X;
   ```

3. The underline indicates the invalid portion.

   SQL reserved words can be in uppercase letters, lowercase letters, or a mixture of both.

   Example 1:
   ```
   EXEC SQL
     SELECT MEM INTO  :NAME FROM TABLE;
   ```

Example 2:

```
exec sql
select MEM into  :NAME from TABLE;
```

Example 3:

```
exec SQL
  SELECT MEM Into  :NAME From TABLE;
```

4. One line each must be used for the SQL prefix, the embedded SQL start declaration, and the embedded SQL termination declaration. A space is used to separate the words making up each item.

   A line consists of a character string that begins with the character following the linefeed character and ends with the next linefeed character. The maximum length of a row in a UAP source program that can be preprocessed is 32,000 characters.

   Valid specification:

   ```
   EXEC SQL
     BEGIN DECLARE SECTION;
   ...
   EXEC SQL
     END DECLARE SECTION;

   EXEC SQL
     SELECT...  ;
   ```

   Invalid specification:

   ```
   EXEC SQL
     BEGIN
     DECLARE SECTION;
       :

   EXEC SQL
     END
     DECLARE SECTION;

   EXEC \
     SQL
     SELECT ...    ;
   ```

5. The embedded SQL declaration section must precede the SQL statements that use the embedded variables and indicator variables.

   Example:

   ```
   EXEC SQL
     BEGIN DECLARE SECTION;
   short SALES;
   EXEC SQL
     END DECLARE SECTION;
   ```

```
...
EXEC SQL
  SELECT PRICE INTO :SALES
    FROM TABLE;
```

6. The following rules apply to specifying embedded variables and indicator variables.

   - A declaration statement can span multiple lines. Multiple definition statements can also be described in a single declaration statement.

   Specification example:

```
short SALES,
      QUANTITY;
short SALES;     short QUANTITY;
```

   - The following table shows the items that can be described in an embedded SQL declaration section.

Table 7-1: Items that can be described in an embedded SQL declaration section

Described Item	Description within embedded declaration
Note	D
C language instruction statement	--
C language control statement	--
SQL statement	--
Embedded variable declaration	D
Indicator variable declaration	D

D: Can be described.

--: Cannot be described.

   - The same embedded variable or indicator variable cannot be repeated within the same source file.
   - Multiple embedded variables or indicator variables can be declared in a single declaration statement.

   Specification example:

```
short SALES, QUANTITY;                   1

short XSALES, XQUANTITY;                 2
```

   1: Declaration of embedded variables

2: Declaration of indicator variables

- For details about the data types that can be used in embedded variables, see *F. SQL Data Types and Data Descriptions*.

7. Embedded variables declared within a function become local variables; embedded variables declared outside a function become global variables.
8. Although embedded SQL statements can also be described in locations within a function block where C language instruction statements can be described, they cannot be described on the same lines as another SQL statement or statements written in C language.

Note

A label can be placed before an SQL prefix.

The following table shows the locations where SQL statements can be specified.

Table 7-2: Locations where SQL statements can be described

Description location within a line		SQL statement description
C language and instruction statement	Front	--
	Middle	--
	Back	--
C language control statement	Front	--
	Middle	--
	Back	--
Label	Front	--
	Back	D
Comments	Front	D
	Middle	--
	Back	D
SQL statement#	Front	--
	Middle	--
	Back	--

D: Can be described.

--: Cannot be described.

#: Must begin with an SQL prefix and end with an SQL terminator.

9. To include a Microsoft Foundation Class (MFC) header file (AFX*xxxxx*.H) in a UAP source program that uses HiRDB with the Visual C++ compiler, include the HiRDB header file after the MFC header file by using the following SQL statement:

   ```
   EXEC SQL INCLUDE HIRDB_HEADERS;
   ```

   - INCLUDE HIRDB_HEADERS includes the HiRDB header file that was automatically included at the beginning of the post source file at the specified location.
   - INCLUDE HIRDB_HEADERS can be used only with C and C++. It cannot be used with other languages.
   - INCLUDE HIRDB_HEADERS can only be used once in a UAP.
   - If INCLUDE HIRDB_HEADERS is not used, the HiRDB header file is included at the beginning of the post source file.

   The MFC header files provided by Visual C++ are sequentially related according to the order in which they are included. If the WINDOWS.H header file (header file used by HiRDB) is included first, an error may occur. In this case, use INCLUDE HIRDB_HEADERS.

   HiRDB uses the following Visual C++ header files:

   - WINDOWS.H
   - STRING.H

   An example of using INCLUDE HIRDB_HEADERS is shown as follows.

   ```
   #include <afx.h>
   EXEC SQL INCLUDE HIRDB_HEADERS ;
   ```

10. Comments (/*...*/) specified between the SQL prefix and the SQL terminator are deleted. However, SQL optimization specifications (/*>>...<<*/) are not deleted but instead treated as SQL statements. For details about comments and SQL optimization specifications in SQL statements, see the *HiRDB Version 9 SQL Reference* manual.

11. The backslash (\) symbol cannot be used to indicate row continuation.

12. When you use the -E option, the preprocessor declaration statement for the C compiler becomes effective. Consequently, you can use #ifdef to specify SQL statement switching and use macro literals to specify literals in the embedded SQL declare section. However, the following restrictions apply:

    - Preprocessor declaration statements cannot be specified between the SQL prefix and SQL terminator.

- Macros that change the column positions of the SQL prefix and SQL terminator cannot be specified.
- Macro definitions of the SQL prefix and SQL terminator cannot be specified.

13. When you use the `-E` option, you can use an embedded variable, as long as you declare the variable according to the C syntax rules, and the embedded variable corresponds to an SQL data type. This is allowed even if you do not declare the embedded variable in the embedded SQL declare section. If there is another variable with the same name, the effective scope of each variable is determined according to the C syntax rules. You can also use variables declared in an included header. However, the following restrictions apply:
    - Only the first 63 characters of the variable name are distinguished. The subsequent characters are not distinguished. Note that a universal character name (`\u`*xxxx* or `\U`*xxxxxxxx*) is counted as one character.
    - Nested structures cannot be used.
    - Declare statements cannot contain embedded variables that use an expression in a subscript.
    - `const`-type embedded variables can be used only as input variables.
    - `varchar` cannot be used for C language identifiers such as variable names and function names, regardless of whether upper or lower case is used.
14. When you use the `-E` option, you can declare a structure that has multiple embedded variables as members as an embedded variable. All members must have a format that corresponds to an SQL data type. A structure cannot contain another structure or a union. However, you can use a structure that corresponds to the variable-length character string type or the `BINARY` type.
15. When you use the `-E` option, you can declare a pointer as an embedded variable. The declaration format conforms to the C syntax rules. When using such an embedded variable in an SQL statement, specify the variable with the same format used for normal embedded variables. Do not add an asterisk in front of the embedded variable name.
16. When you use the `-E` option and specify a structure member explicitly as an embedded variable, include the structure name as a modifier. The format of the structure member specification becomes `:`*structure*`.`*member-name*. If you are using a pointer to the structure, include the pointer as a modifier. The format of the structure member specification becomes `:`*pointer*`->`*member-name*.
17. When you specify the `-E` option in Windows, the following restrictions apply:
    - Defining the same `typedef` name twice with `typedef` does not trigger a syntax error. However, there is also no check for determining whether the

defined contents are the same.

- Members of anonymous structures cannot be used as embedded variables.
- Declarators specified without a storage class or data type cannot be used as embedded variables.

18. When you specify the `-E` option, you cannot use the `COPY` statement.

19. A variable-length array cannot be used as an embedded variable.

20. If you specify the `-E` option, neither digraphs, such as `<:` and `<%`, nor trigraphs, such as `??(` and `??=`, can be specified. Even if you do not specify the `-E` option, you cannot specify trigraphs in SQL statements or the embedded SQL declare section. If you use trigraphs, they are treated as normal characters.

21. When you specify the `-E` option, you can use the reserved words listed below that are not stipulated in the C standard (C99):

Usage	Processing	Environment#1	Reserved words
Type qualifier or type specifier#2	Treated as a type qualifier or type specifier during syntax analysis	Windows	__int8,__int16,__int32, __int64,_int8,_int16, _int32,_int64
		UNIX	__volatile__, __builtin_va_list, __complex__, __signed__

Usage	Processing	Environment#1	Reserved words
Other	Ignored because the word is treated during syntax analysis as meaningless	Windows	__based,__cdecl, __export,__far, __fastcall, __forceinline, __inline,__near, __pascal,__ptr32, __ptr64,__stdcall, __unaligned,__w64, _based,_cdecl, _export,_far,_fastcall, _forceinline,_inline, _near,_pascal, _stdcall__declspec, __pragma, _declspec__try, __except, __finally__asm,_asm
		UNIX	__const,__const__, __extension__, __inline,__inline__, __restrict, __attribute__, __asm__

#1

The Windows environment's reserved words are used in Visual C++. These reserved words are defined in Visual C++ 6.0 or later and begin with two consecutive underscores (__). You can also use the reserved words that begin with a single underscore (_) in source codes created with an earlier version of Visual C++.

The UNIX environment's reserved words are used in GCC.

#2

Reserved words treated as type qualifiers or type specifiers cannot be used in the declaration of an embedded variable.

22. When you specify the `-E` option, a bit field can be declared as any integer type. Note that a bit field cannot be used in the declaration of an embedded variable.

7.2.2 Program example

This section provides an example of an embedded SQL UAP written in C language. For details about the SQL syntax, see the *HiRDB Version 9 SQL Reference* manual.

(1) Examples of basic operations

(a) PAD chart

Figures 7-2 and *7-3* show a flowchart of the program example.

Figure 7-2: Flowchart example of an embedded SQL UAP written in C

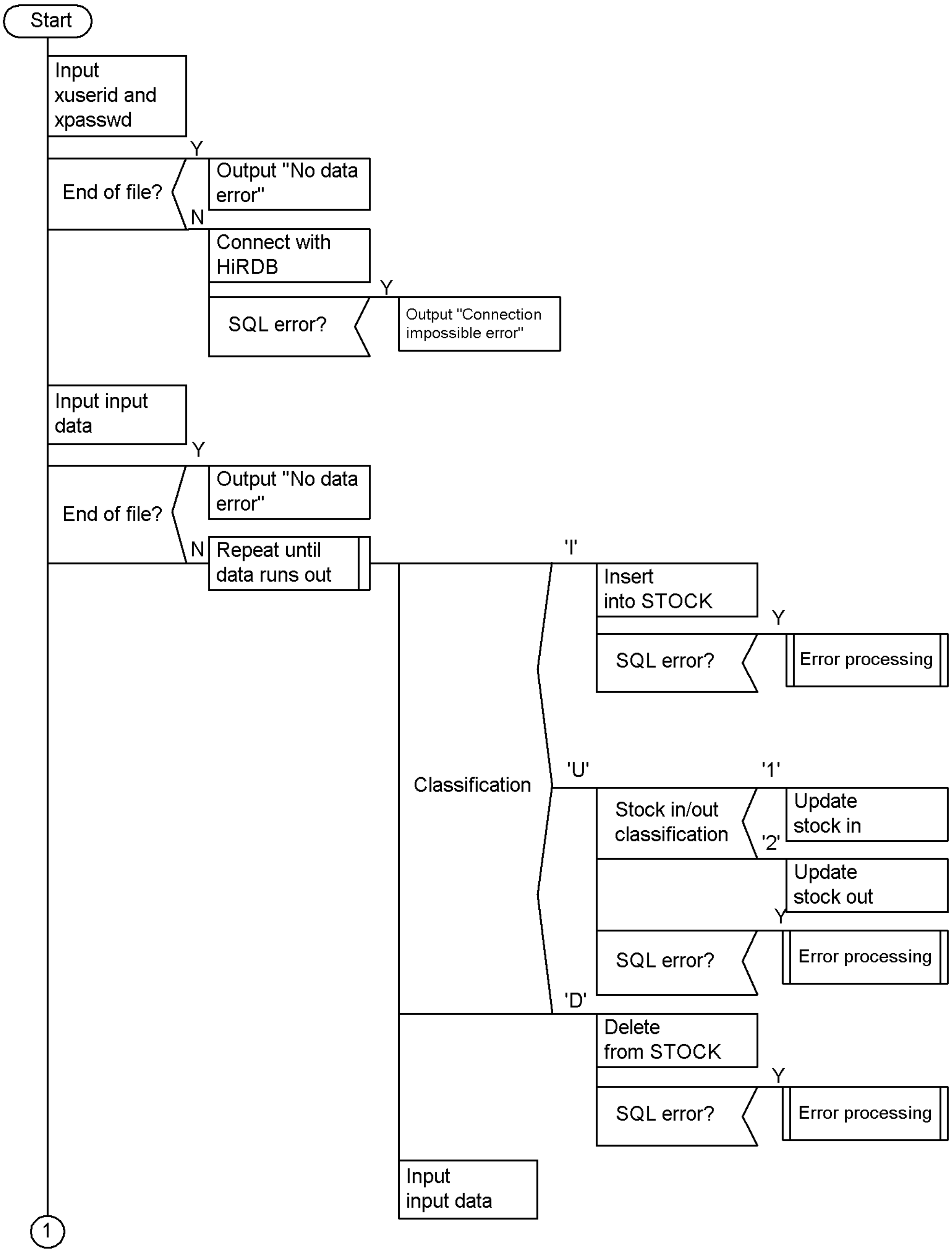
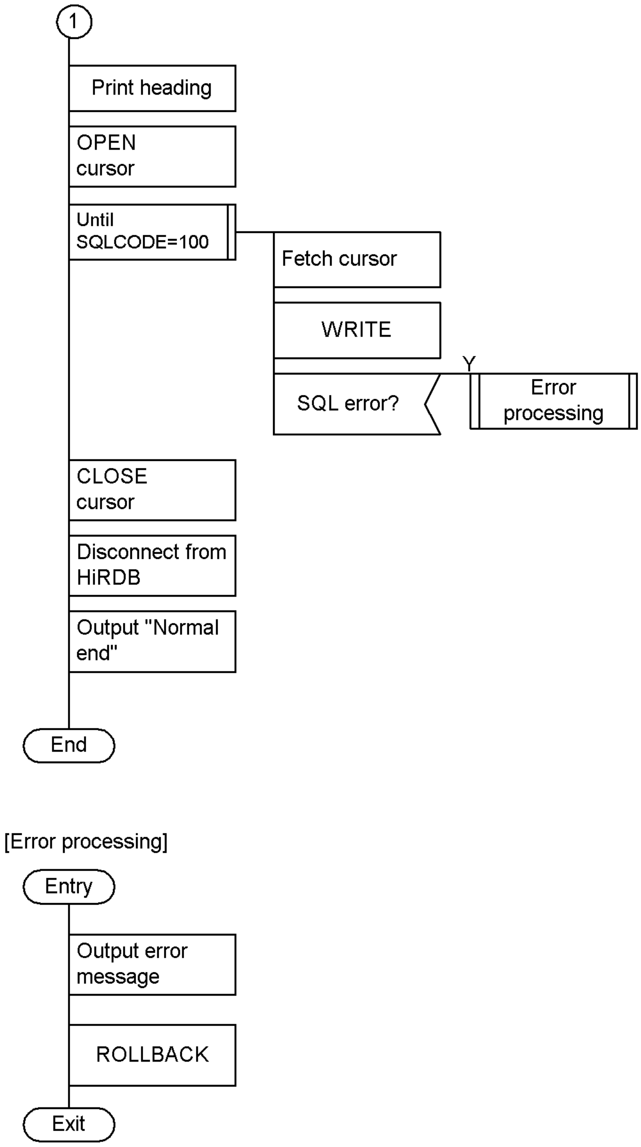


Figure 7-3: Flowchart example of an embedded SQL UAP written in C



(b) Coding example

A coding example of an embedded SQL UAP written in C follows:

```
#include <string.h>
#include <stdlib.h>

#define MAXCOLUMN 80          /* max column in one line */
```

```
#define INFILE   "inputf1"    /* input data file name   */

/* declare functions */
void abnormalend();
void connecterror();

FILE *input = NULL;

main()
{
   /* input data */
   char indata[MAXCOLUMN + 1];

   char in_userid[31];
   char in_passwd[31];
   char in_type;
   char in_pcode[5];
   char in_pname[17];
   char in_color[3];
   int in_price;
   int in_stock;
   char in_flux;

   /* variables for SQL */
  EXEC SQL BEGIN DECLARE SECTION;                            1
     char xuserid[31];                                       1
     char xpasswd[31];                                       1
     char xpcode[5];                                         1
     char xpname[17];                                        1
     char xcolor[3];                                         1
     int  xprice;                                            1
     int  xstock;                                            1
  EXEC SQL END DECLARE SECTION;                              1

  /* input file open */                                      2
  input = fopen(INFILE, "r");                                2
  if (input == NULL) {                                       2
    /* input file open error */                              2
    fprintf(stderr, "can't open %s.", INFILE);               2
    goto FIN;                                                2
  }                                                          2
                                                             2
  /* get userid/passwd */                                    2
  fgets(indata, 81, input);                                  2
  sscanf(indata, "%30s %30s", xuserid, xpasswd);             2
  if (feof(input)) {                                         2
     fprintf(stderr, "*** error *** no data for connect
              ***");                                         2
     goto FIN;                                                  2
  }                                                            2
  printf("connect start,\n");                                  2
EXEC SQL WHENEVER SQLERROR PERFORM connecterror;         (a) 2
EXEC SQL CONNECT USER :xuserid USING :xpasswd;           (b) 2
  printf("connected,\n");                                      2

   /* read data from inputfile */
   EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;
   fgets(indata, MAXCOLUMN, input);

   while (!feof(input)) {
     sscanf(indata, "%c %4s %16s %2s %8d %8d %c",
       &in_type, in_pcode, in_pname, in_color,
       &in_price, &in_stock, &in_flux);
     switch (in_type) {
     case 'I':
       strncpy(xpcode, in_pcode, 4);
       strncpy(xpname, in_pname, 8);
       strncpy(xcolor, in_color, 2);
       xprice = in_price;

       xstock = in_stock;
       EXEC SQL                                                 3
          INSERT INTO
       STOCK(PCODE,PNAME,COLOR,PRICE,SQUANTITY)                 3
         VALUES(:xpcode,:xpname,:xcolor,:xprice,:xstock);
                                                                3
       break;
     case 'U':
      strncpy(xpcode, in_pcode, 4);                             4
      xstock = in_stock;                                        4
      if (in_flux == '1') {                                     4
        EXEC SQL                                            (a) 4
           UPDATE STOCK
               SET SQUANTITY =SQUANTITY+:xstock             (a) 4
           WHERE  PCODE=: xpcode;                           (a) 4
      } else {                                                  4
        EXEC SQL                                            (b) 4
           UPDATE STOCK
               SET SQUANTITY=SQUANTITY-:xstock              (b) 4
            WHERE PCODE=:xpcode;                            (b) 4
       }
       break;
     case 'D':
       strncpy(xpcode, in_pcode, 4);
      EXEC SQL                                                  5
       DELETE FROM STOCK WHERE PCODE=:xpcode;                   5
```

```
       break;
     }
     fgets(indata, MAXCOLUMN, input);
   }

   /* print stock list */
   EXEC SQL                                                      6
    DECLARE CR1 CURSOR FOR                                       6
         SELECT PCODE,PNAME,COLOR,PRICE,SQUANTITY FROM
        STOCK;                                                   6
   EXEC SQL OPEN CR1;                                            7

   /* print title */
   printf("\n\n");
   printf("  ***** Stock Table List *****\n\n");
   printf("  Product code  Product name  Color  Price
              Current stock\n");
   printf(" ----  ---------------- -- --------
          --------\n");

   /* FETCH */
   SQLCODE = 0;
   while (SQLCODE <= 100) {
     EXEC SQL WHENEVER NOT FOUND GO TO FINISH;
     EXEC SQL                                                    8
       FETCH CR1 INTO :
     xpcode,:xpname,:xcolor,:xprice,:xstock;                     8
     EXEC SQL WHENEVER NOT FOUND CONTINUE;
     printf("  %4s   %-16s %2s %8d %8d\n",
         xpcode, xpname, xcolor, xprice, xstock);
   }

 FINISH:
   /* finish */
   EXEC SQL CLOSE CR1;                                       (a) 9
   EXEC SQL COMMIT;                                          (b) 9
   printf(" *** normal ended ***\n");

 FIN:
   if (input != NULL) {
     fclose(input);
 }
  EXEC SQL WHENEVER SQLERROR  CONTINUE;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL DISCONNECT;                                          10
  return(0);
}
```

```


Void connecterror()
{

 printf("\n************ error *** cannot connect ***\n");
    fclose(input);
    EXEC SQL DISCONNECT;
    exit(1);
}


void abnormalend()
    {
    int  wsqlcode;

    wsqlcode = -SQLCODE;
    printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n",
wsqlcode);
    printf("SQLERRMC = %s\n", SQLERRMC);

   EXEC SQL ROLLBACK;                                    (a) 11
   EXEC SQL DISCONNECT;                                  (b) 11
    exit(2);
}
```

1. Starting and ending the embedded SQL declaration section

   Encloses the variables to be used in the UAP between `BEGIN DECLARE SECTION` and `END DECLARE SECTION`. The variables indicate the start and end of the embedded SQL declaration section.

2. Connecting with HiRDB

   Specifying the abnormal processing

   > Specifies the branch destination (`connecterror`) as the process to be executed if an error (`SQLERROR`) occurs after execution of the subsequent SQL statements.

   Connecting to HiRDB

   > Informs HiRDB of the authorization identifier and the password so that the UAP can use HiRDB.

3. Inserting rows into the stock table

   Inserts the values read into the embedded variables into each column of the stock table.

4. Updating stock table rows

Incoming stock

Sets the product code that was read into the embedded variable (`:xpcode`) as the key, and retrieves the row to be updated from the stock table. Updates the row by adding the value that was read into the embedded variable (`:xquantity`) to the `QUANTITY` value of the retrieved row.

Stock

Sets the product code that was read into the embedded variable (`:xpcode`) as the key, and retrieves the row to be updated from the stock table. Updates the row by deleting the value that was read into the embedded variable (`:xquantity`) from the `QUANTITY` value of the retrieved row.

5. Deleting stock table rows

   Sets the product code that was read into the embedded variable (`:xpcode`) as the key, and deletes the rows that have a key equal to that value.

6. Declaring the `CR1` cursor

   Declares the `CR1` cursor for retrieving rows from the stock table (`STOCK`).

7. Opening the `CR1` cursor

   Positions the cursor immediately in front of a row to be retrieved from the stock table (`STOCK`) so that the row can be fetched.

8. Fetching stock table rows

   Retrieves the row indicated by the `CR1` cursor from the stock table (`STOCK`), and sets the row values into the embedded variables.

9. Closing cursor `CR1` and terminating a transaction

   Closing the `CR1` cursor

   Closes the `CR1` cursor.

   Terminating HiRDB

   Terminates the current transaction normally, and validates the results of the database addition, update, and deletion operations that were executed in that transaction.

10. Disconnecting from HiRDB

    Disconnects the UAP from HiRDB.

11. Rolling back the transaction

    Invalidating the transaction

    Rolls back the current transaction to invalidate the results of the database addition, update, and deletion operations that were executed in that translation.

Disconnecting from HiRDB

Disconnects the UAP from HiRDB.

(2) Example that uses an SQL descriptor area for user definitions

(a) PAD chart

Figures 7-4 through *7-7* show the PAD chart for program example 2.

Figure 7-4: PAD chart for program example 2 (1/4)

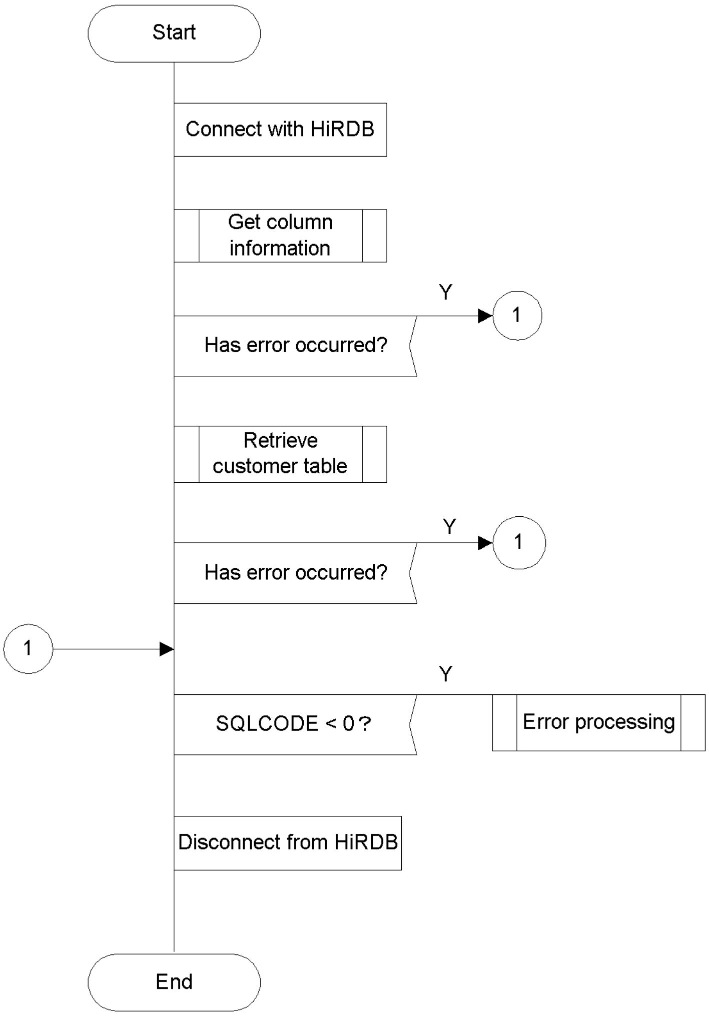


Figure 7-5: PAD chart for program example 2 (2/4)

[Column information acquisition]

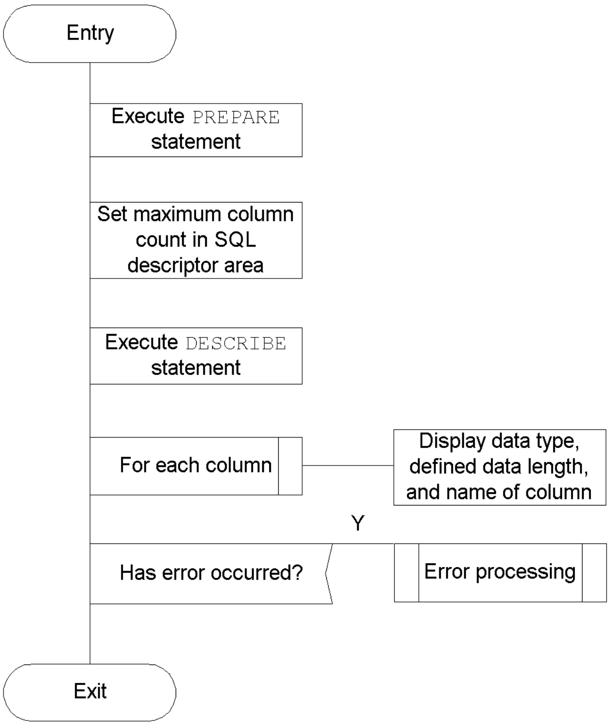

Entry
Execute PREPARE statement
Set maximum column count in SQL descriptor area
Execute DESCRIBE statement
For each column
Display data type, defined data length, and name of column
Y
Has error occurred?
Error processing
Exit

Figure 7-6: PAD chart for program example 2 (3/4)

[Customer table retrieval]

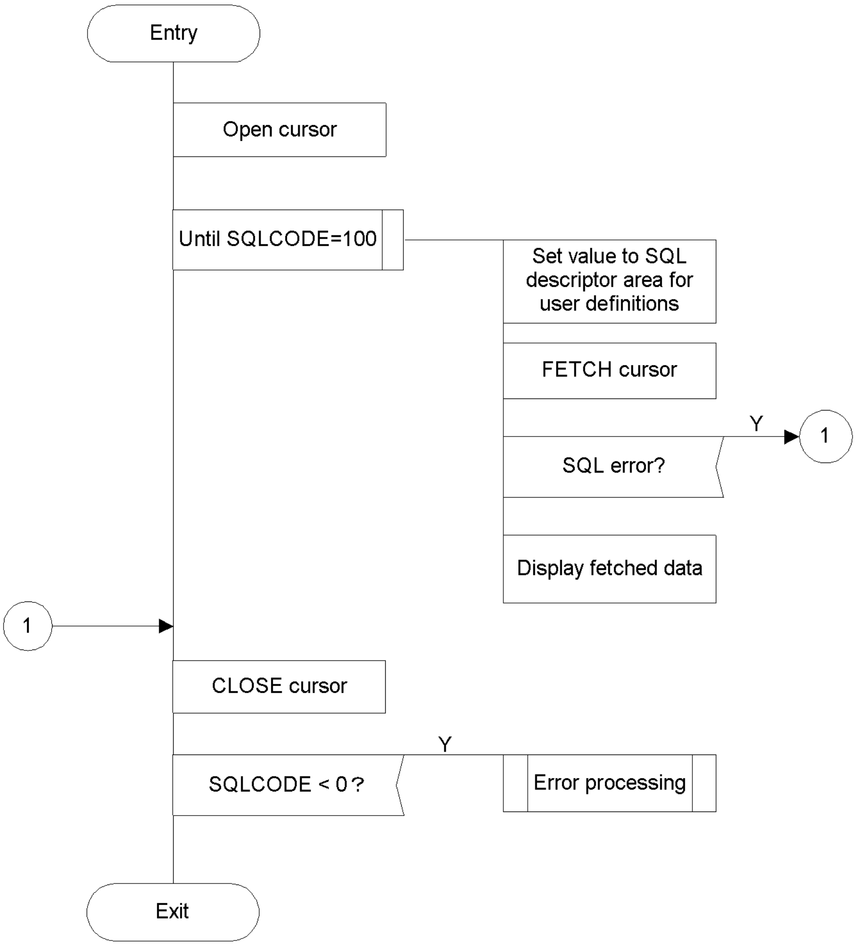
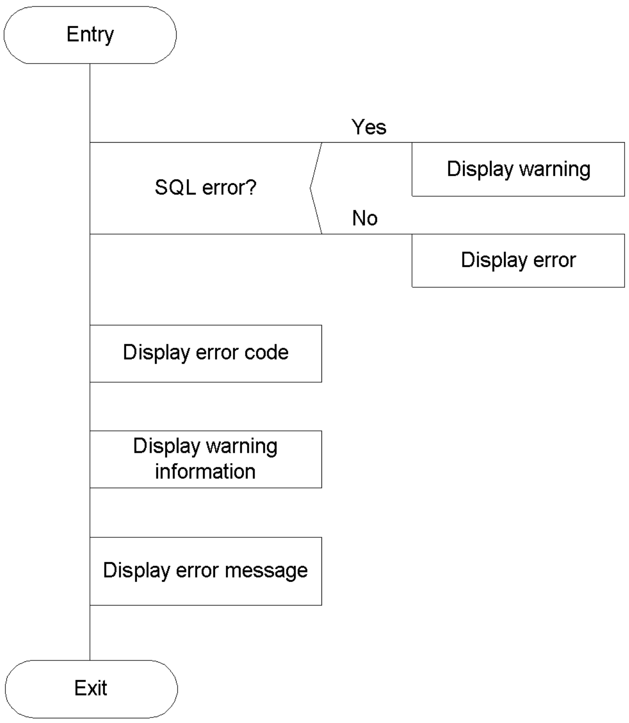


Figure 7-7: PAD chart for program example 2 (4/4)

[Error processing]



(b) Coding example

A coding example for program example 2 follows:

```
/************************************************/
/*                                              */
/*  ALL RIGHTS RESERVED, COPYRIGHT (C) 1997, HITACHI, LTD.
*/
/*  LICENSED MATERIAL OF HITACHI, LTD.          */
/*                                              */
/*  SAMPLE OF FETCH WITH SQLDA                  */
/*                                              */
/************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pdbsqlda.h"                          1
```

```


static void Describe();
static void Fetch();
static void ClearSqlda(short);
static void errmsg();

/************************************************/
/*   GLOBAL VARIABLE                            */
/************************************************/
short ErrFlg;

/************************************************/
/*   GLOBAL VARIABLE                            */
/************************************************/

/* sqlda */
PDUSRSQLDA(10) xsqlda;                           2

/* sqlcnda */                                    3
struct {                                         3
   short     sqlnz;                              3
   struct {                                      3
       short    sqlnamel;                        3
       char     sqlnamec[30];                    3
    } SQLNAME[10];                               3
} ucnda;                                         3


/************************************************/
/*                                              */
/*   MAIN ROUTINE                               */
/*                                              */
/************************************************/
int main(
int argc,
char *argv[])
{

/************************************************/
/*   CONNECT                                    */
/************************************************/
     EXEC SQL
       WHENEVER SQLERROR   GOTO:ERR_EXIT

     printf("***** connect start \n");
     EXEC SQL
       CONNECT;                                  4
```

```
      printf("***** connect : END\n");

 /************************************************/
 /*   DESCRIBE                                  */
 /************************************************/
      Describe();                                 5
      if(ErrFlg <0){                              5
         goto ERR_EXIT;                           5
      }                                           5

                                                  5
 /*********************************************/  5
 /*   FETCH                                   */  5
 /*********************************************/  5
      Fetch();                                    5
      if(ErrFlg <0){                              5
         goto ERR_EXIT;                           5
      }                                           5

/**********************************************/
 /*   END OF ALL                              */
 /*********************************************/
 ERR_EXIT :
      if(SQLCODE <0){
          errmsg();
          ErrFlg = -1;
      }

      EXEC SQL
        WHENEVER SQLERROR CONTINUE;
      EXEC SQL
        WHENEVER NOT FOUNT CONTINUE;
      EXEC SQL
        WHENEVER SQLWARNING CONTINUE;

      EXEC SQL
        DISCONNECT;                               6

      return (ErrFlg);
 }


 /*********************************************/
 /*                                           */
 /*   DYNAMIC CURSOR                          */
 /*                                           */
 /*********************************************/
 static void Fetch()
```

```
{
     EXEC SQL BEGIN DECLARE SECTION;
     char XCUSTOM_CD[6];
     char XCUSTOM_NAME[31];
     char XTELNO[13];
     char XZIPCD[4];
     char XADDRESS[31];
     EXEC SQL END DECLARE SECTION;

     EXEC SQL
       WHENEVER SQLERROR GOTO :Exit_Fetch;

     EXEC SQL
       DECLARE CUR2 CURSOR FOR SEL1;             7

/*********************************************/
/*   OPEN CURSOR                             */
/*********************************************/
     printf("***** DYNAMIC CURSOR open start\n");
     EXEC SQL
       OPEN CUR2;                                8
     printf("***** DYNAMIC CURSOR open : END\n");


/*********************************************/
/*   FETCH                                   */
/*********************************************/
     printf("***** fetch  (use sqlda) start\n");


     EXEC SQL
       WHENEVER NOT FOUND GOTO FETCH2_END;

     for(;;) {
     ClearSqlda(5);                              9
     PDSQLDATA(xsqlda, 0) = (void *)
               XCUSTOM_CD;                   (a) 9
     PDSQLCOD(xsqlda, 0) = PDSQL_CHAR;       (a) 9
     PDSQLLEN(xsqlda, 0) = sizeof
              (XCUSTOM_CD)-1;                (a) 9
     PDSQLDATA(xsqlda, 1) = (void *)
               XCUSTOM_NAME;  (b) 9
     PDSQLCOD(xsqlda, 1) = PDSQL_CHAR;       (b) 9
     PDSQLLEN(xsqlda, 1) = sizeof
             (XCUSTOM_NAME)-1;               (b) 9
     PDSQLDATA(xsqlda, 2) = (void *)XTELNO;  (c) 9
     PDSQLCOD(xsqlda, 2) = PDSQL_CHAR;       (c) 9
     PDSQLLEN(xsqlda, 2) = sizeof(XTELNO)-1; (c) 9
     PDSQLDATA(xsqlda, 3) = (void *)XZIPCD;  (d) 9
     PDSQLCOD(xsqlda, 3) = PDSQL_CHAR;       (d) 9
     PDSQLLEN(xsqlda, 3) = sizeof(XZIPCD)-1; (d) 9
     PDSQLDATA(xsqlda, 4) =
              (void *)XADDRESS;              (e) 9
     PDSQLCOD(xsqlda, 4) = PDSQL_CHAR;       (e) 9
     PDSQLLEN(xsqlda, 4) =
              sizeof(XADDRESS)-1;            (e) 9

     memset(XCUSTOM_CD, 0, sizeof(XCUSTOM_CD));
     memset(XCUSTOM_NAME, 0, sizeof(XCUSTOM_NAME));
     memset(XTELNO, 0, sizeof(XTELNO));
     memset(XZIPCD, 0, sizeof(XZIPCD));
     memset(XADDRESS, 0, sizeof(XADDRESS));

     EXEC SQL FETCH CUR2
       USING DESCRIPTOR :xsqlda;                10

     printf("%s", XCUSTOM_CD);
     printf("%s", XCUSTOM_NAME);
     printf("%s", XTELNO);
     printf("%s", XZIPCD);
     printf("%s\n", XADDRESS);
  }
FETCH2_END:
   printf("***** fetch : END\n");

/*********************************************/
/*   CLOSE CURSOR                            */
/*********************************************/
     printf("***** close start\n");
     EXEC SQL
     WHENEVER NOT FOUND CONTINUE;
     EXEC SQL
       CLOSE CUR2;                              11
     printf("***** close : END\n");

/*********************************************/
/*                                           */
/*********************************************/
Exit_Fetch:
    if(SQLCODE <0){
        errmsg();
        ErrFlg = -1;
    }
    return;
}

```

```

/**********************************************/
/*   DESCRIBE                                  */
/**********************************************/
static void Describe()
{
    short I;

    EXEC SQL
      WHENEVER SQLERROR GOTO :Exit_Describe;

/**********************************************/
/*   PREPARE                                   */
/**********************************************/
     printf("***** prepare start\n");
     EXEC SQL                                       12
       PREPARE SEL1                                 12
         FROM 'SELECT * FROM CUSTOM'                12
         WITH SQLNAME OPTION;                       12
     printf("***** prepare : END\n");

/**********************************************/
/*   DESCRIBE                                  */
/**********************************************/
     PDSQLN(xsqlda) = 10;
     printf("***** describe start\n");
     EXEC SQL
       DESCRIBE SEL1 INTO :xsqlda :ucnda;           13
     printf("***** describe : END\n");

     printf("  describe result\n");
     printf("   NUMBER OF DATA =%d\n",
                PDSQLD(xsqlda));
     printf("   NUMBER OF COLUMN NAME = %d\n",
             ucnda.sqlnz);
     for (i=0 ; i < ucnda.sqlnz ; i++) {
     printf(" [%d]), i);
         printf(" DATA TYPE(%d)", PDSQLCOD(xsqlda,
                i));
         printf(" DATA LENGTH(%d)",
                 PDSQLLEN(xsqlda, i));
         printf(" COLUMN NAME(%s)\n",
                 ucnda.SQLNAME[i].sqlnamec);
     }

/**********************************************/
/*                                             */
/**********************************************/
```

```
Exit_Describe:
    if(SQLCODE <0){
         errmsg();
         ErrFlg = -1;
    }
    return;
}


/*********************************************/
/*   Clear SQLDA                             */
/*********************************************/
static void ClearSqlda (
short num)
{
    PDSQLN(xsqlda) = num;                          14
    PDSQLD(ssqlda) = num;                          14
    while(num → 0){
        PDSQLDATA(xsqlda, num) = NULL;             15
        PDSQLIND(xsqlda, num) = NULL;              15
        PDSQLDIM(xsqlda, num) = 0;                 15
        PDSQLXDIM(xsqlda, num) = 1;                15
        PDSQLSYS(xsqlda, num) = 0;                 15
        PDSQLCOD(xsqlda, num) = 0;                 15
        PDSQLLEN(xsqlda, num) = 0;                 15
}
return;
}


/*********************************************/
/*                                           */
/*   WARNING                                 */
/*                                           */
/*********************************************/
static void errmsg()
{
   int wsqlcode;

   if(SQLCODE > 0){
       printf(">>> warning\n");
   }
   if(SQLCODE <0){
       printf(">>> error occurred\n");
   }
   wsqlcode = SQLCODE;
   printf(">>> sqlcode = %d\n", SQLCODE);
   printf(">>> sqlwarn = %c", SQLWARN0);
   printf("%c", SQLWARN1);
   printf("%c", SQLWARN2);
   printf("%c", SQLWARN3);
   printf("%c", SQLWARN4);
   printf("%c", SQLWARN5);
   printf("%c", SQLWARN6);
   printf("%c", SQLWARN7);
   printf("%c", SQLWARN8);
   printf("%c", SQLWARN9);
   printf("%c", SQLWARNA);
   printf("%c", SQLWARNB);
   printf("%c\n", SQLWARNC);

#if defined(HIUXWE2) || defined(WIN32)
    printf(">>> message = %s\n", SQLERRMC);
#else
    printf(">>> message = %Fs\n", SQLERRMC);
#endif
    return;
}
```

1. Including the distributed header file

   Declares the data code literals used for setting and referencing the SQL descriptor area, and declare the data type of the SQL descriptor area itself.

2. Declaring the SQL descriptor area

   Defines the individual SQL descriptor area for use with the UAP. The data type is defined in the distributed header file.

3. Declaring the column name descriptor area

   Defines the variable to be used when a column name is obtained with the `DESCRIBE` statement.

4. Connecting to HiRDB

   Uses the authorization identifier and password set in the `PDUSER` environment variable to connect to the server.

5. Retrieving the customer table (`CUSTOM`)

   Obtains the name of each column in the customer table (`CUSTOM`), and uses the SQL descriptor area for user definitions to retrieve and display all rows stored in the table.

6. Disconnecting from HiRDB

   Disconnects the UAP from the server.

7. Declaring the `CUR2` cursor

Declares the CUR2 cursor for retrieving customer table (CUSTOM) rows.

8. Opening the CUR2 cursor

   Positions the cursor immediately in front of a row to be retrieved from the customer table (CUSTOM) so that the row can be retrieved.

9. Setting the SQL descriptor area for user definitions

   Sets the SQL descriptor area for user definitions to be specified when the FETCH statement is executed.

   - Sets the storage area address, the data code, and the data length for the column 1 data.
   - Sets the storage area address, the data code, and the data length for the column 2 data.
   - Sets the storage area address, the data code, and the data length for the column 3 data.
   - Sets the storage area address, the data code, and the data length for the column 4 data.
   - Sets the storage area address, the data code, and the data length for the column 5 data.

10. Fetching the customer table rows

    Fetches the row indicated by the CUR2 cursor from the customer table (CUSTOM), and sets it into the area indicated by the SQL descriptor area for user definitions.

11. Closing the CUR2 cursor

    Closes the CUR2 cursor.

12. Preparing for SQL dynamic execution

    Prepares the SELECT statement for retrieving the table so that the DESCRIBE statement can fetch the column name, data type, and data length of each column in the customer table (CUSTOM).

13. Fetching column names and data types

    Fetches the data type and data length of each column in the customer table (CUSTOM), and sets the information into the SQL descriptor area for user definitions. Also, fetches the column name of each column, and sets the information into the user column name descriptor area.

14. Setting the row count for the SQL definition area for user definitions

    In the SQL descriptor area for user definitions, sets the size of the SQL descriptor area and the number of rows to be fetched.

15. Clearing the SQL descriptor area for user definitions

    Clears each column area in the SQL descriptor area for user definitions.

(3) Example of manipulating LOB data

(a) PAD chart for program example 3

Figures 7-8 through *7-10* show the PAD chart for program example 3.

Figure 7-8: PAD chart for program example 3 (1/3)

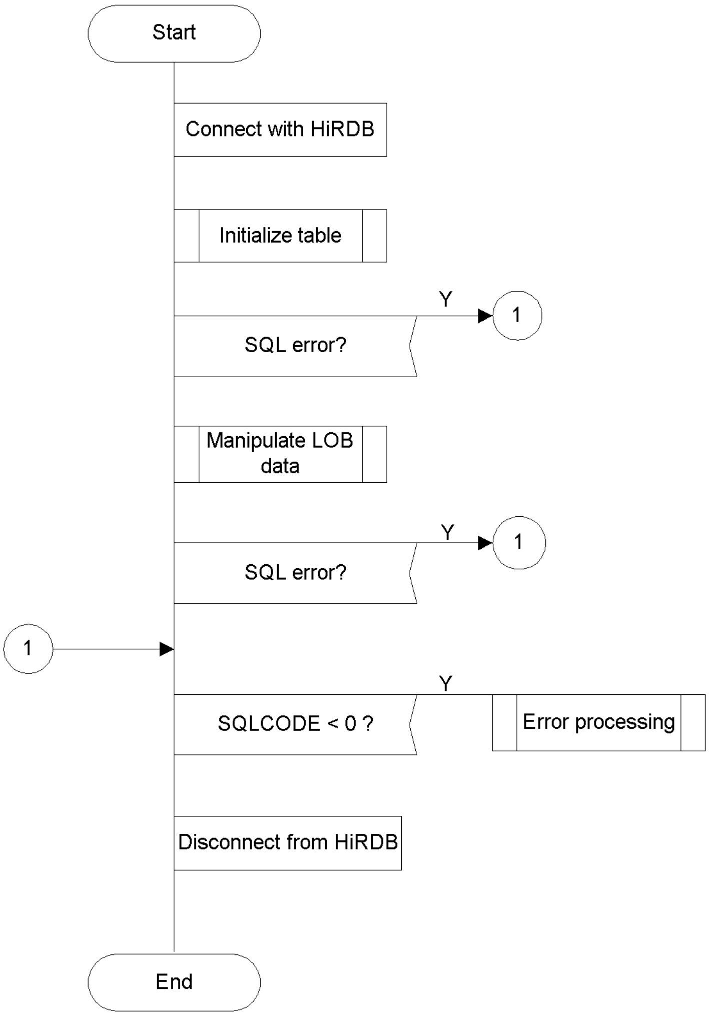


Figure 7-9: PAD chart for program example 3 (2/3)

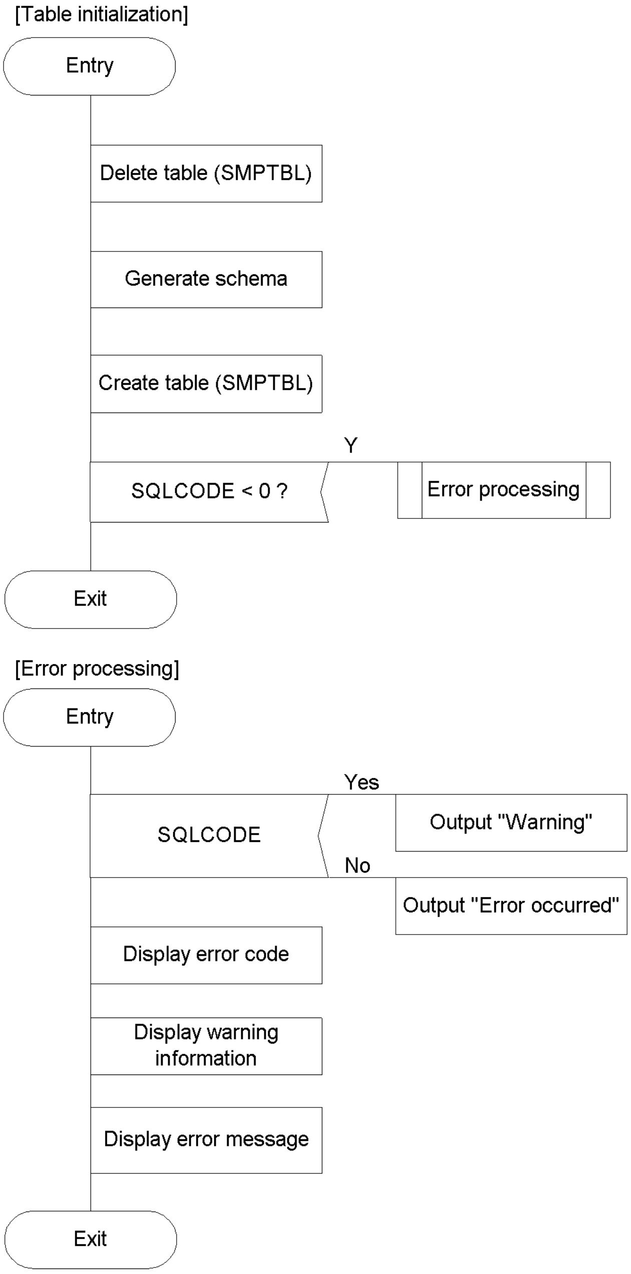
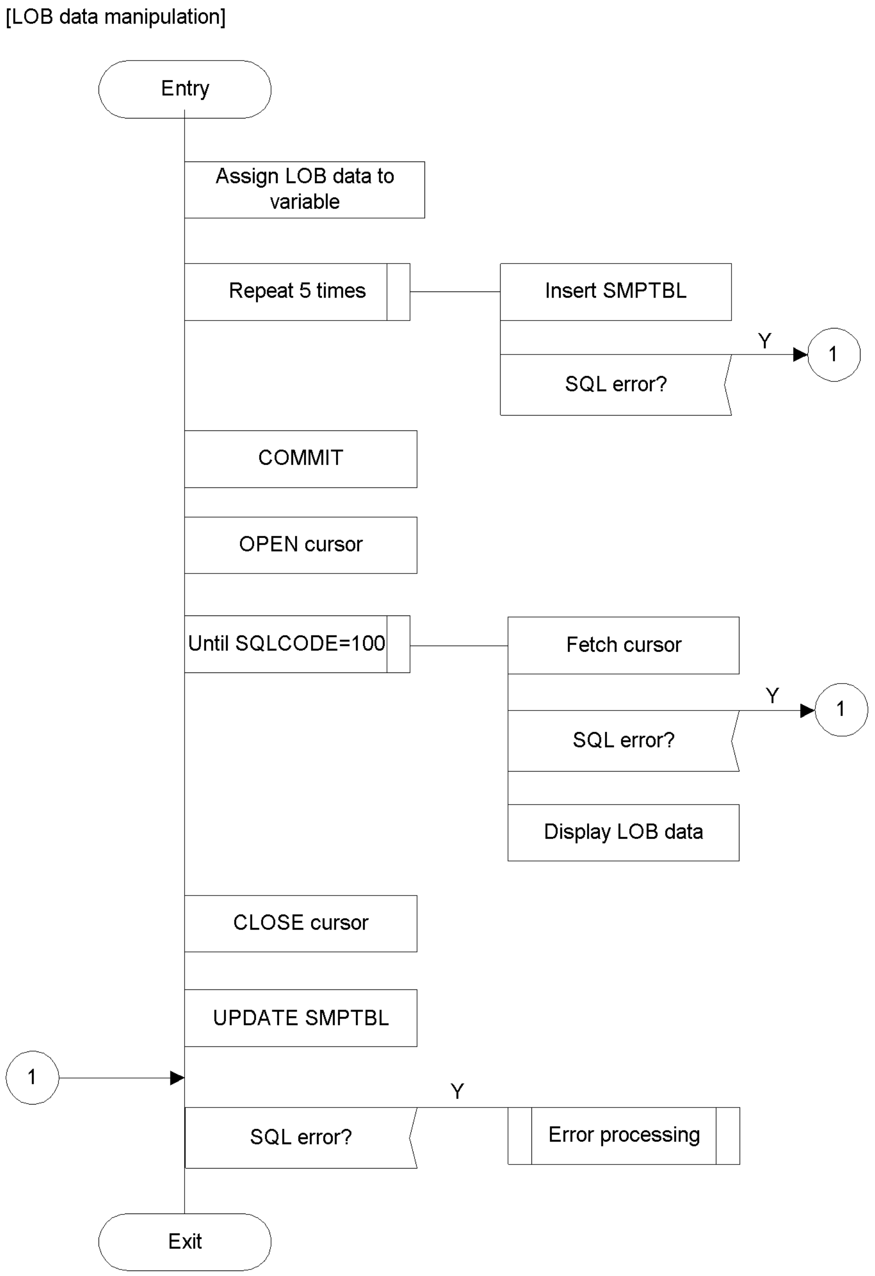


Figure 7-10: PAD chart for program example 3 (3/3)



(b) Coding example

A coding example of program example 3 follows:

```
/***********************************************/
/*                                             */
/*  ALL RIGHTS RESERVED, COPYRIGHT (C) 1997,
    HITACHI, LTD.                              */
/*  LICENSED MATERIAL OF HITACHI, LTD.         */
/*                                             */
/***********************************************/


#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <ctype.h>
#include <string.h>

static void InitTable();
static void TestBlob();
static void warning();


/***********************************************/
/*   GLOBAL VARIABLE                           */
/***********************************************/
short ErrFlg;

EXEC SQL BEGIN DECLARE SECTION;
  short XSINT_IN;
  short XSINT_OUT;
  long  XINT_IN;
  long  XINT_OUT;
  SQL TYPE IS BLOB(16K) XBLOB_IN;                1
  SQL TYPE IS BLOB(16K) XBLOB_OUT;               1
EXEC SQL END DECLARE SECTION;

/*
 *   name = MAIN
 *   func =  SAMPLE
 *   io   =  argc : i :
 *           argv : i :
 *   return = 0,-1
 *   note   = This program needs "RDUSER02" rdarea
              on Server.
 *   date   = 98.04.24 by matsushiba
*/
int main(
int   argc,
char  *argv[])
{
    ErrFlg = 0;

/***********************************************/
/*                                             */
/***********************************************/
EXEC SQL
  WHENEVER SQLERROR   goto ERR_EXIT;

EXEC SQL
  WHENEVER SQLWARNING PERFORM :warning;

EXEC SQL CONNECT;                                 2


/***********************************************/
/*   INIT                                      */
/***********************************************/
     InitTable();                                 3
     if(ErrFlg <0){                               3
     goto ERREXIT;                                3
     }                                            3

/***********************************************/
/*                                             */
/***********************************************/
     TestBlob();                                  4
     if(ErrFlg <0){                               4
         goto ERR_EXIT;                           4
     }                                            4

/***********************************************/
/*                                             */
/***********************************************/
ERREXIT:
    if(SQLCODE <0){
        printf(":> ERROR HAPPENED!!\n");
        warning();
        ErrFlg = -1;
    }

EXEC SQL
  WHENEVER SQLERROR CONTINUE;
EXEC SQL
  WHENEVER NOT FOUND CONTINUE;
```

```
EXEC SQL
  WJEMEVER SQLWARNING CONTINUE;

EXEC SQL DISCONNECT;                             5

return (ErrFlg);
}


/***********************************************/
/*   INIT                                      */
/***********************************************/
static void InitTable()
{

/***********************************************/
/*                                             */
/***********************************************/
     EXEC SQL
       WHENEVER SQLERROR CONTINUE;

     EXEC SQL                                    6
       DROP TABLE SMPTBL;                        6
                                                 6
     EXEC SQL                                    6
       CREATE SCHEMA;                            6

     printf("## CREATE TABLE\n");

     EXEC SQL
       WHENEVER SQLERROR GOTO INIT_ERROR;

     printf("## CREATE SMPTBL\n");
     EXEC SQL            7
       CREATE TABLE SMPTBL(CLM1   BLOB(30K) IN
                                  RDUSER02,      7
                           CLM2   SMALLINT,      7
                           CLM3   INTEGER);      7

     return;

INIT_ERROR:
   warning();
   ErrFlg = -1;
   return;
}


```

```

/************************************************/
/*   TEST BLOB                                  */
/************************************************/
static void TestBlob()
{
    short cnt;

    EXEC SQL
     WHENEVER  SQL ERROR goto :ExitTestBlob;

    EXEC SQL
     WHENEVER  SQLWARNING PERFORM :warning;

/***********************************************/
/*   INSERT                                    */
/***********************************************/
     memset(XBLOB_IN.XBLOB_IN_data,
            0x55,
            sizeof(XBLOB_IN.XBLOB_IN_data));
     XBLOB_IN.XBLOB_IN_length =
     sizeof(XBLOB_IN.XBLOB_IN_data);

     printf("## INSERT \n");
     for(cnt=1; cnt<5; cnt++){
         XSINT_IN = cnt;
         XINT_IN = 100+cnt;
         EXEC SQL                                 8
           INSERT INTO SMPTBL                     8
           VALUES(:XBLOB_IN, :XSINT_IN,
                  :XINT_IN);                      8
     }
     EXEC SQL COMMIT;

/***********************************************/
/*   FETCH                                     */
/***********************************************/
     printf("## FETCH \n");

     EXEC SQL                                     9
       DECLARE CUR_BLOB CURSOR FOR                9
         SELECT * FROM SMPTBL;                    9

     EXEC SQL
       OPEN CUR_BLOB;                             10

     EXEC SQL
       WHENEVER NOT FOUND GOTO FETCH_END;

     for(;;){
         memset (XBLOB_OUT.XBLOB_OUT_data,
                 0
                 sizeof(XBLOB_OUT.XBLOB_OUT_data));
         XBLOB_OUT.XBLOBL_OUT_length = 0;
         EXEC SQL                               11
           FETCH CUR_BLOB INTO :XBLOB_OUT,      11
                               :XSINT_OUT,      11
                               :XINT_OUT;       11

           printf("CLM1 XBLOB_length == %d\n",
                    XBLOB_OUT.XBLOB_OUT_length);
           printf("CLM2 = %d\n", XSINT_OUT);
           printf("CLM3 = %ld\n", XINT_OUT);
      }
FETCH_END:
   EXEC SQL
     WHENEVER NOT FOUND CONTINUE;

   EXEC SQL
     CLOSE CUR_BLOB;                            12

/***********************************************/
/*   UPDATE                                    */
/***********************************************/
     memset(XBLOB_IN.XBLOB_IN_data,
            0x38,
            sizeof(XBLOB_IN.XBLOB_IN_data));
     XBLOB_IN.XBLOB_IN_length =
     sizeof(XBLOB_IN.XBLOB_IN_data);

     printf("## UPDATE\n");
     EXEC SQL
       UPDATE SMPTBL SET CLM1=:XBLOB_IN;        13

     EXEC SQL COMMIT

/***********************************************/
/*                                             */
/***********************************************/
ExitTestBlob:
    if(SQLCODE < 0){
        warning();
        ErrFlg = -1;
    }
     return;
    )
```

```


/***********************************************/
/*   WARNING                                   */
/***********************************************/
static void warning()
{
     if(SQLCODE <0){
         printf(">>>ERROR\n");
         printf(">>> sqlcode = %d\n", SQLCODE);
#if defined(HIUXWE2)  ||  defined(WIN32)
         printf(":> message = %s\n", SQLERRMC);
#else
         printf(":> message = %Fs\n", SQLERRMC);
#endif
    }
    else{
         printf(">>>WARNING\n");
         printf(">>>sqlwarn = %c", SQLWARN0)
         printf("%c", SQLWARN1);
         printf("%c", SQLWARN2);
         printf("%c", SQLWARN3);
         printf("%c", SQLWARN4);
         printf("%c", SQLWARN5);
         printf("%c", SQLWARN6);
         printf("%c", SQLWARN7);
         printf("%c", SQLWARN8);
         printf("%c", SQLWARN9);
         printf("%c", SQLWARNA);
         printf("%c", SQLWARNB);
         printf("%c\n", SQLWARNC);
   }
   return
}
```

1. Declaring LOB-type embedded variables

   Declares the LOB-type embedded variable for writing data (:XBLOB_IN) and the LOB-type embedded variable for reading data (:XBLOB_OUT).

2. Connecting to HiRDB

   Uses the authorization identifier and password set in the PDUSER environment variable to connect to the server.

3. Initializing the table

   Defines an SMPTBL table that contains LOB-type columns.

4. Inserting, retrieving, and updating LOB data

Inserts rows that include LOB-type columns in the empty SMPTBL table, retrieves all rows, and then updates the contents of the LOB-type columns with new LOB data.

5. Disconnecting from HiRDB

   Disconnects the UAP from the server.

6. Preparing for SMPTBL creation

   To create the SMPTBL table containing LOB-type columns, deletes any tables that have the same name, and creates a schema in case there are no schemas.

7. Creating the SMPTBL table containing LOB-type columns

   Creates the SMPTBL table containing LOB-type columns. A LOB RDAREA user must be created, because the LOB data is defined for storage in a special RDAREA for LOB data. If there is no LOB RDAREA user, an error occurs.

8. Adding LOB data

   Adds the values that were set in the embedded variables (:XBLOB_IN, :XINT_IN, and :XSINT_IN) to the SMPTBL table containing LOB-type columns.

9. Declaring the CUR_BLOB cursor

   Declares the CUR_BLOB cursor for retrieving the SMPTBL table containing LOB-type columns.

10. Opening the CUR_BLOB cursor

    Positions the cursor immediately in front of a row to be retrieved from the SMPTBL table containing LOB-type columns so that the row can be fetched.

11. Fetching LOB-type data

    Fetches the row indicated by the CUR_BLOB cursor from the SMPTBL table containing LOB-type columns, and sets the data to the embedded variables (:XBLOB_OUT, :XINT_OUT, and :XSINT_OUT).

12. Closing the CUR_BLOB cursor

    Closes the CUR_BLOB cursor.

13. Updating LOB data

    Updates the values of the LOB-type columns in the SMPTBL table with the embedded variable (:XBLOB_IN) values.

7.3 Writing a UAP in COBOL

This section explains, by way of examples, the coding rules for embedding SQL statements in UAPs written in COBOL.

Note that UAPs written in COBOL cannot be created for Windows Server 2003 (IPF) edition or Linux for AP8000 edition clients.

7.3.1 Coding rules

When a UAP is written, the labeling rules, SQL coding rules, and SQL syntax rules must be followed.

(1) Labeling rules

Labels must be assigned according to COBOL rules. These rules apply to labels:

(a) SQL reserved words

- Both uppercase and lowercase letters can be used
- Uppercase and lowercase letters can be mixed

(b) Host names

- Labels that begin with `SQL` cannot be used
- Spaces can be entered following a colon within a host name
- Host names are not case-sensitive
- Uppercase and lowercase letters can be mixed
- The corresponding double-byte and single-byte versions of letters, numeric characters, symbols, katakana characters, and the space character are treated as different characters.

Embedded variables, indicator variables, and branching destination labels must be named in accordance with the COBOL labeling rules. The following types of labels, which have the external attribute, cannot be used:

- Labels that begin with the uppercase `SQL`
- Labels that begin with the lowercase `p_`
- Labels that begin with the lowercase `pd`

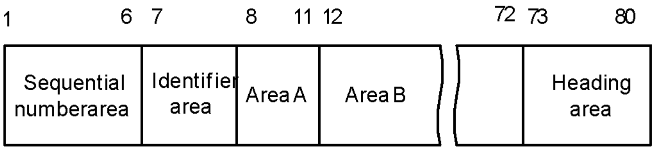
(2) SQL coding rules

1. Each SQL statement must be preceded by the SQL prefix (`EXEC SQL`) and followed by the SQL terminator (`END-EXEC`).

   Example:

EXEC SQL *SQL-statement*; END-EXEC.

2. COBOL statements and SQL statements can both be specified on the same line.
3. All SQL statements (from the SQL prefix through the SQL suffix) must be entered in the B area (columns 12 - 72)



4. The SQL statement continuation rules are generally the same as the COBOL line continuation rules.

   A line break can occur in an SQL description wherever a space must be specified or can be specified; a description can span multiple lines.

   To break a line where a space cannot be specified in the SQL statement, a hyphen (-) must be specified in the indicator area; the description can resume on the next line in any column in the B area.

   To break a line in the middle of a character string literal, the description must be specified through column 72, and a quotation mark (") must be specified anywhere in the B area on the next line. To continue the character string, first specify a quotation mark or an apostrophe (whichever was specified at the beginning of the character string), and resume the character string specification from the next column after that quotation mark or apostrophe.
5. A paragraph header can be entered before the SQL prefix (but not on the same line as the SQL prefix).

   Valid specification:
   FINISH.
   EXEC SQL *SQL-statement* END-EXEC.

   Invalid specification:
   FINISH.
   EXEC SQL *SQL-statement* END-EXEC.

   Bold letters indicate the invalid portion.
6. One SQL statement is treated as one COBOL language instruction. Therefore, if an SQL statement is the last instruction of a concluding statement, a period and space must be specified following the SQL terminator.

   Example of when an SQL statement constitutes a concluding statement:
   EXEC SQL

```
  SQL-statement
END-EXEC.
```

Example of when an SQL statement is the last instruction of a concluding statement:

```
IF U-FLUX = '2'
THEN
  EXEC SQL SQL-statement END-EXEC.
```

Example of when an SQL is an instruction in the middle of a concluding statement:

```
IF U-FLUX = '1'
THEN
  EXEC SQL SQL-statement
  END-EXEC
ELSE IF U-FLUX = '2'
    THEN NEXT SENTENCE.
```

7. Although a comment cannot be specified within an SQL statement, any number comment lines can be specified between the SQL prefix and the SQL terminator.

   Example:

```
  EXEC SQL
*Declaration of cursor for SELECT statement           (1)
*that retrieves STOCK table                             (1)
    SQL-statement
  END-EXEC.
```

   (1): Comment lines

8. The following rules apply to declaring embedded variables.
   - Specify the embedded SQL declare section in one of the following sections:
     - FILE SECTION of DATA DIVISION
     - WORKING-STORAGE SECTION
     - LOCAL-STORAGE SECTION
     - LINKAGE SECTION
   - For details about embedded variables for SQL data types, see *F. SQL Data Types and Data Descriptions*.
   - JUSTIFIED, BLANK, and the WHEN ZERO clause cannot be specified in the data description item of an embedded variable.
   - Although a level 66 re-instruction item or level 88 conditional name item cannot be used as an embedded variable, such items can be defined in an embedded SQL declaration section.

- The COBOL line continuation rules apply to continuation of data description item lines of an embedded SQL declaration section.
- FILLER cannot be used as an embedded variable.
- A data item that uses the TYPE, TYPEDEF, or SAME AS clause can be used as an embedded variable.
- If you use the REDEFINES clause, the system does not check whether the item that performs the redefining and item to be redefined use the same column justification. The size of the larger area is used.
- A data item in which the PICTURE clause is omitted and only the VALUE clause is specified cannot be used as an embedded variable.
- When you use the -E option, you can use the declared data item as an embedded variable even if you do not use an embedded SQL declare section. However, the only data items that can be used as embedded variables in SQL statements are those that are declared with a format described in *F. SQL Data Types and Data Descriptions*. Data items that are declared with other formats cannot be used as embedded variables.

  The effective scope of each data item name is determined by the COBOL syntax rules. The data items that can be used as embedded variables must be specified in the source program. Data items in library text that is included with the COPY or INCLUDE statement cannot be used as embedded variables.
- Data items that are inherited from a parent class by the class inheritance facility of COBOL2002 cannot be used as embedded variables.
- If you specify the -XU16 option when you execute the SQL preprocessor, UAPs that use the Unicode functionality of COBOL2002 can use Japanese data items for storing UTF-16 character data as embedded variables. You can specify an embedded variable using Japanese data items at any location in a SQL statement that allows an embedded variable for a mixed character string data type (MCHAR or MVARCHAR). The supported characters are limited to the range of characters supported by the Unicode functionality of COBOL2002.

  For details about UAP execution using the Unicode functionality of COBOL2002, see *8.4.3 UAP execution using the Unicode functionality of COBOL2002*.

9. The following rules apply to declaring indicator variables.

   - An indicator variable must be either a basic item between level 01 and level 49 or an independent item of level 77.
   - For details about the data description terms for embedded variables, see *F. SQL Data Types and Data Descriptions*.
   - The SIGN, JUSTIFIED, BLANK, and WHEN ZERO clauses cannot be specified

in the data description item of an indicator variable.

- FILLER cannot be used as an indicator variable name.
- When you use the -E option, you can use the declared data item as an indicator variable even if you do not use an embedded SQL declare section. However, the only data items that can be used as indicator variables in SQL statements are those that are declared with a format described in *F. SQL Data Types and Data Descriptions*. Data items that are declared with other formats cannot be used as indicator variables.

  The effective scope of each data item name is determined by the COBOL syntax rules. The data items that can be used as indicator variables must be specified in the source program. Data items in library text that is included with the COPY or INCLUDE statement cannot be used as indicator variables.
- Data items that are inherited from a parent class by the class inheritance facility of COBOL2002 cannot be used as indicator variables.

10. The following table shows the divisions (DIVISION) in COBOL in which SQL statements can be described.

Table 7-3: Divisions in COBOL for describing SQL statements

SQL statement		Data division#	Procedure division
Definition SQL		--	D
Data manipulation SQL		--	D
Control SQL		--	D
Embedded language	BEGIN DECLARE SECTION	D	--
	END DECLARE SECTION	D	--
	COPY	D	D
	WHENEVER	--	D
	DECLARE CONNECTION HANDLE UNSET	--	--
	COMMAND EXECUTE	--	--
	Other statement	--	D

D: Can be described.

--: Cannot be described.

#: Indicates the working section, file section, or linkage section.

11. Because the `WHENEVER` statement and cursor declaration are declaration statements, they cannot be specified within an `IF` or `EVALUATE` instruction.
12. Do not specify a control word for compile list output (`EJECT`, `SKIP1`, `SKIP2`, `SKIP3`, or `TITLE`) in the SQL statements that are enclosed between the SQL prefix and SQL terminator. To use `EJECT`, `SKIP1`, `SKIP2`, `SKIP3`, or `TITLE` as a table name or column name, close the word in double quotation marks. However, if `EJECT`, `SKIP1`, `SKIP2`, `SKIP3`, or `TITLE` is contained in a word phrase, such as a table name or column name, enclosing the word in quotation marks is unnecessary.
13. Comments (`/*...*/`) specified between the SQL prefix and the SQL terminator are deleted. However, SQL optimization specifications (`/*>>...<<*/`) are not deleted but instead treated as SQL statements. If a specified comment or SQL optimization specification extends over several lines, each line is assumed to start from the beginning of the B area until `*/` is specified. Do not use line continuation characters. For details about comments and SQL optimization specifications in SQL statements, see the manual *HiRDB Version 9 SQL Reference*.
14. A note (`*>`) can be specified in a line. However, notes within lines cannot be used between the SQL prefix and the SQL terminator. If a note within a line is specified, the note is treated as a character string instead of a note.
15. In lines specified in the embedded SQL declare section and SQL statements, the tab code is treated as being one character in length. If you use the `-E2` or `-E3` option, the tab code is treated as being one character in length in all data sections.
16. When the object-oriented facility of COBOL2002 is used, the rules described in *7.5.1(2) SQL coding rules* also apply.

7.3.2 Program example

This section provides an example of an embedded SQL UAP written in COBOL. For details about the SQL syntax, see the *HiRDB Version 9 SQL Reference* manual.

(1) Example of basic operation

(a) PAD chart

Figures 7-11 through *7-13* show a PAD flowchart of example 4.

Figure 7-11: Flowchart of program example 4 (1/3)

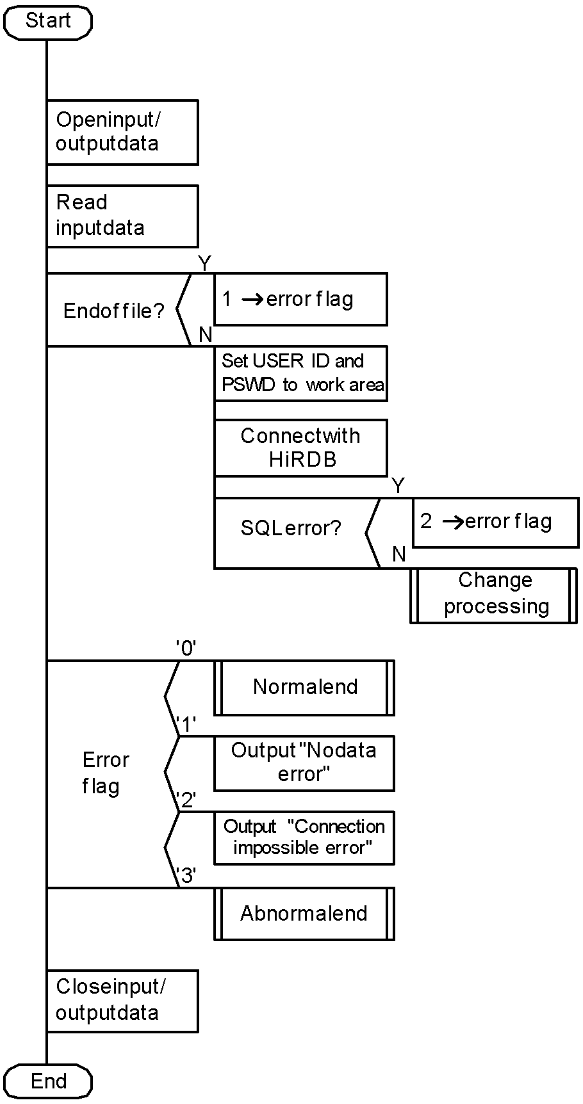


Figure 7-12: Flowchart of program example 4 (2/3)

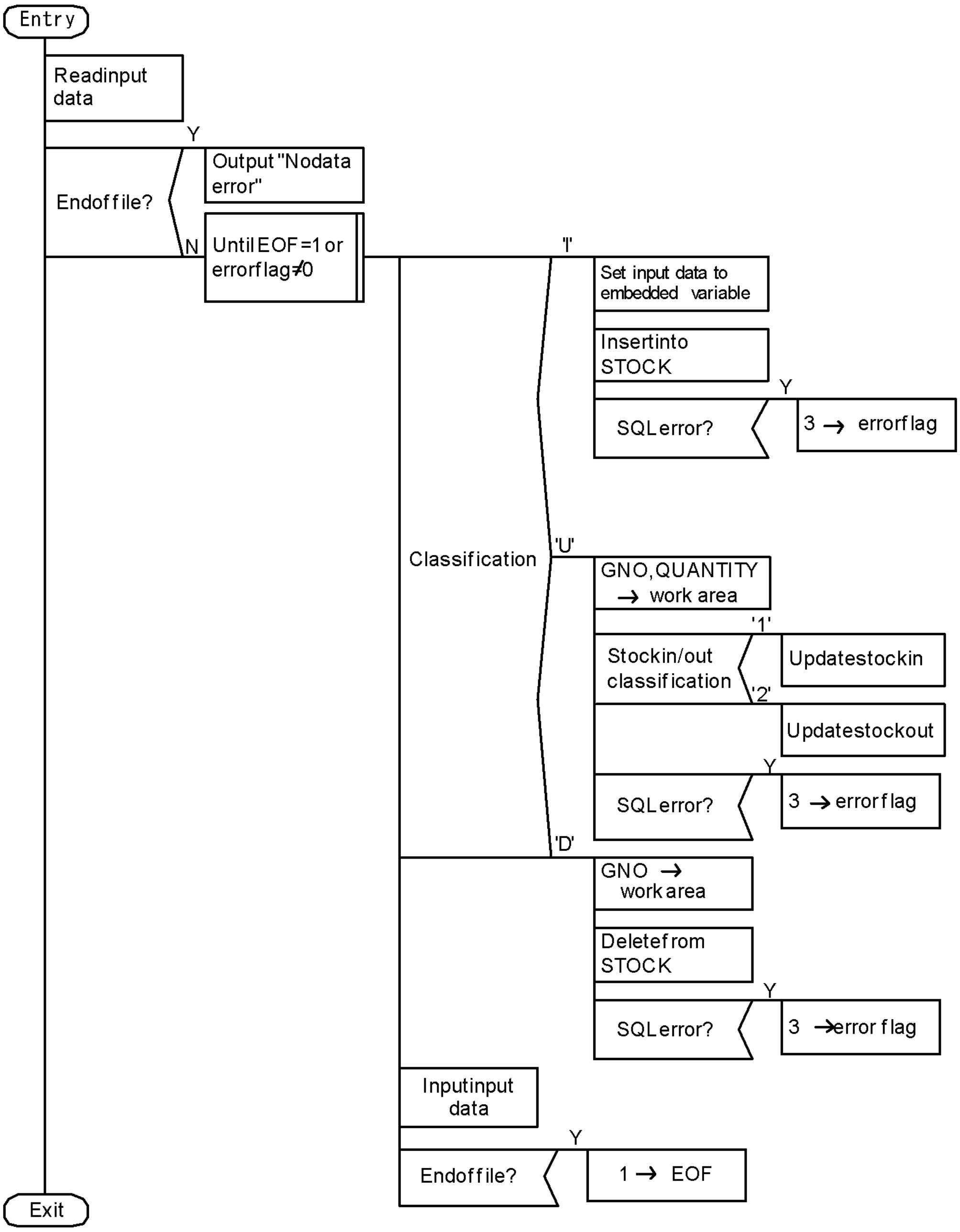
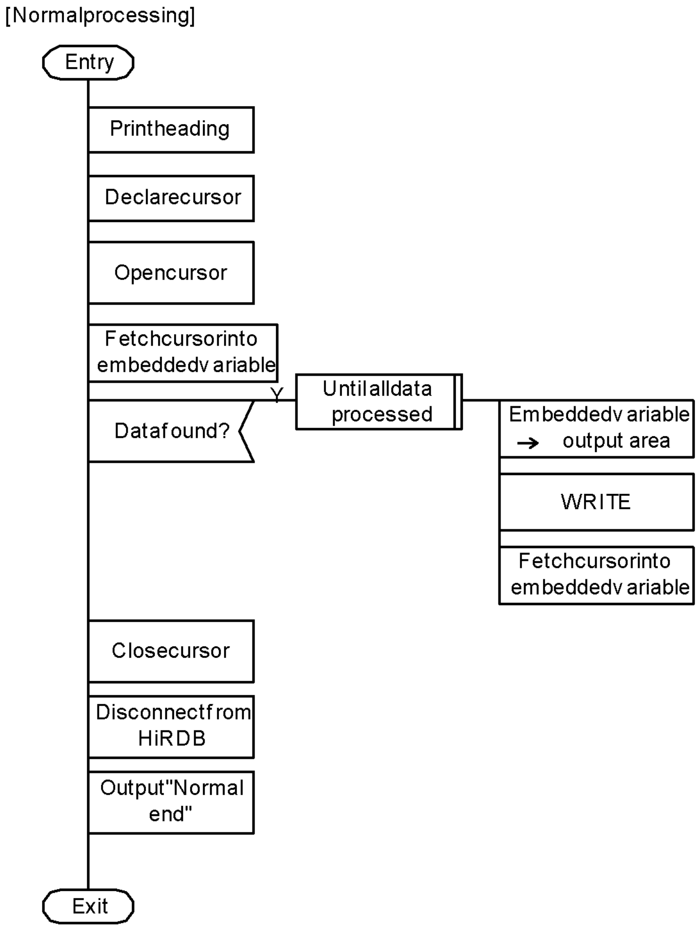
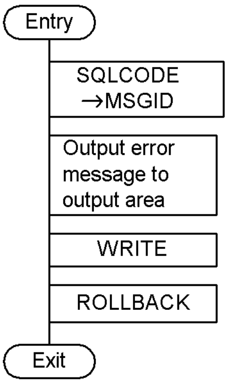


Figure 7-13: Flowchart of program example 4 (3/3)



[Error processing]



(b) Coding example

A coding example of an embedded SQL UAP written in COBOL follows:

```
00010*STOCK MANAGEMENT PROG.
00020*
00030*
00040* ALL RIGHTS RESERVED,COPYRIGHT (C)1997 HITACHI,LTD.
00050* LICENSED MATERIAL OF HITACHI,LTD.
00060*
00070 IDENTIFICATION DIVISION.
00080 PROGRAM-ID. ECOBUAP.
00090*
00100 ENVIRONMENT DIVISION.
00110 CONFIGURATION SECTION.
00120 SOURCE-COMPUTER. HITAC.
00130 OBJECT-COMPUTER. HITAC.
00140 INPUT-OUTPUT SECTION.
00150 FILE-CONTROL.
00160     SELECT INPUT-CARD-FILE
00170       ASSIGN TO  DISK
00180       ORGANIZATION IS LINE SEQUENTIAL.
00190     SELECT PRINT-STOCK-FILE
00200       ASSIGN TO  LP.
00210*
00220 DATA DIVISION.
00230 FILE SECTION.
00240 FD  INPUT-CARD-FILE
00250                    DATA RECORD USER-CARD-REC I-STOCK-REC.
00260*
00270 01  USER-CARD-REC.
00280     02 IUSERID          PIC X(20).
00290     02 IPSWD            PIC X(20).
```

```
00300     02 FILLER           PIC X(40).
00310*
00320 01  I-STOCK-REC.
00330     02 ITYPE            PIC X(1).
00340     02 FILLER           PIC X(2).
00350     02 IPCODE           PIC X(4).
00360     02 FILLER           PIC X(2).
00370     02 IPNAME           PIC N(8).
00380     02 ICOLOR           PIC N(1).
00390     02 IPRICE           PIC X(9).
00400     02 ISTOCK           PIC X(9).
00410     02 IFLUX            PIC X(1).
00420     02 FILLER           PIC X(34).
00430*
00440 FD  PRINT-STOCK-FILE RECORDING MODE IS F
00450                      LABEL RECORD IS OMITTED
00460                      DATA RECORD PRINT-STOCK-REC.
00470 01  PRINT-STOCK-REC      PIC X(132).
00480*
00490 WORKING-STORAGE SECTION.
00500*
00510     EXEC SQL                                                1
00520           BEGIN DECLARE SECTION                             1
00530     END-EXEC.                                               1
00540 77  XUSERID            PIC X(30).                           1
00550 77  XPSWD              PIC X(30).                           1
00560 77  XPCODE             PIC X(4)  VALUE '0000'.              1
00570 77  XPNAME             PIC N(8).                            1
00580 77  XCOLOR             PIC N(1).                            1
00590 77  XPRICE             PIC S9(9) COMP.                      1
00600 77  XSTOCK             PIC S9(9) COMP.                      1
00610* INDICATOR VARIABLE                                         1
00620 77  XIPCODE            PIC S9(4) COMP  VALUE 1040.          1
00630 77  XIPNAME            PIC S9(4) COMP  VALUE 1050.          1
00640 77  XICOLOR            PIC S9(4) COMP  VALUE 1060.          1
00650 77  XIPRICE            PIC S9(4) COMP  VALUE 1070.          1
00660 77  XISTOCK            PIC S9(4) COMP  VALUE 1080.          1
00670*                                                            1
00680*                                                            1
00690     EXEC SQL                                                1
00700           END DECLARE SECTION                               1
00710     END-EXEC.                                               1
00720*
00730 01  HEADING-REC.
00740     02 FILLER          PIC X(13) VALUE SPACE.
00750     02 FILLER          PIC X(32)
00760           VALUE '******  STOCK TABLE LIST  ******'.
00770     02 FILLER          PIC X(87) VALUE SPACE.
```

```
00780*
00790 01  COLUMN-NAME-REC.
00800     02 FILLER          PIC X(14) VALUE SPACE.
00810     02 FILLER          PIC X(9)  VALUE 'PCODE'.
00820     02 FILLER          PIC X(16) VALUE 'PNAME'.
00830     02 FILLER          PIC X(8)  VALUE 'COLOR'.
00840     02 FILLER          PIC X(8)  VALUE 'PRICE'.
00850     02 FILLER          PIC X(8)  VALUE 'QUANTITY'.
00860     02 FILLER          PIC X(69) VALUE SPACE.
00870*
00880 01  LINE-REC.
00890     02 FILLER          PIC X(14) VALUE SPACE.
00900     02 FILLER          PIC X(9)  VALUE '-----  '.
00910     02 FILLER          PIC X(16) VALUE '--------  '.
00920     02 FILLER          PIC X(8)  VALUE '------- '.
00930     02 FILLER          PIC X(8)  VALUE '------- '.
00940     02 FILLER          PIC X(8)  VALUE '------- '.
00950     02 FILLER          PIC X(69) VALUE SPACE.
00960*
00970 01  SELECT-OUT-REC.
00980     02 FILLER          PIC X(14) VALUE SPACE.
00990     02 O-PCODE         PIC X(5).
01000     02 FILLER          PIC X(2)  VALUE SPACE.
01010     02 O-KANJI         CHARACTER TYPE KEIS.
01020        03 O-PNAME         PIC N(8).
01030        03 FILLER          PIC X(2)  VALUE SPACE.
01040        03 O-COLOR         PIC N(5).
01050        03 FILLER          PIC X(6)  VALUE SPACE.
01060        03 O-PRICE         PIC X(8)  JUST RIGHT.
01070        03 FILLER          PIC X(2)  VALUE SPACE.
01080        03 O-STOCK         PIC X(8)  JUST RIGHT.
01090        03 FILLER          PIC X(69) VALUE SPACE.
01100 77  O-PCODE-NULL       PIC X(5) VALUE '*****'.
01110 77  O-PNAME-NULL       PIC N(10) VALUE NC'----------'.
01120 77  O-COLOR-NULL       PIC N(5) VALUE NC'-----'.
01130 77  O-PRICE-NULL       PIC X(8) VALUE '********'.
01140 77  O-STOCK-NULL        PIC X(8) VALUE '********'.
01150*
01160 01  I-CARD-ERROR-REC.
01170     02 FILLER          PIC X(14) VALUE SPACE.
01180     02 FILLER          PIC X(41)
01190         VALUE '***  ERROR  ***  NO CARD FOR CONNECT  ***'.
01200     02 FILLER          PIC X(77) VALUE SPACE.
01210*
01220 01  CONNECT-ERROR-REC.
01230     02 FILLER          PIC X(14) VALUE SPACE.
01240     02 FILLER          PIC X(45)
01250         VALUE '***  ERROR  ***  CANNOT CONNECT  ***  CODE
```

```
= '.
01260     02 CNCT-EC          PIC X(5).
01270     02 FILLER           PIC X(68) VALUE SPACE.
01280*
01290 01  NORMAL-END-REC.
01300     02 FILLER           PIC X(14) VALUE SPACE.
01310     02 FILLER           PIC X(22)
01320           VALUE '***  NORMAL ENDED  ***'.
01330     02 FILLER           PIC X(96) VALUE SPACE.
01340*
01350 01  SQLERR-PRINT-REC.
01360     02 FILLER           PIC X(14) VALUE SPACE.
01370     02 FILLER           PIC X(34)
01380           VALUE '***  HiRDB SQL ERROR MESSAGE-ID = '.
01390     02 RC-MSGID         PIC X(8).
01400     02 FILLER           PIC X(14) VALUE ' SQLERRORMC ='.
01500     02 RC-SQLERRMC      PIC X(62).
01510*
01520 01  WSQLCODE            PIC -(10)9.
01530*
01540 01  WMSGID.
01550     02 FILLER           PIC X(8).
01560     02 MSGID            PIC X(3).
01570*
01580 01  ERRORMSGID.
01590     02 FILLER           PIC X(5) VALUE 'KFPA1'.
01600     02 E-MSGID          PIC X(4).
01610     02 FILLER           PIC X(2) VALUE '-E'.
01620*
01630 01  EOF                 PIC X(1) VALUE '0'.
01640 01  ERR-FLG             PIC X(1) VALUE '0'.
01650*
01660*
01670 PROCEDURE DIVISION.
01680 MAIN SECTION.
01690 M-1.
01700     OPEN INPUT  INPUT-CARD-FILE
01710             OUTPUT PRINT-STOCK-FILE.
01720     READ INPUT-CARD-FILE
01730           AT END
01740             MOVE '1' TO ERR-FLG
01750             GO TO M-3
01760     END-READ.
01770     MOVE IUSERID TO XUSERID.
01780     MOVE IPSWD TO XPSWD.
01790*
01800     EXEC SQL                                            (a) 2
01810          WHENEVER SQLERROR                              (a) 2
```

```
01820             GO TO M-2                                (a) 2
01830     END-EXEC.                                        (a) 2
01840     EXEC SQL                                         (b) 2
01850          CONNECT USER :XUSERID USING :XPSWD          (b) 2
01860     END-EXEC.                                        (b) 2
01870     PERFORM CHANGE.
01880     GO TO M-3.
01890 M-2.
01900     MOVE '2' TO ERR-FLG.
01910*
01920 M-3.
01930     EVALUATE ERR-FLG
01940       WHEN '0'
01950         PERFORM NORMAL
01960       WHEN '1'
01970         WRITE PRINT-STOCK-REC
01980           FROM I-CARD-ERROR-REC
01990           AFTER ADVANCING 2 LINES
02000       WHEN '2'
02010         MOVE SQLCODE  TO CNCT-EC
02020         WRITE PRINT-STOCK-REC
02030           FROM CONNECT-ERROR-REC
02040           AFTER ADVANCING 2 LINES
02050       WHEN '3'
02060         PERFORM ERROR
02070     END-EVALUATE.
02080 M-4.
02090     CLOSE INPUT-CARD-FILE
02100           PRINT-STOCK-FILE.
02110 M-EX.
02120     EXEC SQL
02130        WHENEVER SQLERROR   CONTINUE
02140     END-EXEC.
02150     EXEC SQL
02160        WHENEVER NOT FOUND  CONTINUE
02170     END-EXEC
02180     EXEC SQL
02190        WHENEVER SQLWARNING CONTINUE
02200     END-EXEC.
02210    EXEC SQL
02220        DISCONNECT
02230     END-EXEC.
02240     GOBACK.
02250 CHANGE SECTION.
02260 H-1.
02270     READ INPUT-CARD-FILE
02280       AT END
02290         MOVE '1' TO ERR-FLG
```

```
02300     END-READ.
02310     EXEC SQL
02320          WHENEVER SQLERROR
02330            GO TO H-2
02340     END-EXEC.
02350     PERFORM UNTIL EOF = '1' OR ERR-FLG NOT = '0'
02360       EVALUATE ITYPE
02370         WHEN 'I'
02380           PERFORM ADDITION
02390         WHEN 'U'
02400           PERFORM UPDATE
02410         WHEN 'D'
02420           PERFORM DELETION
02430       END-EVALUATE
02440       READ INPUT-CARD-FILE
02450         AT END
02460           MOVE '1' TO EOF
02470       END-READ
02480     END-PERFORM.
02490     GO TO H-EX.
02500 H-2.
02510     MOVE '3' TO ERR-FLG.
02520 H-EX.
02530     EXIT.
02540*
02550 ADDITION SECTION.
02560 T-1.
02570     MOVE IPCODE  TO XPCODE.
02580     MOVE IPNAME  TO XPNAME.
02590     MOVE ICOLOR  TO XCOLOR.
02600     MOVE IPRICE  TO XPRICE.
02610     MOVE ISTOCK  TO XSTOCK.
02620     EXEC SQL
02610           WHENEVER SQLERROR GO TO T-2
02620     END-EXEC.
02630    EXEC SQL                                             3
02640           INSERT INTO STOCK(PCODE, PNAME, COLOR, PRICE,
SQUANTITY)   3
02650              VALUES(:XPCODE, :XPNAME, :XCOLOR, :XPRICE,
:XSTOCK)   3
02660    END-EXEC.                                            3
02670     GO TO T-EX.
02680 T-2.
02690     MOVE '3' TO ERR-FLG.
02700 T-EX.
02710     EXIT.
02720 UPDATE SECTION.
02730 K-1.
```

```
02740     MOVE IPCODE TO XPCODE.
02750     MOVE ISTOCK TO XSTOCK.
02760     EXEC SQL
02770           WHENEVER SQLERROR GO TO K-2
02780     END-EXEC.
02790     EVALUATE IFLUX
02800      WHEN '1'                                           4
02810        EXEC SQL                                     (a) 4
02820               UPDATE STOCK SET SQUANTITY = SQUANTITY +
:XSTOCK    (a) 4
02830               WHERE PCODE=:XPCODE                   (a) 4
02840        END-EXEC                                     (a) 4
02850      WHEN '2'                                           4
02860        EXEC SQL                                     (b) 4
02870               UPDATE STOCK SET SQUANTITY = SQUANTITY -
:XSTOCK    (b) 4
02880               WHERE PCODE=:XPCODE                   (b) 4
02890        END-EXEC                                     (b) 4
02900     END-EVALUATE.
02910     GO TO K-EX.
02920 K-2.
02930     MOVE '3' TO ERR-FLG.
02940 K-EX.
02950     EXIT.
02960*
02970 DELETION SECTION.
02980 S-1.
02990     MOVE IPCODE TO XPCODE.
03010     EXEC SQL
03020           WHENEVER SQLERROR GO TO S-2
03030     END-EXEC.
03040    EXEC SQL                                             5
03050         DELETE FROM STOCK                               5
03060            WHERE PCODE=:XPCODE                          5
03070    END-EXEC.                                            5
03080     GO TO S-EX.
03090 S-2.
03100     MOVE '3' TO ERR-FLG.
03110 S-EX.
03120     EXIT.
03130*
03140 NORMAL SECTION.
03150 F-0.
03160     WRITE PRINT-STOCK-REC
03170        FROM HEADING-REC
03180        AFTER ADVANCING 4 LINES.
03190     WRITE PRINT-STOCK-REC
03200        FROM COLUMN-NAME-REC
```

```
03210        AFTER ADVANCING 2 LINES.
03220     WRITE PRINT-STOCK-REC
03230        FROM LINE-REC
03240        AFTER ADVANCING 2 LINES.
03250 F-1.
03260     EXEC SQL
03270           WHENEVER SQLERROR GO TO F-4
03280     END-EXEC.
03290    EXEC SQL                                         (a) 6
03300         DECLARE CR1 CURSOR FOR                      (a) 6
03310            SELECT PCODE,PNAME,COLOR,PRICE,SQUANTITY FROM
STOCK  (a) 6
03320    END-EXEC.                                        (a) 6
03330    EXEC SQL                                         (b) 6
03340         OPEN CR1                                    (b) 6
03350    END-EXEC.                                        (b) 6
03360 F-2.
03370    EXEC SQL                                         (a) 7
03380         WHENEVER NOT FOUND                          (a) 7
03390            GO TO F-3                                (a) 7
03400    END-EXEC.                                        (a) 7
03410    EXEC SQL                                         (b) 7
03420         FETCH CR1                                   (b) 7
03430            INTO :XPCODE:XIPCODE, :XPNAME:XIPNAME,
(b) 7
03440                :XCOLOR:XICOLOR, :XPRICE:XIPRICE,
:XSTOCK:XISTOCK  (b) 7
03450    END-EXEC.                                        (b) 7
03460     EXEC SQL
03470          WHENEVER NOT FOUND
03480             CONTINUE
03490     END-EXEC.
03500     IF XIPCODE IS >= 0 THEN
03510       MOVE XPCODE TO O-PCODE
03520     ELSE
03530       MOVE O-PCODE-NULL TO O-PCODE
03540     END-IF.
03550     IF XIPNAME IS >= 0 THEN
03560       MOVE XPNAME TO O-PNAME
03570     ELSE
03580       MOVE O-PNAME-NULL TO O-PNAME
03590     END-IF.
03600     IF XICOLOR IS >= 0 THEN
03610       MOVE XCOLOR TO O-COLOR
03620     ELSE
03630       MOVE O-COLOR-NULL TO O-COLOR
03640     END-IF.
03650     IF XIPRICE IS >= 0 THEN
```

```
03660       MOVE XPRICE TO O-PRICE
03670     ELSE
03680       MOVE O-PRICE-NULL TO O-PRICE
03690     END-IF.
03700     IF XISTOCK IS >= 0 THEN
03710       MOVE XSTOCK TO O-STOCK
03720     ELSE
03730       MOVE O-STOCK-NULL TO O-STOCK
03740     END-IF.
03750     WRITE PRINT-STOCK-REC
03760           FROM SELECT-OUT-REC
03770           AFTER ADVANCING 2 LINES.
03780     GO TO F-2.
03790 F-3.
03800     EXEC SQL
03810        WHENEVER SQLERROR   CONTINUE
03820     END-EXEC.
03830     EXEC SQL
03840        WHENEVER NOT FOUND  CONTINUE
03850     END-EXEC
03860     EXEC SQL
03870        WHENEVER SQLWARNING CONTINUE
03880     END-EXEC.
03890    EXEC SQL                                 (a) 8
03900       CLOSE CR1                             (a) 8
03910    END-EXEC.                                (a) 8
03920*
03930    EXEC SQL                                 (b) 8
03940       COMMIT                                (b) 8
03950    END-EXEC.                                (b) 8
03960*
03970     WRITE PRINT-STOCK-REC
03980           FROM NORMAL-END-REC
03990           AFTER ADVANCING 2 LINES.
04000     GO TO F-EX.
04010 F-4.
04020     PERFORM ERROR.
04030 F-EX.
04040     EXIT.
04050 ERROR SECTION.
04060 I-1.
04070     MOVE SQLCODE TO WSQLCODE.
04080     MOVE WSQLCODE TO WMSGID.
04090     MOVE MSGID TO E-MSGID.
04100     MOVE ERRORMSGID TO RC-MSGID.
04110     MOVE SQLERRMC TO RC-SQLERRMC.
04120     WRITE PRINT-STOCK-REC
04130           FROM SQLERR-PRINT-REC
```

```
04140             AFTER ADVANCING 2 LINES.
04150      EXEC SQL                                          (a) 9
04160         WHENEVER SQLERROR   CONTINUE                   (a) 9
04170      END-EXEC.                                         (a) 9
04180      EXEC SQL                                          (a) 9
04190         WHENEVER NOT FOUND  CONTINUE                   (a) 9
04200      END-EXEC.                                         (a) 9
04210      EXEC SQL                                          (a) 9
04220         WHENEVER SQLWARNING CONTINUE                   (a) 9
04230      END-EXEC.                                         (a) 9
04240      EXEC SQL                                          (b) 9
04250         ROLLBACK                                       (b) 9
04260      END-EXEC.                                         (b) 9
04270 I-EX.
04280      EXIT.
```

1. Starting and ending the embedded SQL declaration section

   Encloses the variables to be used in the UAP between BEGIN DECLARE SECTION and END DECLARE SECTION. These variables indicate the start and end of the embedded SQL declaration section.

2. Connecting with HiRDB

   (a)Specifying the abnormal processing

   Specifies the branch destination (M-2) as the process to be executed if an error (SQLERROR) occurs after execution of the subsequent SQL statements.

   (b)Connecting to HiRDB

   Informs HiRDB of the authorization identifier (XUSERID) and the password (XPSWD) so that the UAP can use HiRDB.

3. Inserting rows into the stock table

   Inserts the values read into the embedded variables into each column of the stock table.

4. Updating stock table rows

   (a)Incoming stock

   Sets the product code that was read into embedded variable :XPCODE as the key, and retrieves the row to be updated from the stock table. Updates the row by adding the value that was read into embedded variable :XQUANTITY to the QUANTITY value of the retrieved row.

   (b)Stock

   Sets the product code that was read into embedded variable :XPCODE as the key, and retrieves the row to be updated from the stock table. Updates the row by deleting the value that was read into embedded variable :XQUANTITY from the QUANTITY value of the retrieved row.

5. Deleting stock table rows

   Sets the product code that was read into embedded variable :XPCODE as the key, and deletes the rows having a key equal to that value.

6. Declaring and opening the CR1 cursor

   (a)Declaring the CR1 cursor

   Declares the CR1 cursor for retrieving rows from the stock table (STOCK).

   (b)Opening the CR1 cursor

   Positions the cursor immediately in front of a row to be retrieved from the stock table (STOCK) so that the row can be fetched.

7. Fetching stock table rows

   (a)Specifying the abnormal processing

   Retrieves the row indicated by the CR1 cursor from the stock table (STOCK), and sets the row values into the embedded variables.

   (b)Executing the FETCH statement

   Fetches the row indicated by the CR1 cursor from the stock table (STOCK), and sets the data to the embedded variables.

8. Terminating the transaction

   (a)Closing the CR1 cursor

   Closes the CR1 cursor.

   (b)Terminating the transaction

   Terminates the current transaction normally, and validates the results of the database addition, update, and deletion operations that were executed in that transaction.

9. Rolling back the transaction

   Specifying the processing

   Specifies continuation to the next instruction (without special processing) if an error (SQLERROR) or warning (SQLWARNING) occurs during execution of a subsequent SQL statement.

   Invalidating the transaction

   Rolls back the current transaction to invalidate the results of the database addition, update, and deletion operations that were executed in that transaction.

(2) Example that uses a row interface

(a) PAD chart

Figures 7-14 through *7-17* show the PAD chart for program example 5.

Figure 7-14: PAD chart for program example 5 (1/4)

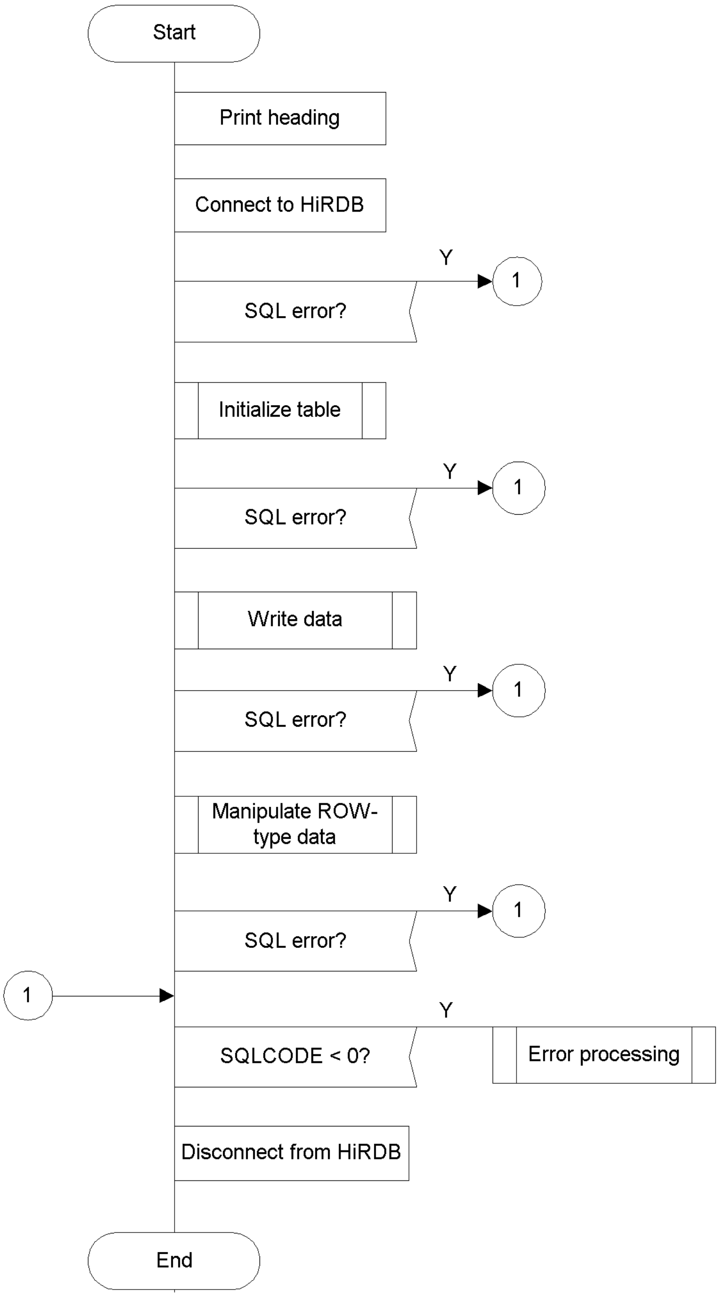
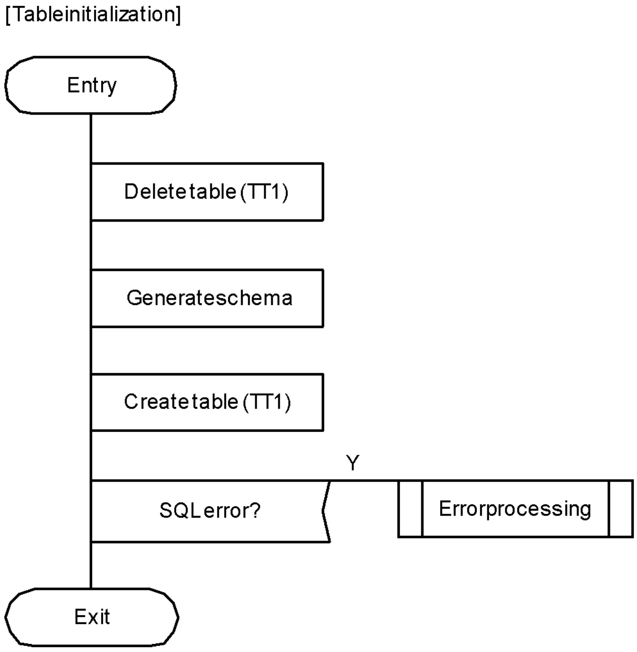


Figure 7-15: PAD chart for program example 5 (2/4)



[Errorprocessing]

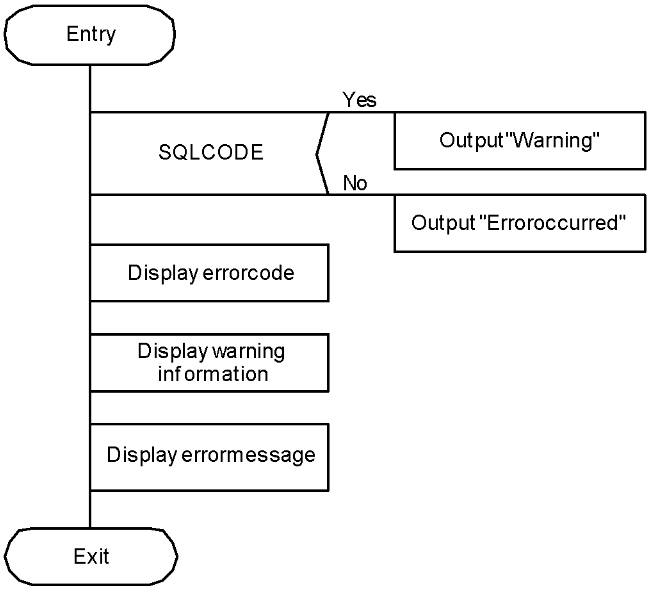


Figure 7-16: PAD chart for program example 5 (3/4)

[ROW-type data manipulation]

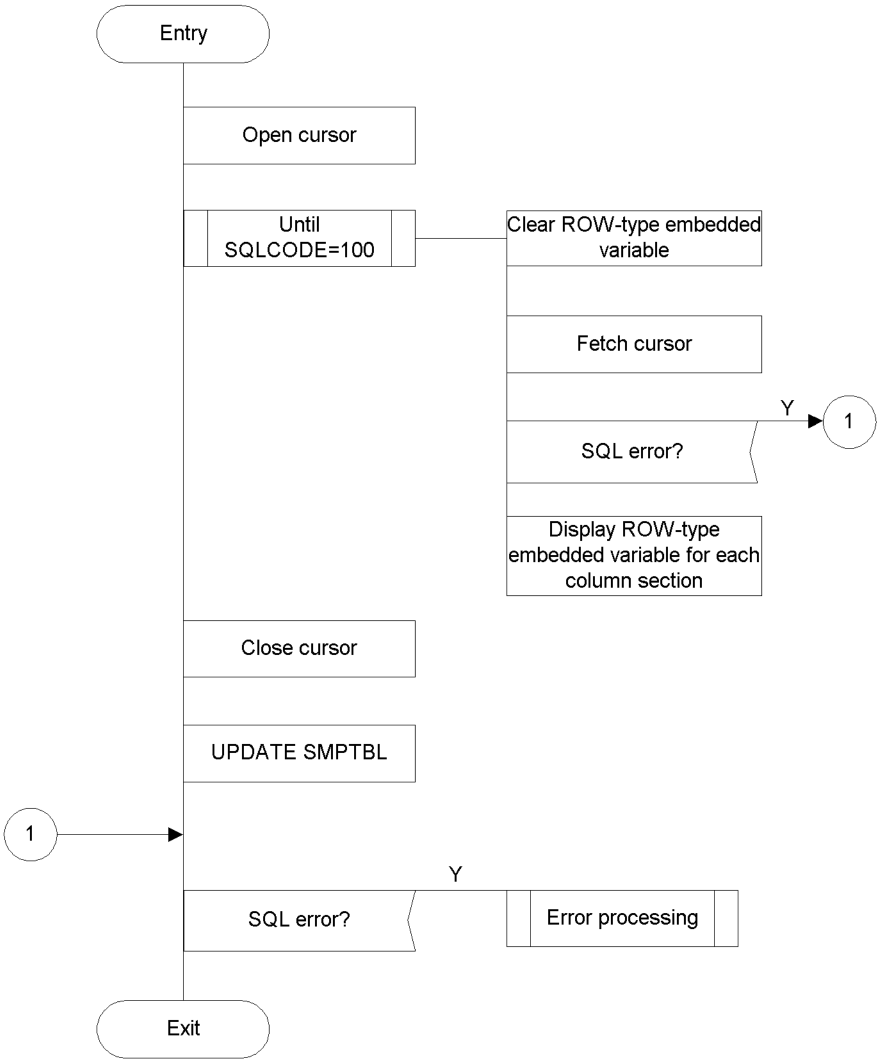
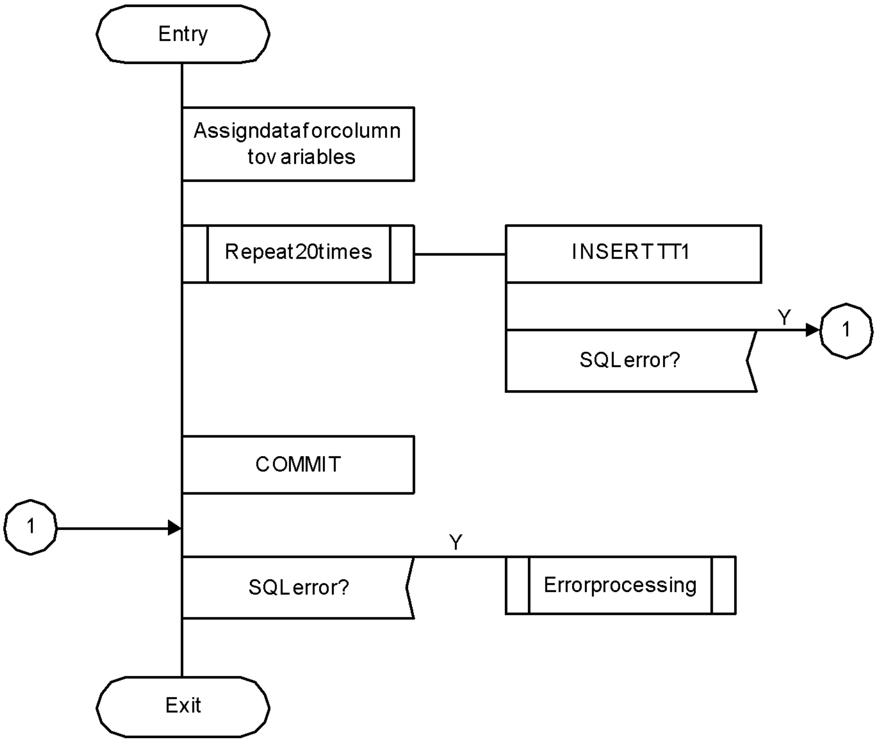


Figure 7-17: PAD chart for program example 5 (4/4)

[Datainsertion]



(b) Coding example

```
00010  **********************************************
00020  *                                            *
00030  *  EMBEDDED TYPE SQL  COBOL  UAP             *
00040  *  ROW INTERFACE SAMPLE                      *
00050  *                   1997/11/27               *
00060  **********************************************
00070   IDENTIFICATION DIVISION.
00080   PROGRAM-ID.            ROW-SAMPLE.
00090   AUTHOR.                CLIENT.
00100   DATA-WRITTEN.          1997/11/27.
00110   DATA-COMPILED.         ROW-SAMPLE.
00120   REMARKS.
00130   *
00140   ENVIRONMENT DIVISION.
00150   CONFIGURATION SECTION.
00160   SOURCE-COMPUTER. HITAC.
```

```
00170   OBJECT-COMPUTER. HITAC.
00180   INPUT-OUTPUT SECTION.
00190   FILE-CONTROL.
00200       SELECT OUTLIST ASSIGN TO LP.
00210   *
00220   DATA DIVISION.
00230   FILE SECTION.
00240   FD OUTLIST RECORDING MODE IS F
00250              LABEL RECORD IS OMITTED
00260              DATA RECORD OUTREC.
00270   01 OUTREC               PIC X(80).
00280   *
00290   WORKING STORAGE SECTION.
00300       EXEC SQL
00310         BEGIN DECLARE SECTION
00320       END-EXEC.
00330   01 IN-REC1 IS GLOBAL.
00340     02 IN-CHR1            PIC X(15)      VALUE 'EVA-00'.
00350     02 IN-INT1            PIC S9(9) COMP VALUE 255.
00360     02 IN-INT2            PIC S9(9) COMP VALUE 1.
00370
00380   01 XSQLROW IS GLOBAL1
00390     02 ROW-CHR1    PIC X(30).                    1
00400     02 ROW-INT1    PIC S9(9) COMP.               1
00410     02 ROW-INT2    PIC S9(9) COMP.               1
00420
00430     EXEC SQL
00440       END DECLARE SECTION
00450     END-EXEC.
00460
00470   01 DISP-REC IS GLOBAL.
00480     02 DISP-CHR1   PIC X(15).
00490     02 DISP-INT1   PIC S9(9).
00500     02 DISP-INT2   PIC S9(4).
00510   01 ERRFLG  PIC S9(4) COMP IS GLOBAL.
00520
00530   01 MSG-ERR   PIC X(10) VALUE '!! ERROR'.
00540   01 MSG-CODE IS GLOBAL.
00550      02 FILLER   PIC X(15) VALUE '!! SQLCODE ='
00560      02 MSG-SQLCODE  PIC S9(9) DISPLAY.
00570   01 MSG-MC IS GLOBAL.
00580      02 FILLER   PIC X(15) VALUE '!! SQLERRMC ='
00590      02 MSG-SQLERRMC PIC X(100).
00600
00610   PROCEDURE DIVISION.
00620   **********************************************
00630   *   DISPLAY TITLE
00640   **********************************************
```

```
MAIN SECTION
   CALL 'DISPLAY-TITLE'.
   MOVE ZERO TO ERRFLG.

**********************************************
*   CONNECT
**********************************************
EXEC SQL
  WHENEVER SQLERROR GOTO ERR-EXIT
END-EXEC

DISPLAY '***** CONNECT '.
EXEC SQL                                        2
  CONNECT                                       2
END-EXEC.                                       2
DISPLAY '***** CONNECT : END'.

**********************************************
*   INIT
**********************************************
    DISPLAY '## TABLE WILL BE INITIALIZED'.
    CALL 'INIT-TABLE'.
    IF ERRFLG < ZERO
       GO TO ERR-EXIT
    END-IF
    DISPLAY '## IS NORMAL'.

**********************************************
*   INSERT
**********************************************
    DISPLAY 'INSERT ## DATA'.
    CALL 'TEST-INSERT'.
    IF ERRFLG < ZERO
    GO TO ERR-EXIT
    END-IF
    DISPLAY '## IS NORMAL'.

**********************************************
*   ROW
**********************************************
    DISPLAY '## ROW TYPE TEST WILL BE EXECUTED'.
    CALL 'TEST-ROW'.
    IF ERRFLG < ZERO
    GO TO ERR-EXIT
    END-IF
    DISPLAY '## IS NORMAL'.

**********************************************
```

```
01130  *   DISCONNECT
01140  **********************************************
01150  ERR-EXIT.
01160       IF SQLCODE < ZERO
01170         MOVE SQLCODE TO MSG-SQLCODE
01180         MOVE SQLERRMC TO MSG-SQLERRMC
01190         DISPLAY MSG-ERR
01200         DISPLAY MSG-CODE
01210         DISPLAY MSG-MC
01220         MOVE -1 TO ERRFLG
01230     END-IF
01240
01250     EXEC SQL
01260       WHENEVER SQLERROR CONTINUE
01270     END-EXEC
01280     EXEC SQL
01290       WHENEVER NOT FOUND CONTINUE
01300     END-EXEC
01310     EXEC SQL
01320       WHENEVER SQLWARNING CONTINUE
01330     END-EXEC
01340
01350     DISPLAY '##DISCONNECT'
01360
01370     EXEC SQL                                    3
01380       DISCONNECT                                3
01390     END-EXEC                                    3
01400     STOP RUN.
01410
01420  **********************************************
01430  *   INSERT STATEMENT TEST
01440  **********************************************
01450  IDENTIFICATION DIVISION.
01460  PROGRAM-ID. TEST-INSERT.
01470  DATA DIVISION.
01480  WORKING-STORAGE SECTION.
01490    01 DCNT PIC S9(9) COMP.
01500  PROCEDURE DIVISION.
01510       EXEC SQL
01520         WHENEVER SQLERROR GOTO :Exit-Test-Insert
01530       END-EXEC.
01540  **********************************************
01550  *   INSERT HOST
01560  **********************************************
01570       DISPLAY 'INSERT start WITH ***** EMBEDDED
            VARIABLE'
01580       MOVE ZERO TO DCNT.
01590  INSERT-LOOP.
```

```
01600        COMPUTE IN-INT1 = DCNT
01610        COMPUTE IN-INT2 = DCNT + 100
01620        COMPUTE DCNT = DCNT + 1
01630        EXEC SQL                                  4
01640          INSERT INTO TT1(CLM1,                    4
01650                          CLM2,                    4
01660                          CLM3)                    4
01670           VALUES (:IN-CHR1,                      4
01680                   :IN-INT1,                      4
01690                   :IN-INT2)                      4
01700        END-EXEC                                  4
01710        IF DCNT <20 THEN
01720          GO TO INSERT-LOOP
01730        END-IF
01740        DISPLAY '***** insert : SUCCESS'.
01750   **********************************************
01760   *
01770   **********************************************
01780   EXIT-TEST-INSERT.
01790        IF SQLCODE < ZERO
01800          MOVE SQLCODE TO MSG-SQLCODE
01810          MOVE SQLERRMC TO MSG-SQLERRMC
01820          DISPLAY MSG-CODE
01830          DISPLAY MSG-MC
01840          MOVE -1 TO ERRFLG
01850       END-IF
01860       DISPLAY '>> TEST-INSERT<<'
01870       GOBACK.
01880   **********************************************
01890   *   WARNING
01900   **********************************************
01910   INSERT-WARNING.
01920       DISPLAY 'WARNING'
01930       MOVE SQLCODE TO MSG-SQLCODE
01940       MOVE SQLERRMC TO MSG-SQLERRMC
01950       DISPLAY MSG-CODE
01960       DISPLAY MSG-MC,
01970   END PROGRAM TEST-INSERT.
01980
01990   **********************************************
02000   *    TEST ROW
02010   **********************************************
02020   IDENTIFICATION DIVISION.
02030   PROGRAM-ID. TEST-ROW.
02040   DATA DIVISION.
02050   WORKING-STORAGE SECTION.
02060   PROCEDURE DIVISION.
02070          DISPLAY '***** ROW CURSOR OPEN'
```

```
02080          EXEC SQL                                5
02090            DECLARE CUR_ROW CURSOR FOR            5
02100               SELECT ROW FROM TT15
02110               WHERE CLM2 = 10                    5
02120               FOR UPDATE OF CLM35
02130          END-EXEC                                5
02140   **********************************************
02150   *   ROW CURSOR
02160   **********************************************
02170           DISPLAY '***** ROW CURSOR OPEN'.
02180           EXEC SQL
02190             WHENEVER SQLERROR GOTO :Exit-Test-Row
02200          END-EXEC
02210          EXEC SQL                                6
02220            OPEN CUR_ROW                          6
02230          END-EXEC                                6
02240
02250   **********************************************
02260   *   FETCH ROW CURSOR
02270   **********************************************
02280       DISPLAY '***** ROW CURSOR FETCH'
02290       EXEC SQL
02300         WHENEVER NOT FOUND GOTO :Exit-Test-ROW
02310       END-EXEC
02320       EXEC SQL
02330         WHENEVER SQLERROR GOTO :Exit-Test-ROW
02340       END-EXEC
02350       MOVE SPACE TO XSQLROW
02360       EXEC SQL                                   7
02370         FETCH CUR_ROW INTO :XSQLROW              7
02380       END-EXEC                                   7
02390       DISPLAY '## FETCH DATA'
02400       MOVE ROW-CHR1 TO DISP-CHR1
02410       MOVE ROW-INT1 TO DISP-INT1
02420       MOVE ROW-INT2 TO DISP-INT2
02430       DISPLAY DISP-REC
02440
02450       DISPLAY '***** ROW UPDATE'
02460       MOVE 'ANGEL' TO ROW-CHR1
02470       EXEC SQL                                   8
02480         UPDATE TT1 SET ROW = :XSQLROW            8
02490          WHERE CURRENT OF CUR_ROW                8
02500       END-EXEC                                   8
02510
02520   **********************************************
02530   *   FETCH ROW CURSOR
02540   **********************************************
02550       DISPLAY '***** ROW CURSOR CLOSE'
```

```
02560       EXEC SQL
02570         WHENEVER NOT FOUND CONTINUE
02580       END-EXEC
02590       EXEC SQL
02600         WHENEVER SQLERROR CONTINUE
02610       END-EXEC
02620       EXEC SQL                                    9
02630         CLOSE CUR_ROW                             9
02640       END-EXEC.                                   9
02650   **********************************************
02660   *
02670   **********************************************
02680   EXIT-TEST-ROW.
02690        IF SQLCODE < ZERO THEN
02700          MOVE SQLCODE TO MSG-SQLCODE
02710          MOVE SQLERRMC TO MSG-SQLERRMC
02720          DISPLAY MSG-CODE
02730          DISPLAY MSG-MC
02740          MOVE -1 TO ERRFLG
02750        END-IF
02760        EXEC SQL
02770          WHENEVER NOT FOUND CONTINUE
02780        END-EXEC
02790        EXEC SQL
02800          WHENEVER SQLERROR CONTINUE
02810        END-EXEC
02820        EXEC SQL
02830          COMMIT
02840        END-EXEC
02850        DISPLAY '>> TEST-ROW END <<'
02860        GOBACK.
02870
02880   **********************************************
02890   *   WARNING
02900   **********************************************
02910   ROW-WARNING.
02920       DISPLAY 'WARNING'
02930       MOVE SQLCODE TO MSG-SQLCODE
02940       MOVE SQLERRMC TO MSG-SQLERRMC
02950       DISPLAY MSG-CODE
02960       DISPLAY MSG-MC.
02970   END PROGRAM TEST-ROW.
02980
02990
03000   ************************************************
03010   *   INITIALIZE TABLE
03020   ************************************************
03030   IDENTIFICATION DIVISION.
```

```
03040   PROGRAM-ID. INIT-TABLE.
03050   DATA DIVISION.
03060   WORKING-STORAGE SECTION.
03070   PROCEDURE DIVISION.
03080      EXEC SQL
03090        WHENEVER SQLERROR CONTINUE
03100      END-EXEC
03110
03120   ************************************************
03130   *   DROP TABLE
03140   ************************************************
03150            DISPLAY '***** DROP TABLE'.
03160            EXEC SQL                               10
03170              DROP TABLE TT1                       10
03180            END-EXEC                               10
03190            DISPLAY '***** CREATE SCHEMA'.
03200            EXEC SQL                               11
03210              CREATE SCHEMA                        11
03220            END-EXEC                               11
03230
03240   ************************************************
03250   *   COMMIT
03260   ************************************************
03270           DISPLAY '***** COMMIT START'.
03280           EXEC SQL
03290             WHENEVER SQLERROR GOTO EXIT-INIT-TABLE
03300           END-EXEC
03310           EXEC SQL
03320             COMMIT
03330           END-EXEC
03340           DISPLAY '***** COMMIT : END '.
03350
03360   ************************************************
03370   *   CREATE TABLE
03380   ************************************************
03390           DISPLAY '***** create table'.
03400           EXEC SQL                                12
03410              CREATE FIX TABLE TT1(CLM1 CHAR(30),  12
03420                                   CLM2 INTEGER,   12
03430                                   CLM3 INTEGER)   12
03440           END-EXEC                                12
03450
03460           DISPLAY '***** create table : SUCCESS'.
03470
03480   ************************************************
03490   *
03500   ************************************************
03510   EXIT-INIT-TABLE.
```

```
  IF SQLCODE < ZERO THEN
    MOVE SQLCODE TO MSG-SQLCODE
    MOVE SQLERRMC TO MSG-SQLERRMC
    DISPLAY MSG-CODE
    DISPLAY MSG-MC
    MOVE -1 TO ERRFLG
  END-IF
  GOBACK.

************************************************
*   WARNING
************************************************
INIT-TABLE-WARNING.
    DISPLAY 'WARNING'
    MOVE SQLCODE TO MSG-SQLCODE
    MOVE SQLERRMC TO MSG-SQLERRMC
    DISPLAY MSG-CODE
    DISPLAY MSG-MC.
END PROGRAM INIT-TABLE.

************************************************
*   DISPLAY
************************************************
IDENTIFICATION DIVISION.
PROGRAM-ID. DISPLAY-TITLE.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
   DISPLAY '###################################'
   DISPLAY '#                                 #'
   DISPLAY '# THIS PROGRAM IS A SAMPLE        #'
   DISPLAY '# PROGRAM FOR THE ROW-TYPE        #'
   DISPLAY '# INTERFACE                       #'
   DISPLAY '###################################'
END PROGRAM DISPLAY-TITLE.
END PROGRAM ROW-SAMPLE.
```

1. Declaring a ROW-type embedded variable

   Declares the embedded variable (:XSQLROW) to be used by the row interface.

2. Connecting to HiRDB

   Uses the authorization identifier and password set in the PDUSER environment variable to connect to the server.

3. Disconnecting from HiRDB

   Disconnects the UAP from the server.

4. Adding rows

   Adds data to the FIX table (TT1).

5. Declaring the CUR_ROW cursor

   Declares the CUR_ROW cursor, because the row interface will be used to retrieve the FIX table (TT1).

6. Opening the CUR_ROW cursor

   Positions the cursor immediately in front of a row to be retrieved from the FIX table (TT1) so that the row can be fetched.

7. Fetching rows

   Fetches the row indicated by the CUR_ROW cursor from the FIX table (TT1), and sets the value to the embedded variable (:XSQLROW).

8. Updating rows

   Updates the FIX table (TT1) row where the CUR_ROW cursor is positioned with the embedded variable (:XSQLROW) value.

9. Closing the CUR_ROW cursor

   Closes the CUR_ROW cursor.

10. Dropping tables (TT1)

    Deletes any existing tables of the same name so that the FIX table (TT1) can be created.

11. Creating a schema

    Creates a schema in case there are no schemas.

12. Creating the FIX table (TT1)

    Creates the FIX table (TT1). The row interface can be used only for tables that have the FIX attribute.

(3) Example that uses the TYPE, TYPEDEF, and SAME AS clauses

A coding example that uses the TYPE, TYPEDEF, and SAME AS clauses follows:

```
000100 IDENTIFICATION  DIVISION.
000200 PROGRAM-ID.     CBL001.
000300 DATA            DIVISION.
000400 WORKING-STORAGE SECTION.
000500     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
000600* -- type declaration --
000700 01  VCHR20 TYPEDEF.
000800     05  LEN  PIC S9(4) COMP.
```

```
000900     05  STR  PIC X(20).
001000
001100* -- data declaration --
001200 01  D-4C.
001300     05  XCUT     TYPE VCHR20.
001400     05  XCOLOR   PIC X(10).
001500     05  XCLARITY SAME AS XCOLOR.
001600     05  XCARAT   PIC S9(4) COMP.
001700
001800     EXEC SQL END DECLARE SECTION END-EXEC.
   :                 :
   :                 :
002000 PROCEDURE DIVISION.
002100 CB_001 SECTION.
   :                 :
   :                 :
003400 INS-1.
003500       EXEC SQL
003600        INSERT INTO A_DIM (C1, C2, C3, C4)
003700            VALUES (:XCUT, :XCOLOR, :XCLARITY, :XCARAT)
003800       END-EXEC.
   :                :
   :                :
005000 INS-EX.
005100     EXIT.
005200 END PROGRAM CBL001.
```

7.4 Writing a UAP in C++

This section explains the coding rules for embedding SQL statements in UAPs written in C++.

7.4.1 Coding rules

When a UAP is written, the labeling rules, SQL coding rules, and SQL syntax rules must be followed.

(1) Labeling rules

The labeling rules are basically same as for C. These types of labels cannot be used:

- Labels that begin with uppercase letters `SQL`
- Labels that begin with lowercase letter `p_`
- Labels that begin with lowercase letters `pd`

For naming embedded variables, indicator variables, and branching destination labels, the labeling rules and the C rules must be followed.

(2) SQL coding rules

- The notation `//` can be used to indicated a comment statement.
- Members of an object cannot be used as embedded variables.
- An object method cannot be specified in the `WHENEVER` statement.
- An SQL statement cannot be coded in a class definition.

All other coding rules are the same as for C. For details, see *7.2.1(2) SQL coding rules*.

7.5 Writing a UAP in OOCOBOL

This section explains the coding rules for embedding SQL statements in UAPs written in OOCOBOL.

7.5.1 Coding rules

When a UAP is written, the labeling rules, SQL coding rules, and SQL syntax rules must be followed.

(1) Labeling rules

The labeling rules are basically same as for COBOL. These rules apply to labels:

(a) SQL reserved words

- Both uppercase and lowercase letters can be used
- Uppercase and lowercase letters can be mixed

(b) Host names

- Labels that begin with `SQL` cannot be used
- Spaces can be entered following a colon within a host name
- Host names are not case-sensitive
- Uppercase and lowercase letters can be mixed

Embedded variables, indicator variables, and branching destination labels must be named in accordance with the COBOL labeling rules.

The following labels, which have the external attribute, cannot be used:

- Labels that begin with the uppercase letters `SQL`
- Labels that begin with the lowercase letter `p_`
- Labels that begin with the lowercase letters `pd`

(2) SQL coding rules

1. Members of an object cannot be used as embedded variables.
2. An object method cannot be specified in the `WHENEVER` statement.
3. An SQL statement cannot be coded in a class definition.

All other coding rules are the same as for COBOL. For details, see *7.3.1(2) SQL coding rules*.

7.6 Creating a UAP in 64-bit mode

This section describes how to create at a HiRDB client a UAP that supports 64-bit mode.

(1) Languages and functions available to 64-bit-mode UAPs

(a) Languages

You can use C language, C++ language, and COBOL (COBOL2002) to create UAPs. OOCOBOL is not supported.

(b) Functions

The XA interfaces are not supported. Other functions are fully supported.

The multi-connection facility provides real threads, not pseudo-threads.

(2) Differences in SQL Communications Area

The structure of the SQL Communications Area changes when you set your HiRDB in 64-bit mode. The lengths of the communications area names also change, as shown in the table below. For details about the SQL Communications Area, see Appendix *A. SQL Communications Area*.

Table 7-4: Communications area names that change in 64-bit mode

Communications area name	**Length (bytes)**	
	32-bit mode	**64-bit mode**
SQLCA	336	368
SQLCABC	4	8
SQLCODE	4	8
SQLERRD	4 x 6	8 x 6

(3) Differences in the SQL descriptor area

The structure of SQL descriptor areas changes when you set your HiRDB in 64-bit mode. The lengths of the descriptor area names also change, as shown in the table below. For details about the SQL descriptor areas, see Appendix *B. SQL Descriptor Areas*.

Table 7-5: Descriptor area names that change in 64-bit mode

Descriptor area name	32-bit mode		64-bit mode	
	Length (bytes)	Data type	Length (bytes)	Data type
SQLDA	16 + 16 x *n*	--	24 + 24 x *n*	--
SQLDABC	4	--	8	--
SQLVAR	16 x *n*	--	24 x *n*	--
SQLVAR_LOB	16 x *n*	--	24 x *n*	--
SQLLOBLEN	--	`long`	--	`int`
SQLDATA	4	--	8	--
SQLIND	4	--	8	--
SQLLOBIND	4	`long *`	8	`int *`

Legend:

n: Number of `SQLVAR`s specified in the descriptor area name `SQLN`

--: There is no change to the length or data type.

(4) Differences between SQL data types and C data description

In a UAP written in C language when 64-bit mode is supported, the size of the `long` type is 8 bytes. Therefore, embedded variables that have been using `long` will use `int` instead of `long`. This change affects the data descriptions in C language as shown in the table below. For details about the data descriptions in C language, see Appendix *F.1 SQL data types and C data descriptions*.

Table 7-6: C data descriptions that change in 64-bit mode

SQL data type		C data description	Item specification	Remarks
`INTEGER`	Simple format	`int` *variable-name*`;`	`variable`	None
	Array format	`int` *variable-name*`[`*n*`];`	`array`	1 ≤ *n* ≤ 4096
Indicator variable for BLOB		`int` *indicator-variable-name* `;`	--	None

SQL data type	C data description	Item specification	Remarks

SQL statement	`struct{`
	`  int len;`
	`  char str[`*n*`];`
	`}`*variable-name*`;`
		`structure`	None

Legend:

n: Length (bytes)

--: Cannot be specified

(5) Differences in creating UAPs that use hash functions for table partitioning (applicable to HP-UX and Linux editions only)

When you compile and link a UAP that uses hash functions for table partitioning, the shared libraries to be specified are different. For details about how to create a UAP that uses hash functions for table partitioning, see Appendix *H.1 Hash function for table partitioning*.

(6) Migrating a HiRDB client from 32-bit mode to 64-bit mode

In order to migrate a HiRDB client from 32-bit mode to 64-bit mode, you must upgrade your HiRDB client to 64-bit mode edition (by installing 64-bit mode edition of HiRDB client and then setting up the client environment). For details about client environment setup, see *6. Client Environment Setup*.

When you install the 64-bit mode edition of HiRDB client, files for 64-bit mode are created. For details about the files that are created during installation, see *6.4 Organization of directories and files for a HiRDB client*.

Once you have set up the client environment, you must make changes to the UAPs so that they will function in 64-bit mode. The procedure is explained below:

Procedure

1. If the `long` type is specified in the declaration of an embedded variable, change it to the `int` type.
2. Preprocess the UAP. In this step, specify the option for generating a 64-bit-mode post source (`-h64` in UNIX, `/h64` in Windows).
3. Compile the UAP. In this step, specify the option for generating a 64-bit-mode object.
4. Link the UAP. In this step, specify the 64-bit-mode client libraries to be linked.

Note

For details about preprocessing, compiling, and linking, see *8.2 Preprocessing* and *8.3 Compiling and linking*.

Chapter

8. Preparation for UAP Execution

This chapter explains the preparations required before a UAP is executed.

This chapter contains the following sections:

8.1 UAP execution procedure
8.2 Preprocessing
8.3 Compiling and linking
8.4 Notes

8.1 UAP execution procedure

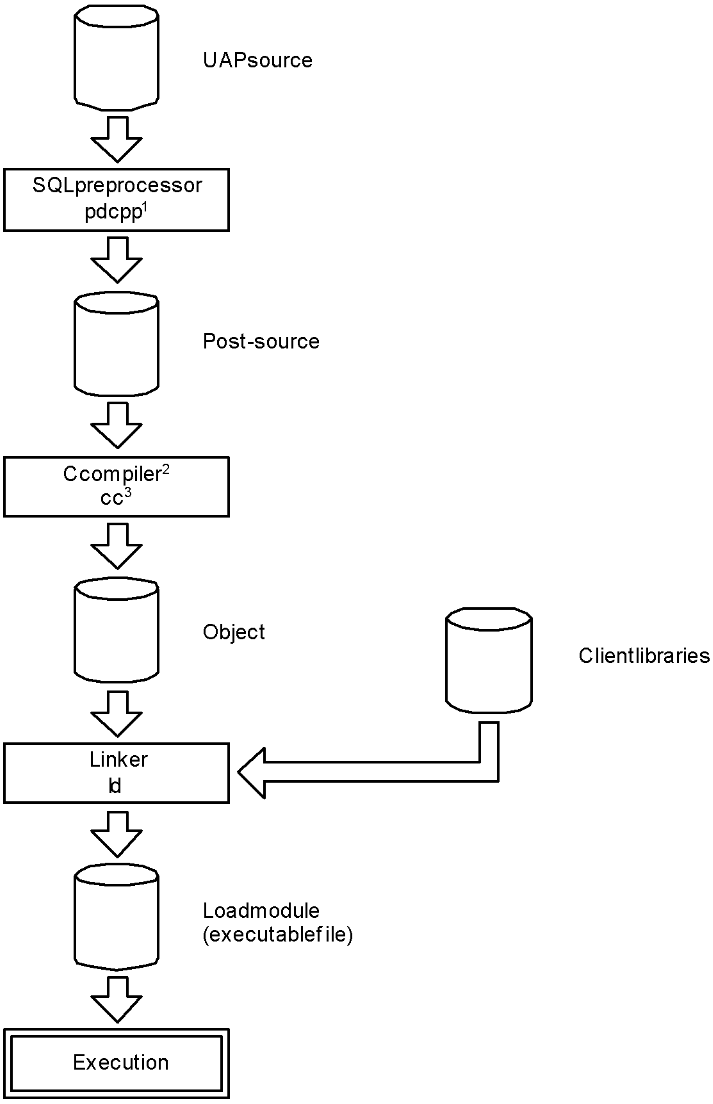
A UAP in which SQL statements are embedded cannot be executed directly. This section explains the procedure for executing such a UAP.

8.1.1 Executing a UAP written in C

A UAP in which SQL statements are embedded in a source program written in C must be converted to a post-source program by an SQL preprocessor. The resulting post source becomes a load module (executable file) when it is compiled and linked by a dedicated language compiler.

The following figure shows the procedure for executing a UAP written in C.

Figure 8-1: Execution procedure for UAP written in C



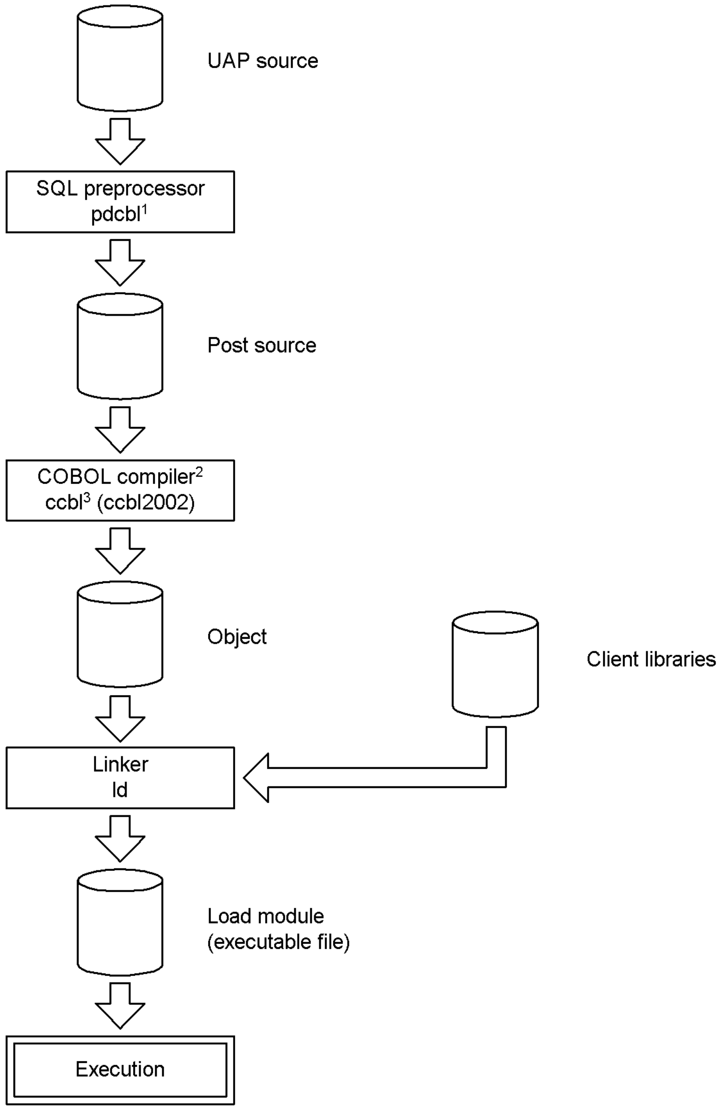
[1] ForC++, theSQLpreprocessorispdocc.
[2] For C++,theC++ compileris used.
[3] ForC++, thecompilerisCC.

8.1.2 Executing a UAP written in COBOL

A UAP in which SQL statements are embedded in a source program written in COBOL must be converted to a post-source program by an SQL preprocessor. The resulting post source becomes a load module (executable file) when it is compiled and link edited by the COBOL85 compiler.

The following figure shows the procedure for executing a UAP written in COBOL.

Figure 8-2: Execution procedure for a UAP written in COBOL



[1] For OOCOBOL, the SQL preprocessor is pdocb.
[2] For OOCOBOL, the OOCOBOL compiler is used.
[3] For OOCOBOL, the compiler is ocbl.

8.2 Preprocessing

8.2.1 Overview

(1) What is preprocessing?

Because a UAP source file in which SQL statements are embedded cannot be executed directly, an SQL preprocessor must be executed. Executing the SQL preprocessor converts the embedded SQL statements into a high-level language description. This process of converting a UAP source to a post source that can be converted by a language compiler by executing the SQL preprocessor is called *preprocessing*.

The SQL preprocessor creates a high-level language function corresponding to the SQL statements and embeds it in the source. During this process, the SQL preprocessor checks the validity of the data types and values of the variables, as well as the syntax of various names. If the checking results indicate that an error was detected in the input source program, a message to that effect is output to the standard error output.

(2) Items that are not checked by the SQL preprocessor

The SQL preprocessor does not check the following items:

- If there are any table names for which a query needs to be addressed to the server
- If there are any column names for which a query needs to be addressed to the server
- If there are other identifiers, data types, or functions for which a query needs to be addressed to the server
- Table access privileges

(3) Notes about preprocessing

1. The setting of environment variables and the command specification methods differ depending on the language used in the source program and the environment. The SQL preprocessor must therefore match the language and the environment being used.
2. In the Windows environment, the SQL preprocessor cannot perform a rigorous SQL syntax check unless `/Xp` is specified. Therefore, the preprocessor may not be able to detect all syntax errors in the SQL statements. Also, with Linux for AP8000 and AIX clients, the SQL preprocessor cannot perform a rigorous SQL syntax check if a character code type other than SJIS is used.
3. In the Windows environment, only `sjis` and `lang-c` character codes can be used because `EUC` codes cannot be recognized. If a value other than `sjis` or `lang-c` is specified as the character code type in the HiRDB server, an error occurs when

a HiRDB client executes a UAP.

8.2.2 Preprocessing in UNIX

(1) C

(a) Environment variable setting

The following environment variable can be set before a UAP is preprocessed:

`PDDIR`

This environment variable specifies the absolute path name for the installation directory for HiRDB (server or client); the default directory is `/HiRDB`.

This variable need not be specified when the installation destination is `/HiRDB`.

`LANG`

This environment variable specifies the character code classification of the HiRDB client environment. The following figure shows the character codes that can be specified in `LANG`.

Table 8-1: Character codes that can be specified for LANG

Server character code classification set with pdsetup command	**Value of LANG environment variable**			
	HP-UX	**Solaris**	**AIX**	**Linux**[#1]
`sjis`	`ja_JP.SJIS`	`ja_JP.PCK`	`Ja_JP`	Any value[#2]
`chinese`	`chinese-s`	`chinese-s`	`chinese-s`	`chinese-s`
`ujis`	`ja_JP.eucJP`	`ja`	`ja_JP`	`ja_JP.eucJP`, `ia_JP`, or `Ja_JP.ujis`
`lang-c`	Single-byte character codes of each language			
`utf-8`	--			
`chinese-gb18030`	--			
No specification	`ja_JP.SJIS`	`ja`	`Ja_JP`	`ja_JP.eucJP`, `ia_JP`, or `Ja_JP.ujis`

Legend:

--: Specify the `PDCLTLANG` environment variable instead of `LANG`.

Note

If the character code classification of the character strings described in a UAP does not match the character code classification for UAP execution, the UAP does not operate correctly. Therefore, the value of the `LANG` environment variable specified for UAP creation must be the same as the `LANG` value specified for UAP execution.

#1: For locales that are not supported by HiRDB, `lang-c` is assumed.

#2: Hitachi recommends that you set the `LANG` environment variable to a character code classification that can be used by the corresponding HiRDB server. If the HiRDB server cannot use the target character code classification, specify the `lang-c` value.

If this environment variable is not specified, or if the specified value is different from the value that was set to the environment variable, `ja_SP.SJIS` is assumed. In the HP-UX environment, only `ja_JP.SJIS` can be specified.

For details about the `pdsetup` command, see the *HiRDB Version 9 Command Reference* manual.

`PDCLTLANG`

This environment variable specifies the character code classification to be used in preprocessing instead of the character code classification specified by the `LANG` environment variable. For details about the `PDCLTLANG` operand, see *6.6.4 Environment definition information*.

Example 1: Setting the environment in sh (Bourne shell)

- HiRDB is being installed in the `/prdb` directory.

```
$ PDDIR="/prdb"$ export PDDIR
```

Example 2: Setting the environment in csh (C shell)

- HiRDB is being installed in the `/prdb` directory

```
% setenv PDDIR "/prdb"
```

(b) SQL preprocessor activation

To activate the SQL preprocessor, use the `pdcpp` command (for C) or the `pdocc` command (for C++).

Following is the input format for starting the SQL preprocessor:

```
pdcpp input-file-name [options
[output-file-name|authorization-identifier]]
```

Note

In the case of C++, the underlined section must be changed to `pdocc`.

input-file-name

Specifies the name of the C source file. `.ec` must be used as the file identifier.

output-file-name

Specifies the name of the C source file. If the output filename is omitted, `.c` is used as the file identifier.

authorization-identifier

Specifies the default authorization identifier to be used when an authorization identifier is omitted in the SQL.

If the authorization identifier is omitted, the user identifier used during `CONNECT` is assumed.

options

Specifies, as necessary, the options shown in the table below. Upper-case and lower-case characters are not discriminated in the options.

Table 8-2: Preprocessing options (for C in the UNIX environment)

Preprocessing option	Description
`-s`	Specifies that only syntax is checked and that no post source is to be output; when this option is omitted, the post source is output.
`-o` *file-name*	Specifies a filename for the post source that is to be output; when this option is omitted, the input filename with its file identifier changed to `.c` is used as the output filename.
`-A` *authorization-identifier*	Specifies that the default authorization identifier, which is used when no authorization identifier is specified in a static SQL statement, is to be changed. A static SQL statement refers to the `INSERT`, `UPDATE`, `DELETE`, single-row `SELECT`, `OPEN` (format 1), `CALL`, `LOCK`, or `PURGE TABLE` statement.
`-h64`	Specifies that a post source for 64-bit mode is to be created. Note that this option cannot be specified if you have used a 32-bit edition preprocessor. If an embedded variable that uses the `long` type is declared, an error occurs.
`-P`	Specifies that no syntax check is to be executed on the SQL. This option can be specified when one of the following UAPs is preprocessed: • UAP for XDM/RD E2 connection • UAP that uses the SQL reserved word deletion facility If this option is not specified, the reserved words to be deleted by the SQL reserved word deletion facility and the SQL statements that can be used by XDM/RD E2 may cause syntax errors.

Preprocessing option	Description
`-Xo`	Specifies that the SQL statements extracted from the UAP are to be output to standard output. The output method for outputting the SQL statements is described below. • Embedded variables in SQL statements are replaced with the `?` parameter. • `INTO` clauses in single-row `SELECT` statements are deleted. • Multiple space characters between word clauses in SQL statements are replaced with one space character. • Any SQL statement that is split across several lines is consolidated into a single line. • Only SQL statements that are sent to the server during execution are output. SQL statements that are not executed (such as `WHENEVER` statements and `BEGIN DECLARE SECTION`) are not output. • A semicolon (`;`) is added to the end of an SQL statement. • Declarations of embedded variables are not output. • A dynamic SQL statement is output only if the SQL is specified with a literal. In all other cases, dynamic SQL statements are not output. • An `OPEN` statement outputs a query expression only if a format 1 cursor is used. • A post source is not generated.
`-Xe{y|n}`	Specifies whether the cursor for `PREPARE` statement execution is to be closed automatically. `y`: Creates a post source that closes the cursor automatically. `n`: Creates a post source that does not close the cursor automatically. If this option is omitted, the preprocessor creates a post source according to the specification value in the `PDPRPCRCLS` client environment definition.
`-Xv`	Specifies that `VARCHAR`- and `BINARY`-type structures are to be analyzed as normal structures when the `-E2` option is specified. To declare `VARCHAR`- and `BINARY`-type embedded variables, use the `SQL TYPE IS`-type variable declaration. This option must be specified together with the `-E2` option. Do not specify this option if the UAP uses macros for repetition columns.

Preprocessing option	Description
`-E{1|2|3} ["`*option-character-string*`"]`	Specifies that preprocessor declaration statements used in the UAP are to be validated or that embedded variables are to be used without being declared in the embedded SQL declare section, or both. Note that this option is ignored in `pdocc`. `-E1`: Specifies that preprocessor declaration statements are to be validated. `-E2`: Specifies that embedded variables are to be used without being declared in the embedded SQL declare section. This value can also specify that pointers or structure references are to be used as embedded variables. `-E3`: Specifies that both `-E1` and `-E2` apply. `"`*option-character-string*`"`: Specifies the path name of the directory from which the file to be included is to be retrieved. Specify the path name in the format of the `-I` option specified in the C compiler. When specifying multiple options in *option-character-string*, use semicolons to separate the options. You can also specify any C compiler. When the `-E2` option is specified, this value ignored. When the `-E1` option is specified, the path name to the compiler must be specified in the `PATH` environment variable because the preprocessor calls the C compiler internally.

Preprocessing option	Description

`-XU16[L|B][T"`*type-specifier*`"]`	• Specifies the UTF-16 byte order.
	If a UTF-16 character set name is specified in the declaration of an embedded variable, you must specify the UTF-16 byte order to be stored in embedded variables.
	`-XU16L[T"`*type-specifier*`"]`
	Use little endian for the UTF-16 byte order.
	`-XU16B[T"`*type-specifier*`"]`
	Use big endian for the UTF-16 byte order.
	If neither `L` nor `B` is specified, or this option is omitted, the UTF-16 byte order of the OS used for preprocessing is assumed as the byte order.
	• Specifies the type specifier used for expanding UTF-16 embedded variables.
	If you have declared a variable in a data description beginning with `SQL TYPE IS CHAR` or `SQL TYPE IS VARCHAR`, this option specifies the type specifier to be added to the variable or structure member that stores UTF-16 character string data in a C declaration that is expanded into the post source.
	You can only specify `sizeof(`*type-specifier)* == 2 for the type specifier, because one 2-byte UTF-16 character data item is stored in one array element of the type specified by the type specifier.
	Example of type specifier:
	This example can be used only by a compiler that is `sizeof(wchar_t)==2`:
	`-XU16T"wchar_t"`
	
	The following example can be used by any compiler because it is `sizeof(unsigned short)==2`:
	`-XU16T"unsigned short"`
	
	If you omit `T"`*type-specifier*`"` or this option, `char` is used as the type specifier. In such a case, one 2-byte UTF-16 character data is stored in two `char`-type array elements.

`-g{c89|c99}`	When the `-E2` or `-E3` option is specified, specifies the C standard that is to be complied with when the SQL preprocessor analyzes a UAP source. -gc89 Comply with C89 (ISO/IEC 9899:1990, Programming languages - C). -gc99 Comply with C99 (ISO/IEC 9899:1999, Programming languages - C). If this option is omitted, `-gc99` is assumed. If the `-E2` and `-E3` options are both omitted, this option is ignored, if specified.

Note 1

The following table shows the functions that can be used when the `-E` option is specified.

Function	Omitted	-E1	-E2	-E3
Validate the macro defined with `#define`.	N	Y	N	Y
Validate the header file that was included with `#include`.	N	Y	N	Y
Enable conditional compilation with `#if`, `#ifdef`, and other statements.	N	Y	N	Y
Use variables declared anywhere in the UAP as embedded variables.	N	N	Y	Y
Use structures as embedded variables.	N	N	Y	Y
Use pointers as embedded variables.	N	N	Y	Y

Legend:

Y: The function can be used.

N: The function cannot be used.

Note 2

When the `-E` option is specified, the preprocessor calls the C compiler internally. The following table shows the C compiler for each platform.

Platform	Compiler type	Load name for calling
HP-UX	HP-C compiler	`cc`
Solaris	SUN Workshop compiler	`cc`
AIX	C for AIX compiler	`xlc`
Linux	gcc compiler	`gcc`
Windows	Microsoft Visual C++ compiler	`CL.EXE`

If you wish to use any other C compiler, specify the absolute path name of the compiler, including the load directory, at the beginning of the *option-character-string* value. The directory name and the load name cannot include spaces or semicolons. If a path name has been added to the `PATH` environment variable, the path name does not have to be the absolute path name.

When specifying a load name, separate the load name and the options with a semicolon.

The compiler to be used must support the `-C` and `-E` options. This is because to process pseudo-instructions such as `#define` and `#include`, the preprocessor internally specifies the `-C` and `-E` options to the C compiler and creates temporary work files. In Linux, the preprocessor uses the `-xc` option in addition to the `-C` and `-E` options. In Solaris, the preprocessor also uses the `-Xs` option.

The other options that can be specified in *option-character-string* depend on the specifications of the compiler to be used. However, if an option that is incompatible with the `-C` or `-E` option is specified, the preprocessor produces an error. If an option that displays help information is used, the operation is not guaranteed.

Examples are shown as follows.

Example 1: The default C compiler is to be used

```
pdcpp connect.ec -E1"-I$PDDIR/include;-DDEBUG"
```

Example 2: If a user-specified C compiler is to be used

```
pdcpp connect.ec -E1"/usr/bin/gcc;-I$PDDIR/
include;-DDEBUG"
```

If the `-E2` and `-E3` options are both specified, the preprocessor analyzes syntax based on the C standard (C89 or C99) that is specified in, or assumed for, the `-g` option in order to recognize embedded variables declared at any location in the UAP. Therefore, a syntax error might result in the following cases:

- The `-E2` or `-E3` option is specified, but a syntax that does not comply with the selected standard is used in the UAP source file.
- The `-E3` option is specified, but a syntax that does not comply with the selected standard is used in the header file included by applying the `#include` statement.

To avoid syntax errors, use the syntax that complies with the selected standard in the UAP source files and header files. If a syntax error occurs because a syntax that does not comply with the selected standard is used in header files provided by the compiler product, you might be able to avoid the errors by specifying compiler options that comply with the selected standard in `-E3` *option-character-string*.

Example 3: IBM XL C V9.0 is used and the C89-compliant option (`-qlanglvl=extc89`) is specified

```
pdcpp connect.ec -E3"-qlanglvl=extc89;-I$PDDIR/include"
```

```
-gc89
```

Note 3

SQL statements and `SQL TYPE IS`-type variable declarations cannot be specified in the included header file. If the preprocessor finds an SQL statement or an `SQL TYPE IS`-type variable declaration in the header file, it displays an error message and continues processing but does not generate a post source. If you specify the `-E1` option and also specify an embedded SQL declare section in the header file, that section becomes invalid. To use variables defined in the header file as embedded variables, specify the `-E3` option. However, in this case as well, `SQL TYPE IS`-type variable declarations cannot be specified in the include file.

Note 4

If a `UTF16` character set name is specified in the declaration of an embedded variable, the SQL preprocessor expands the source code for specifying the character set name in the character set descriptor area in the post source. The source code depends on the specified `-XU16` option: `UTF-16LE` is set for little endian, and `UTF-16BE` is set for big endian.

If you determine the character set name for I/O variables during UAP execution by using an SQL descriptor area and character set descriptor area without using an embedded variable, the byte order specified in the `-XU16` option is ignored. In such a case, if you specify `UTF16` for the character set name, the byte order is set to big endian.

Note 5

The following data descriptions are applicable to a function for which a type specifier is specified in the `-XU16` option:

- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]`UTF16` *variable-name*`;`
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]`UTF16` *variable-name*`[m];`
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]`UTF16` `*`*variable-name*`;`
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]`UTF16` *variable-name*`;`
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]`UTF16` *variable-name*`[m];`
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS]

[MASTER.]`UTF16` `*`*variable-name*`;`

Note 6

The handling of the reserved words added in C99 depends on the specified `-g` option. The following reserved words have been added in C99:

`restrict`, `inline`, `_Bool`, `_Complex`, `_Imaginary`, `_Pragma`

In C89, `restrict` and `inline` can be used as identifiers such as variable names, but they can no longer be used as identifiers in C99. If `restrict` and `inline` are used as identifiers in an existing UAP that is compliant with C89 and that UAP is analyzed as being compliant with C99, an error will result.

The following table describes the handling of the reserved words added in C99:

-g option	Handling
Omitted `-gc99` specified	Handled as reserved words as in C99.
`-gc89` specified	Handled as identifiers.#

#

Names that begin with one underscore (_) followed immediately by an upper-case letter (`A` to `Z`) or by a second underscore are also reserved for libraries in C89. Therefore, if `_Bool`, `_Complex`, `_Imaginary`, or `_Pragma` is used as an identifier in a UAP, the preprocessor operation is not guaranteed.

The table below explains the `-g` option specifications. The `-g` option has no effect other than on the handling of reserved words added in C99.

Usage of reserved words added in C99	-g option specification
Used as reserved words	Omit the `-g` option or specify `-gc99`.
Used as identifiers	Specify `-gc89`.
Not used as reserved words or identifiers	There is no need to specify the `-g` option. The words can be specified as comments.

You should note the following about specifying the `-g` option:

- If a C99-compliant compiler product is used, reserved words added in C99 might be used in a header file provided by the compiler. If you specify the `-E3` option, such a header file might be included by the `#include` statement.

- Some compiler products enable you to specify the handling of reserved words individually (for example, specifying `restrict` as a reserved word and `inline` as a non-reserved word). However, with the SQL preprocessor, the handling of reserved words cannot be specified individually.

In a UNIX environment, we recommend that you specify in the `-g` option the standard supported by the compiler in order to avoid problems. Do not specify a compiler option that supports specification of the handling of reserved words added in C99 individually.

Note 7

If you omit the `-E2` and `-E3` options, reserved words added in C99 are handled as identifiers in the embedded SQL declaration section. Only the embedded SQL declaration section and SQL statements (`EXEC SQL ...;`) are analyzed.

1. Examples of command specification in C

Example 1

The C source filename is `sample` and post source is not to be output.

```
pdcpp sample.ec -s
```

Example 2

The C source filename is `sample` and the filename of the post source to be output is `main.`

```
pdcpp sample.ec -o main.c
```

2. Examples of command specification in C++

Example 1

The C source filename is `sample` and post source is not to be output.

```
pdocc sample.EC -s
```

Example 2

The C source filename is `sample` and the filename of the post source to be output is `main.`

```
pdocc sample.EC -o main.C
```

(c) SQL preprocessor return codes

The SQL preprocessor returns a return code to the OS when the processing is complete. The return code can be referenced by displaying the contents of the `$?` shell variable (in the case of Bourne shell) or the `$status` shell variable (in the case of C shell).

The following table lists and describes the return codes.

Table 8-3: SQL preprocessor return codes (for C programs in a UNIX environment)

Return code	Explanation
`0`	Normal termination
`4, 8`	Error (preprocessing was completed)
`12, 16`	Error (preprocessing terminated abnormally)

(d) Error output

When a syntax error is detected in an SQL statement, the SQL preprocessor ignores the SQL statement and continues processing. If an error is detected in an option specification, however, processing is suspended. Processing terminates abnormally when a system error, such as a memory shortage or a file I/O error, occurs and processing cannot be continued.

For a syntax error in an SQL statement, the SQL preprocessor outputs an error message to the standard error output. By redirecting the standard error output, the error message can be stored in a file. This file can be referenced for the error content, the UAP source filename, and the error location (line number in the SQL statement).

The following table shows the standard input and output for the SQL preprocessor.

Table 8-4: SQL preprocessor standard input and output (for C programs in a UNIX environment)

File	Application
Standard input	File input (cannot be used by the user)
Standard output	File output (cannot be used by the user)
Standard error output	Output of error messages

(2) COBOL

(a) Environment variable setting

The following environment variables can be set before a UAP is preprocessed:

`PDDIR`

This environment variable specifies the absolute path name for the installation directory for HiRDB (server or client); the default directory is `/HiRDB`.

This variable need not be specified if the installation destination is `/HiRDB`.

PDCBLFIX

This environment variable specifies an optional file identifier other than the standard identifier of the COBOL source file.

The specification must be a character string of 1-4 alphabetic characters beginning with a period. The file identifier specified in this environment variable can be used only for the input file.

PDCBLLIB

This environment variable specifies directories from which library texts to be included in the source file are to be retrieved by the SQL COPY statement. Multiple directories must be separated by a colon. When this environment variable is omitted, only the current directory is retrieved.

LANG

This environment variable specifies the character code classification of the HiRDB client environment.

For details about the character code classifications that can be specified in LANG, see *Table 8-1*.

If you do not specify this environment variable or if you specify a different value from the value that is specified in the environment variable, ja_JP.SJIS is assumed.

PDCLTLANG

This environment variable specifies the character code classification to be used in preprocessing instead of the character code classification specified by the LANG environment variable. For details about the PDCLTLANG operand, see *6.6.4 Environment definition information*.

Example 1: Setting the environment in sh (Bourne shell)

```
$ PDDIR="/prdb"                                          1
$ PDCBLFIX=".Cob"                                        2
$ PDCBLLIB=$HOME/cobol/include:$HOME/cobol/source        3
$ export PDDIR PDCBLFIX PDCBLLIB                         4
```

1. Specifies the installation directory (/prdb in this example).
2. Specifies .Cob as the COBOL language source file identifier.
3. Specifies the two directories ($HOME/cobol/include and $HOME/cobol/source) from which library text is to be retrieved.
4. Enables referencing by the SQL preprocessor.

Example 2: Setting the environment in csh (C shell)

```
% setenv PDDIR "/prdb"                                         1
% setenv PDCBLFIX ".Cob"                                   2
% setenv PDCBLLIB $HOME/cobol/include:$HOME/cobol/source   3
```

1. Specifies the installation directory (`/prdb` in this example).
2. Specifies `.Cob` as the COBOL language source file identifier.
3. Specifies two directories (`$HOME/cobol/include` and `$HOME/cobol/source`) from which library text is to be retrieved.

(b) SQL preprocessor activation

To activate the SQL preprocessor, use the `pdcbl` command (for COBOL) or the `pdocb` command (for OOCOBOL).

Following is the input format for starting the SQL preprocessor:

```
pdcbl input-file-name [options
[output-file-name|authorization-identifier]]
```

Note

In the case of OOCOBOL, the underlined section must be changed to `pdocb`.

input-file-name

Specifies the name of the COBOL source file. `.ecb`, `.cob`, or `.cbl` must be used as the file identifier. If any other file identifier was registered during environment setting, that identifier can also be used.

output-file-name

Specifies the name of the COBOL source file. If the output filename is *omitted*, `.cbl` is used as the file identifier.

authorization-identifier

Specifies the default authorization identifier to be used when an authorization identifier is omitted in the SQL.

If the authorization identifier is omitted, the user identifier used during `CONNECT` is assumed.

options

Specifies, as necessary, the options shown in the table below. Upper-case and lower-case characters are not discriminated in the options.

Table 8-5: Preprocessing options (for COBOL in the UNIX environment)

Preprocessing option	Description
`-s`	Specifies that only syntax is checked and that no post source is output; when this option is omitted, the post source is output.

Preprocessing option	Description
`-o` *file-name*	Specifies a filename for the post source; when this option is omitted, the input filename with its file identifier changed to `.cbl` is used as the output filename. If the input file identifier is `.cbl`, this option must be specified to change the post source filename to an identifier other than `.cbl`.
`-Xc`	Specifies that the double quotation mark is used as the quotation mark in the character string created by the SQL preprocessor; the default quotation mark is the apostrophe.
`-A` *authorization-identifier*	Specifies that the default authorization identifier, which is used when no authorization identifier is specified in a static SQL statement, is to be changed. A static SQL statement refers to the `INSERT`, `UPDATE`, `DELETE`, single-row `SELECT`, `OPEN` (format 1), `CALL`, `LOCK`, or `PURGE TABLE` statement.
`-h64`	Specifies that a post source for 64-bit mode is to be created. Note that this option cannot be specified if you have used a 32-bit edition preprocessor.
`-P`	Specifies that no syntax check is to be executed on the SQL. This option can be specified when one of the following UAPs is preprocessed: • UAP for XDM/RD E2 connection • UAP that uses the SQL reserved word deletion facility If this option is not specified, the reserved words to be deleted by the SQL reserved word deletion facility and the SQL statements that can be used by XDM/RD E2 may cause syntax errors.

Preprocessing option	Description
`-Xo`	Specifies that the SQL statements extracted from the UAP are to be output to standard output. The output method for outputting the SQL statements is described below. • Embedded variables in SQL statements are replaced with the `?` parameter. • `INTO` clauses in single-row `SELECT` statements are deleted. • Multiple space characters between word clauses in SQL statements are replaced with one space character. • Any SQL statement that is split across several lines is consolidated into a single line. • Only SQL statements that are sent to the server during execution are output. SQL statements that are not executed (such as `WHENEVER` statements and `BEGIN DECLARE SECTION`) are not output. • A semicolon (`;`) is added to the end of an SQL statement. • Declarations of embedded variables are not output. • A dynamic SQL statement is output only if the SQL is specified with a literal. In all other cases, dynamic SQL statements are not output. • An `OPEN` statement outputs a query expression only if a format 1 cursor is used. • A post source is not generated.
`-c {m|s}`	Specifies the COBOL compiler type. `m`: Micro Focus COBOL `s`: SUN Japanese COBOL
`-Xe{y|n}`	Specifies whether the cursor for `PREPARE` statement execution is to be closed automatically. `y`: Creates a post source that closes the cursor automatically. `n`: Creates a post source that does not close the cursor automatically. If this option is omitted, the preprocessor creates a post source according to the specification value in the `PDPRPCRCLS` client environment definition.
`-E2`	Specifies that embedded variables are to be used without being declared in the embedded SQL declare section.

Preprocessing option	Description

`-XU16[L|B]`	You specify this option when you use the Unicode function of COBOL2002. For details about the execution of UAPs that use the Unicode function of COBOL2002, see *8.4.3 UAP execution using the Unicode functionality of COBOL2002*.
	This option specifies the byte order of character codes (UTF-16) in Japanese data items.
	`-XU16L`
	Uses little endian for the UTF-16 byte order.
	`-XU16B`
	Uses big endian for the UTF-16 byte order.
	`-XU16`
	Uses the byte order of the OS used for preprocessing for the UTF-16 byte order:
	• In Windows and Linux, little endian is used.
	• In AIX, big endian is used.
	If this option is specified for any other OS, preprocessor operations are not guaranteed.
	Do not specify this option if you do not use the Unicode function of COBOL2002. If this option is specified in such a case, it will not be possible to use Japanese data items in the declaration of embedded variables.

1. Examples of command specification in COBOL

   Example 1

   The UAP source filename is `sample` and no post source will be output.
   ```
   pdcbl sample.ecb -s
   ```

   Example 2

   The UAP source filename is `sample` and the filename of the post source to be output is `main`.
   ```
   pdcbl sample.ecb -o main.cbl
   ```

2. Examples of command specification in OOCOBOL

   Example 1

   The UAP source filename is `sample` and no post source will be output.
   ```
   pdocb sample.eoc -s
   ```

   Example 2

   The UAP source filename is `sample` and the filename of the post source to be output is `main`.
   ```
   pdocb sample.eoc -o main.ocb
   ```

(c) SQL preprocessor return codes

The SQL preprocessor returns a return code to the OS when the processing is completed. The return code can be referenced by displaying the contents of the `$?` shell variable (in the case of Bourne shell) or the `$status` shell variable (in the case of C shell).

The following table lists and describes the return codes.

Table 8-6: SQL preprocessor return codes (for COBOL programs in a UNIX environment)

Return code	Explanation
`0`	Normal termination
`4, 8`	Error (preprocessing was completed)
`12, 16`	Error (preprocessing terminated abnormally)

(d) Error output

When a syntax error is detected in an SQL statement, the SQL preprocessor ignores that SQL statement and continues processing. If an error is detected in an option specification, however, processing is suspended. Processing terminates abnormally when a system error, such as a memory shortage or a file I/O error, occurs and processing cannot be continued.

For a syntax error in an SQL statement, the SQL preprocessor outputs an error message to the standard error output. By redirecting the standard error output, the error message can be stored in a file. This file can be referenced for the error content, the UAP source filename, and the error location (line number in the SQL statement).

The following table shows the standard input and output of the SQL preprocessor.

Table 8-7: SQL preprocessor standard input and output (for COBOL programs in a UNIX environment)

File	Application
Standard input	File input (cannot be used by the user)
Standard output	File output (cannot be used by the user)
Standard error output	Output of error messages

8.2.3 Preprocessing in Windows

(1) C

(a) Environment variable setting

Before a UAP is preprocessed, the environment variable described below can be specified in the `HiRDB.INI` file, if necessary.

The `HiRDB.INI` file is installed in the `%windir%` directory.

`PDCLTLANG`

Specifies the character code classification to be used for preprocessing. If this environment variable is omitted, `sjis` is assumed. For details about the `PDCLTLANG` operand, see *6.6.4 Environment definition information*.

(b) SQL preprocessor activation

Following are the three methods of activating the SQL preprocessor:

- Execution by means of overlaying icons
- Execution by means of filename specification
- Execution from the command prompt or MS-DOS prompt

■ **Execution by overlaying icons**

In Windows Explorer, drag the file to be preprocessed onto and overlay it on the preprocessor file (`PDCPP.EXE`); execution then occurs automatically.

When this method is used, no options can be specified to be set during execution.

■ **Execution by filename specification**

Click the preprocessor icon (`PDCPP.EXE`), and follow the procedures below:

1. Select **Run** from the **File** menu.
2. Specify a filename and options on the command line.

■ **Execution from the command prompt or MS-DOS prompt**

Activate the command prompt or MS-DOS prompt. Execute the program by entering either `PDCPP.EXE` (in C) or `PDOCC.EXE` (in C++).

A command is entered in the following format:
<u>`PDCPP.EXE`</u> *input-file-name* [*options*
[*output-file*-name|authorization-*identifier*]]

Note

In the case of C++, the underlined section must be changed to `PDOCC.EXE`.

input-file-name

Specifies the name of the C source file. .EC must be used as the file identifier.

output-file-name

Specifies the name of the C source file. If the output filename is omitted, .C is used as the file identifier.

authorization-identifier

Specifies the default authorization identifier to be used when an authorization identifier is omitted in the SQL. If the authorization identifier is omitted, the user identifier used during CONNECT is assumed.

options

Specifies, as necessary, the options shown in table below. Upper-case and lower-case characters are not discriminated in the options.

Table 8-8: Preprocessing options (for C in the Windows environment)

Preprocessing option	Description
/S	Specifies that only syntax is checked and that no post source will be output; when this option is omitted, the post source is output. Note that the SQL preprocessor may not be able to detect all syntax errors in the SQL statements because it does not perform a rigorous SQL syntax check unless /Xp is also specified.
/O *file-name*	Specifies a filename for the post source that is output; when this option is omitted, the input filename with its file identifier changed to .C is used as the output filename.
/A *authorization-identifier*	Specifies that the default authorization identifier, which is used when no authorization identifier is specified in a static SQL statement, is to be changed. A static SQL statement refers to the INSERT, UPDATE, DELETE, single-row SELECT, OPEN (format 1), CALL, LOCK, or PURGE TABLE statement.
/h64	Specifies that a post source for 64-bit mode is to be created. Note that this option cannot be specified if you have used a 32-bit edition preprocessor. If an embedded variable that uses the long type is declared, an error occurs.
/Xe{y|n}	Specifies whether the cursor for PREPARE statement execution is to be closed automatically. y: Creates a post source that closes the cursor automatically. n: Creates a post source that does not close the cursor automatically. If this option is omitted, the preprocessor creates a post source according to the specification value in the PDPRPCRCLS client environment definition.

Preprocessing option	Description
/Xv	Specifies that VARCHAR- and BINARY-type structures are to be analyzed as normal structures when the /E2 option is specified. To declare VARCHAR- and BINARY-type embedded variables, use the SQL TYPE IS-type variable declaration. This option must be specified together with the /E2 option. Do not specify this option if the UAP uses macros for repetition columns.
/XA	Specifies that an X/Open-compliant API is to be used to create the UAP.
/Xo	Specifies that the SQL statements extracted from the UAP are to be output to standard output. The output method for outputting the SQL statements is described below. • Embedded variables in SQL statements are replaced with the ? parameter. • INTO clauses in single-row SELECT statements are deleted. • Multiple space characters between word clauses in SQL statements are replaced with one space character. • Any SQL statement that is split across several lines is consolidated into a single line. • Only SQL statements that are sent to the server during execution are output. SQL statements that are not executed (such as WHENEVER statements and BEGIN DECLARE SECTION) are not output. • A semicolon (;) is added to the end of an SQL statement. • Declarations of embedded variables are not output. • A dynamic SQL statement is output only if the SQL is specified with a literal. In all other cases, dynamic SQL statements are not output. • An OPEN statement outputs a query expression only if a format 1 cursor is used. • A post source is not generated.

Preprocessing option	Description
`/E{1|2|3} ["`*option-character-string*`"]`	Specifies that preprocessor declaration statements used in the UAP are to be validated or that embedded variables are to be used without being declared in the embedded SQL declare section, or both. Note that this option is ignored in `PDOCC.EXE`. `/E1`: Specifies that preprocessor declaration statements are to be validated. `/E2`: Specifies that embedded variables are to be used without being declared in the embedded SQL declare section. This value can also specify that pointers or structure references are to be used as embedded variables. `/E3`: Specifies that both `/E1` and `/E2` apply. `"`*option-character-string*`"`: Specifies the path name of the directory from which the file to be included is to be retrieved. Specify the path name in the format of the `/I` option specified in the C compiler. When specifying multiple options in *option-character-string*, use semicolons to separate the options. You can also specify any C compiler. When the `/E2` option is specified, this value ignored. When the `/E1` option is specified, the path name to the compiler must be specified in the `PATH` environment variable because the preprocessor calls the C compiler internally.
`/Xp`	Specifies that a rigorous SQL syntax check is to be executed. However, do not specify this option when the SQL reserved word deletion facility is used.

Preprocessing option	Description

`/XU16[L|B][T"`*type-specifier*`"]`	• Specifies the UTF-16 byte order.
	If a UTF16 character set name is specified in the declaration of an embedded variable, you must specify the UTF-16 byte order to be stored in embedded variables.
	`/XU16L[T"`*type-specifier*`"]`
	Use little endian for the UTF-16 byte order.
	`/XU16B[T"`*type-specifier*`"]`
	Use big endian for the UTF-16 byte order.
	If neither `L` nor `B` is specified, or this option is omitted, the UTF-16 byte order of the OS used for preprocessing is assumed as the byte order.
	• Specifies the type specifier used for expanding UTF-16 embedded variables.
	If you have declared a variable in a data description beginning with `SQL TYPE IS CHAR` or `SQL TYPE IS VARCHAR`, this option specifies the type specifier to be added to the variable or structure member that stores UTF-16 character string data in a C declaration that is expanded into the post source.
	You can only specify `sizeof`(*type-specifier)* == 2 for the type specifier, because one 2-byte UTF-16 character data item is stored in one array element of the type specified by the type specifier.
	Example of type specifier:
	This example can be used only by a compiler that is `sizeof(wchar_t)==2`:
	`/XU16T"wchar_t"`
	The following example can be used by any compiler because it is `sizeof(unsigned short)==2`:
	`/XU16T"unsigned short"`
	If you omit `T"`*type-specifier*`"` or this option, `char` is used as the type specifier. In such a case, one 2-byte UTF-16 character data is stored in two `char`-type array elements.

`/g{c89|c99}`	When the `/E2` or `/E3` option is specified, specifies the C standard that is to be complied with when the SQL preprocessor analyzes a UAP source.
	`/gc89`
	Comply with C89 (ISO/IEC 9899:1990, Programming languages - C).
	`/gc99`
	Comply with C99 (ISO/IEC 9899:1999, Programming languages - C).
	If this option is omitted, `/gc89` is assumed.
	If the `/E2` and `/E3` options are both omitted, this option is ignored, if specified.

Note 1

The following table shows the functions that can be used when the /E option is specified.

Function	Omitted	/E1	/E2	/E3
Validate the macro defined with #define.	N	Y	N	Y
Validate the header file that was included with #include.	N	Y	N	Y
Enable conditional compilation with #if, #ifdef, and other statements.	N	Y	N	Y
Use variables declared anywhere in the UAP as embedded variables.	N	N	Y	Y
Use structures as embedded variables.	N	N	Y	Y
Use pointers as embedded variables.	N	N	Y	Y

Legend:

Y: The function can be used.

N: The function cannot be used.

Note 2

When the /E option is specified, the preprocessor calls the Microsoft Visual C++ compiler (load name during calling: CL.EXE) internally.

If you wish to use any other C compiler, specify the absolute path name of the compiler, including the load directory, at the beginning of the *option-character-string* value. The directory name and the load name cannot include spaces or semicolons. If a path name has been added to the PATH environment variable, the path name does not have to be the absolute path name.

When specifying a load name, separate the load name and the options with a semicolon.

The compiler to be used must support the /C and /E options. This is because to process pseudo-instructions such as #define and #include, the preprocessor internally specifies the /C and /E options to the C compiler and creates temporary work files. The other options that can be specified in *option-character-string* depend on the specifications of the compiler to be used. However, if an option that is incompatible with the /C or /E option is specified, the preprocessor produces an error. If an option that displays help information is used, the operation is not guaranteed.

If the /E2 and /E3 options are both specified, the preprocessor analyzes syntax based on the C standard (C89 or C99) that is specified in or assumed for the /g option in order to recognize embedded variables declared at any location in the UAP. Therefore, a syntax error might result in the following cases:

- The /E2 or /E3 option is specified, but a syntax that does not comply with the selected standard is used in the UAP source file.
- The /E3 option is specified, but a syntax that does not comply with the selected standard is used in the header file included by applying the #include statement.

To avoid syntax errors, use the syntax that complies with the selected standard in the UAP source files and header files. If a syntax error occurs because a syntax that does not comply with the selected standard is used in header files provided by the compiler product, you might be able to avoid the errors by specifying compiler options that comply with the selected standard in /E3 *option-character-string*.

Note 3

SQL statements and SQL TYPE IS-type variable declarations cannot be specified in the included header file. If the preprocessor finds an SQL statement or an SQL TYPE IS-type variable declaration in the header file, it displays an error message and continues processing but does not generate a post source. If you specify the /E1 option and also specify an embedded variable declare section in the header file, that section becomes invalid. To use variables defined in the header file as embedded variables, specify the /E3 option. However, in this case as well, SQL TYPE IS-type variable declarations cannot be specified in the include file.

Note 4

If a UTF16 character set name is specified in the declaration of an embedded variable, the SQL preprocessor expands the source code for specifying the character set name in the character set descriptor area in the post source. The source code depends on the specified /XU16 option: UTF-16LE is set for little endian, and UTF-16BE is set for big endian.

If you determine the character set name for I/O variables during UAP execution by using an SQL descriptor area and character set descriptor area without using an embedded variable, the byte order specified in the /XU16 option is ignored. In such a case, if you specify UTF16 for the character set name, the byte order is set to big endian.

Note 5

The following data descriptions are applicable to a function for which a type specifier is specified in the /XU16 option:

- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *variable-name*;

- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *variable-name*[m];
- SQL TYPE IS CHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 **variable-name*;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *variable-name*;
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 *variable-name*[m];
- SQL TYPE IS VARCHAR(2n) CHARACTER SET [IS] [MASTER.]UTF16 **variable-name*;

Note 6

The handling of the reserved words added in C99 depends on the specified /g option. The following reserved words have been added in C99:

`restrict`, `inline`, `_Bool`, `_Complex`, `_Imaginary`, `_Pragma`

In C89, `restrict` and `inline` can be used as identifiers such as variable names, but they can no longer be used as identifiers in C99. If `restrict` and `inline` are used as identifiers in an existing UAP that is compliant with C89 and that UAP is analyzed as being compliant with C99, an error will result.

The following table describes the handling of the reserved words added in C99:

/g option	Handling
Omitted	Handled as identifiers.#
/gc99 specified	Handled as reserved words as in C99.
/gc89 specified	Handled as identifiers.#

#

Names that begin with one underscore (_) followed immediately by an upper-case letter (A to Z) or by a second underscore are also reserved for libraries in C89. Therefore, if `_Bool`, `_Complex`, `_Imaginary`, or `_Pragma` is used as an identifier in a UAP, the preprocessor operation is not guaranteed.

The table below explains the /g option specifications. The /g option has no effect other than on the handling of reserved words added in C99.

Usage of reserved words added in C99	/g option specification
Used as reserved words	If you specify /gc99, you can execute the SQL preprocessor. A post source cannot be compiled because Visual C++ (version up to 2008) does not support C99. If the compiler used is not Visual C++, the compilation operation is not guaranteed.
Used as identifiers	Omit the /g option or specify /gc89.
Not used as reserved words or identifiers	There is no need to specify the /g option. The words can be specified as comments.

You should note the following about specifying the /g option:

- If a C99-compliant compiler product is used, reserved words added in C99 might be used in a header file provided by the compiler. If you specify the /E3 option, such a header file might be included by the #include statement.
- Some compiler products enable you to specify the handling of reserved words individually (for example, specifying restrict as a reserved word and inline as a non-reserved word). However, with the SQL preprocessor, the handling of reserved words cannot be specified individually.

Note 7

If you omit the /E2 and /E3 options, reserved words added in C99 are handled as identifiers in the embedded SQL declaration section. Only the embedded SQL declaration section and SQL statements (EXEC SQL ...;) are analyzed.

1. Examples of command specification in C

Example 1

The UAP source filename is SAMPLE and no post source will be output.

```
PDCPP SAMPLE.EC /S
```

Example 2

The UAP source filename is SAMPLE and the filename of the post source to be output is MAIN.

```
PDCPP SAMPLE.EC /O MAIN.C
```

2. Examples of command specification in C++

Example 1

The UAP source filename is SAMPLE and no post source will be output.

```
PDOCC.EXE SAMPLE.ECP /S
```

Example 2

The UAP source filename is SAMPLE and the filename of the post source to be output is MAIN.

```
PDOCC.EXE SAMPLE.ECP /O MAIN.CPP
```

(c) SQL preprocessor return codes

The SQL preprocessor returns a return code to the OS when the processing is completed. The return code can be referenced by the OS batch command ERRORLEVEL.

The following table lists and describes the return codes.

Table 8-9: SQL preprocessor return codes (for C programs in a Windows environment)

Return code	Explanation
0	Normal termination
4, 8	Error (preprocessing was completed)
12, 16	Error (preprocessing terminated abnormally)

(d) Error output

When a syntax error is detected in an SQL statement, the SQL preprocessor ignores that SQL statement and continues processing. If an error is detected in an option specification, however, processing is suspended. Processing terminates abnormally when a system error, such as a memory shortage or a file I/O error, occurs and processing cannot be continued.

For a syntax error in an SQL statement, the SQL preprocessor outputs an error message to the standard error output. By redirecting the standard error output, the error message can be stored in a file. This file can be referenced for the error content, the UAP source filename, and the error location (line number in the SQL statement).

The following table shows the standard input and output for the SQL preprocessor.

Table 8-10: SQL preprocessor standard input and output (for C programs in a Windows environment)

File	Application
Standard input	File input (cannot be used by the user)
Standard output	File output (cannot be used by the user)

File	Application
Standard error output	Output of error messages

(2) COBOL

(a) Environment variable setting

The following environment variables can be set in the `HIRDB.INI` file before a UAP is preprocessed (the `HIRDB.INI` file is installed in the `%windir%` directory):

`PDCBLFIX`

This environment variable specifies an optional file identifier other than the standard identifier of the COBOL source file.

The specification must be a character string of 1-4 alphabetic characters beginning with a period. The file identifier specified in this environment variable can be used only for the input file.

`PDCBLLIB`

This environment variable specifies directories from which library texts to be included in the source file are to be retrieved by the `COPY` statement. When specifying multiple directories, separate the directories with a colon. When this environment variable is omitted, only the current directory is retrieved.

`PDCLTLANG`

This environment variable should be specified if a specific type of character codes is to be used for preprocessing. The default is `sjis`. For details about the `PDCLTLANG` operand, see *6.6.4 Environment definition information*.

Example

```
[HiRDB]                 1
PDCBLFIX=.AAA           2
PDCBLLIB=E:\USER\COPY   3
```

1. Specifies `[HiRDB]`.
2. Specifies `.AAA` as a COBOL language source file identifier.
3. Specifies a directory (`E:\USER\COPY` in this example) from which library text to be included by the `COPY` statement is to be retrieved.

(b) SQL preprocessor activation

Following are the three methods of activating the SQL preprocessor:

- Execution by means of overlaying icons
- Execution by means of filename specification

- Execution from the command prompt or MS-DOS prompt

■ **Execution by overlaying icons**

In Windows Explorer, drag the file to be preprocessed onto and overlay it on the preprocessor file (`PDCBL.EXE`). Execution then occurs automatically.

When this method is used, no options can be specified to be set during execution.

■ **Execution by filename specification**

Click the preprocessor icon (`PDCBL.EXE`) and use the following procedure:

1. Select **Run** from the **File** menu.
2. Specify a filename and options on the command line.

■ **Execution from the command prompt or MS-DOS prompt**

Activate the command prompt or MS-DOS prompt. Execute the program by entering either `PDCBL.EXE` (in COBOL) or `PDOCB.EXE` (in OOCOBOL).

A command is entered in the following format:
`PDCBL.EXE` *input-file-name* [*options* [*output-file-name*|*authorization-identifier*]]

Note

In the case of OOCOBOL, the underlined section must be changed to `PDOCB.EXE`.

input-file-name

Specifies the name of the COBOL source file. `.ECB`, `.COB`, or `.CBL` must be used as the file identifier. If any other file identifier was registered during environment setting, that identifier can also be used.

output-file-name

Specifies the name of the COBOL source file. If the output filename is omitted, `.CBL` is used as the file identifier.

authorization-identifier

Specifies the default authorization identifier to be used when an authorization identifier is omitted in the SQL. If the authorization identifier is omitted, the user identifier used during `CONNECT` is assumed.

options

Specifies, as necessary, the options shown in the table below. Upper-case and lower-case characters are not discriminated in the options.

Table 8-11: Preprocessing options (for COBOL in the Windows environment)

Preprocessing option	Description
`/S`	Specifies that only syntax is checked and that no post source will be output; when this option is omitted, the post source is output. Note that the SQL preprocessor may not be able to detect all syntax errors in the SQL statements because it does not perform a rigorous SQL syntax check unless `/Xp` is also specified.
`/O` *file-name*	Specifies that the filename for the output post source is to be changed. When this option is omitted, the input filename with its file identifier changed to `.CBL` (for COBOL language) or `.OCB` (for OOCOBOL language) and is used as the output filename. If the input file identifier is `.CBL` (for COBOL language) or `.OCB` (for OOCOBOL language), this option must be specified to change the post source filename to an identifier other than `.CBL` (for COBOL language) or `.OCB` (for OOCOBOL language).
`/XC`	Specifies that the double quotation mark is used as the quotation mark in the character string to be created by the SQL preprocessor; the default quotation mark is the apostrophe.
`/A` *authorization-identifier*	Specifies that the default authorization identifier, which is used when no authorization identifier is specified in a static SQL statement, is to be changed. A static SQL statement refers to the `INSERT`, `UPDATE`, `DELETE`, single-row `SELECT`, `OPEN` (format 1), `CALL`, `LOCK`, or `PURGE TABLE` statement.
`/h64`	Specifies that a post source for 64-bit mode is to be created. Note that this option cannot be specified if you have used a 32-bit edition preprocessor.
`/XD`	Specifies that a DLL is to be created. The prerequisite compiler for creating a DLL is COBOL85 Version 4.0 04-02 or a later version. Do not create an application that contains both UAPs that were preprocessed by specifying the `/XD` option and UAPs that were preprocessed without specifying the `/XD` option. Otherwise, an error (KCCBO204R-S) occurs in the COBOL runtime library during application execution.
`/Xe{y|n}`	Specifies whether the cursor for `PREPARE` statement execution is to be closed automatically. `y`: Creates a post source that closes the cursor automatically. `n`: Creates a post source that does not close the cursor automatically. If this option is omitted, the preprocessor creates a post source according to the specification value in the `PDPRPCRCLS` client environment definition.
`/XAD`	Specifies that a UAP that used an X/Open-compliant API is to be created as a DLL.

Preprocessing option	Description
/XA	Specifies that the UAP is to be created by using an X/Open-compliant API.
/Xo	Specifies that the SQL statements extracted from the UAP are to be output to standard output. The output method for outputting the SQL statements is described below. • Embedded variables in SQL statements are replaced with the ? parameter. • INTO clauses in single-row SELECT statements are deleted. • Multiple space characters between word clauses in SQL statements are replaced with one space character. • Any SQL statement that is split across several lines is consolidated into a single line. • Only SQL statements that are sent to the server during execution are output. SQL statements that are not executed (such as WHENEVER statements and BEGIN DECLARE SECTION) are not output. • A semicolon (;) is added to the end of an SQL statement. • Declarations of embedded variables are not output. • A dynamic SQL statement is output only if the SQL is specified with a literal. In all other cases, dynamic SQL statements are not output. • An OPEN statement outputs a query expression only if a format 1 cursor is used. • A post source is not generated.
/E2	Specifies that embedded variables are to be used without being declared in the embedded SQL declare section.
/Xp	Specifies that a rigorous SQL syntax check is to be executed. However, do not specify this option when the SQL reserved word deletion facility is used.

Preprocessing option	Description
`/XU16[L|B]`	You specify this option when you use the Unicode function of COBOL2002. For details about the execution of UAPs that use the Unicode function of COBOL2002, see *8.4.3 UAP execution using the Unicode functionality of COBOL2002*. This option specifies the byte order of character codes (UTF-16) in Japanese data items. `/XU16L` Uses little endian for the UTF-16 byte order. /XU16B Uses big endian for the UTF-16 byte order. /XU16 Uses the byte order of the OS used for preprocessing for the UTF-16 byte order: • In Windows and Linux, little endian is used. • In AIX, big endian is used. If this option is specified for any other OS, the preprocessor operations are not guaranteed. Do not specify this option if you do not use the Unicode function of COBOL2002. If this option is specified in such a case, it will not be possible to use Japanese data items in the declaration of embedded variables.

1. Examples of command specification in COBOL

   Example 1

   The UAP source filename is `SAMPLE` and no post source will be output.

   ```
   PDCBL SAMPLE.ECB /S
   ```

   Example 2

   The UAP source filename is `SAMPLE` and the filename of the post source to be output is `MAIN`.

   ```
   PDCBL SAMPLE.ECB /O MAIN.CBL
   ```

2. Examples of command specification in OOCOBOL

   Example 1

   The UAP source filename is `SAMPLE` and no post source will be output.

   ```
   PDOCB.EXE SAMPLE.EOC /S
   ```

   Example 2

   The UAP source filename is `SAMPLE` and the filename of the post source to be output is `MAIN`.

   ```
   PDOCB.EXE SAMPLE.EOC /O MAIN.OCB
   ```

(c) SQL preprocessor return codes

The SQL preprocessor returns a return code to the OS when the processing is completed. The return code can be referenced by the OS batch command `ERRORLEVEL`.

The following table lists and describes the return codes.

Table 8-12: SQL preprocessor return codes (for COBOL programs in a Windows environment)

Return code	Explanation
`0`	Normal termination
`4, 8`	Error (preprocessing was completed)
`12, 16`	Error (preprocessing terminated abnormally)

(d) Error output

When a syntax error is detected in an SQL statement, the SQL preprocessor ignores that SQL statement and continues processing. If an error is detected in an option specification, however, processing is suspended. Processing terminates abnormally when a system error, such as a memory shortage or a file I/O error, occurs and processing cannot be continued.

For a syntax error in an SQL statement, the SQL preprocessor outputs an error message to the standard error output. By redirecting the standard error output, the error message can be stored in a file. This file can be referenced for the error content, the UAP source filename, and the error location (line number in the SQL statement).

The following table shows the standard input and output of the SQL preprocessor.

Table 8-13: SQL preprocessor standard input and output (for COBOL programs in a Windows environment)

File	Application
Standard input	File input (cannot be used by the user)
Standard output	File output (cannot be used by the user)
Standard error output	Output of error messages

8.2.4 Validating preprocessor declaration statements

(1) Overview

The preprocessor features an option that allows you to use preprocessor declaration statements of the C compiler.

By specifying the -E option, you can execute the following functions with the preprocessor:

- Define literals and macros by using the #define declaration statement.
- Define literals and macros in include files that have been included with the #include statement[#].
- Execute conditional compilation based on #ifdef, #if, and other statements.
- Use macros to specify literals in embedded variable declarations.

#: SQL statements and SQL TYPE IS-type variable declarations cannot be specified in include files. (Otherwise, an error occurs during compilation because the preprocessor does not generate a header post source.)

(2) Usage examples

(a) Literal usage

Assume that the following embedded variable declaration is specified in a UAP source file:

```
#include "user.h"
EXEC SQL BEGIN DECLARE SECTION;
  char xchar1[MAX_CHAR_LEN];
EXEC SQL END DECLARE SECTION;
```

Also assume that the following literal is defined in the header file (user.h) that the UAP has included:

```
#define MAX_CHAR_LEN 256
```

In this case, the preprocessor reads the include file and uses the MAX_CHAR_LEN definition value to convert the embedded variable declaration to char xchar1[256]; before analyzing the UAP source file. However, macro literals cannot be used between the SQL prefix and the SQL terminator (in SQL statements).

Specify the directory path for include file retrieval as an option argument. The default directory of the C compiler does not need to be specified.

(b) Conditional compilation

You can use the #ifdef statement to select the SQL statement to be preprocessed. An example is shown as follows.

```
#ifdef DEF_SWITCH
    EXEC SQL DECLARE CUR1 CURSOR FOR SELECT * FROM TABLE1;
#else
```

```
    EXEC SQL DECLARE CUR1 CURSOR FOR SELECT * FROM TABLE2;
#endif
```

However, preprocessor declaration statements of the C compiler cannot be specified between the SQL prefix and SQL terminator (in SQL statements).

8.2.5 Dispensing with the embedded SQL declare section

(1) Overview

When you specify the `-E` option, the preprocessor can use variables that correspond to SQL data types as embedded variables regardless of where those variables are declared in the UAP source file. However, variables of the `register` storage class cannot be used as embedded variables.

The rules of the host language used to write the UAP source file determines the effective scope of a variable. Only UAP source files written in C or COBOL can use this function.

When this function is used, the following operations can be performed:

- Variable declarations can be used as embedded variables without having to be specified between an embedded SQL begin declaration (`BEGIN DECLARE SECTION`) and an embedded SQL end declaration (`END DECLARE SECTION`). Variable declarations can also be used together with embedded variables.
- The effective scope of a global variable, local variable, or function argument is determined by the syntax of the host language. If embedded variables have different effective scopes, they are discriminated as different embedded variables even if variables of the same name are declared. In this case, the preprocessor assumes that the innermost variable that includes the SQL statement that uses that variable was specified.

(2) Usage example

A usage example is shown as follows.

```
int fetchdata(long xprice){
  char    xpcode[5];
  char    xpname[17];
  char    xcolor[3];
  long    xstock;
:
  EXEC SQL
      DECLARE CR3 CURSOR FOR
         SELECT PCODE,PNAME,COLOR,SQUANTITY
         FROM STOCK WHERE PRICE=:xprice;
:
  EXEC SQL OPEN CR3 ;
:
  /*  heading */
  printf("    *****  STOCK TABLE LIST  *****\n\n");
 printf("   PRODUCT CODE  PRODUCT NAME         COLOR  PRICE
CURRENT STOCK\n");
  printf("    ----        ----------------  --  --------
--------\n");

  EXEC SQL WHENEVER SQLERROR GOTO END;
  EXEC SQL WHENEVER NOT FOUND GOTO END;

  /* FETCH */
  for(;;){
      EXEC SQL
          FETCH CR3 INTO :xpcode,:xpname,:xcolor,:xstock;
          printf("    %4s      %-16s  %2s  %8d  %8d\n",
                  xpcode, xpname, xcolor, xprice, xstock);
    }
  }
END:
```

8.2.6 Specifying pointers as environment variables

(1) Overview

In C, the `-E` option allows you to declare pointers as embedded variables. When you use this function, you can directly specify dynamically allocated areas in SQL statements.

For details about SQL preprocessor options, see the option descriptions in *8.2.2 Preprocessing in UNIX*, or *8.2.3 Preprocessing in Windows*. For details about SQL statements that can use pointers, see *8.2.8 Use of pointers, structures, and structure qualifiers when the -E2 or -E3 option of the preprocessor is specified*.

Declare pointer variables according to the C syntax. An example is shown as follows.

```
long  *xprice;
long  *xstock;
char  *xpname;
...
xprice  =  (long *)malloc(sizeof(long));
xstock  =  (long *)malloc(sizeof(long));
xpname  =  (char *)malloc(MAX_CHAR_LEN+1);
memset(xspname, ' ', MAX_CHAR_LEN);
xspname [MAX_CHAR_LEN] = '\0';
EXEC SQL FETC CUR1 INTO :xprice,:xstock,:xpname;
```

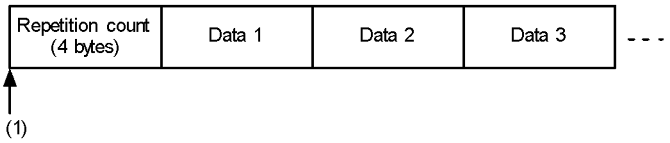
(2) Rules

1. When specifying a pointer variable, add a colon before the variable name in the SQL statement. An asterisk cannot be used.
2. The size of the value to be referenced becomes the size of the data type specified in the declaration. However, this does not apply to the fixed-length character string type (`CHAR`).
3. The data length of a fixed-length character string-type pointer is determined during execution, and not during preprocessing. The value size becomes the length (`strlen` (pointer variable)) up to the end (`\0`) of the character string that is stored in the area indicated by the pointer. When storing the search results of a single-row `SELECT` statement or a `FETCH` statement, you must first clear the entire area with a character other than `\0` before executing the SQL statement and then specify `\0` at the end of the area.
4. You must allocate the area that the pointer points to. If the pointer is a fixed-length character string-type pointer, allocate an extra byte to the area so that `\0` can be stored. If the pointer is an invalid value or if the area allocated for storing the data is too small, the operation is not guaranteed.
5. Pointers to pointers cannot be used.
6. Pointers to structures can be specified.
7. Pointers to classes cannot be used.
8. Pointers to arrays cannot be used. To use a pointer to an array, use a structure and declare the structure as follows:

```
struct {
long  xprice[50];
long  xstock[50];
char  xpname[50][17];
} *xrec_ptr;
```

(3) Notes on using repetition-type pointers in machines that use a RISC-type CPU

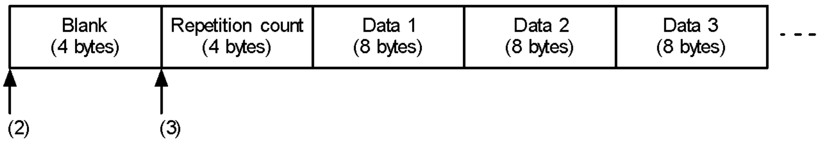
1. Because repetition column-type variables have the following structure, address (1), which coincides with a word boundary, must be specified in the pointer.



Normally, there is no problem because areas allocated with `malloc()` are already adjusted to word boundaries. However, if you calculate and allocate the memory address on your own, you must adjust the address to a word boundary.

If the address specified in the pointer does not coincide with a word boundary, a memory access exception occurs when the UAP uses macros for repetition column manipulation to reference or set data. For details about the structure of embedded variables in repetition columns, see *B.2(5) Expansion format of repetition columns*.

2. For `FLOAT`-type repetition columns, the data length of the repetition elements becomes larger than the area that stores the number of repetitions. The boundary address must be adjusted to the word length of the repetition items. In the pointer, specify address (2), which includes the leading free space.



The preprocessor creates a post source that automatically uses address (3), which is 4 bytes from the beginning, as the beginning of the repetition column. Macros that manipulate a `FLOAT`-type repetition column also use address (3) as the beginning of the column. If you use the SQL descriptor area to specify the address of the repetition column directly, specify address (3).

3. Because the maximum number of repetition elements is determined by the declared value, a memory access exception may occur if an area smaller than the declared value is allocated.

Normally, problems can be avoided by allocating the memory as shown in the following coding.

```
PD_MV_SINT(32) *ptr;     /* maximum element count 32 */
ptr = malloc(sizeof(*ptr));
EXEC SQL FETCH CUR1 INTO :ptr;
```

8.2.7 Referencing structures

(1) Overview

The preprocessor features an option that allows you to use a structure written in C to specify multiple embedded variables at one time.

Structures can be used as embedded variables in the following locations:

- `INTO` clause of the single-row `SELECT` or `FETCH` statement

- VALUES clause of the INSERT statement
- USING or INTO clause of the EXECUTE statement

For details about SQL preprocessor options, see the option descriptions in *8.2.2 Preprocessing in UNIX* or *8.2.3 Preprocessing in Windows*. For details about SQL statements that can use structures, see *8.2.8 Use of pointers, structures, and structure qualifiers when the -E2 or -E3 option of the preprocessor is specified*.

(2) Rules

1. When you specify a structure as an embedded variable, the preprocessor assumes that each member of the structure was specified as an embedded variable and generates the same post source it would generate if the members were specified separately. The member expansion sequence in the post source is the same as the member declaration sequence in the structure. The sequence of the retrieval items and columns in SQL statements that specify the structure must match the member sequence.
2. The members of a structure can also be specified individually as embedded variables.
3. Structures that contain a union cannot be used.
4. Structures that contain another structure cannot be used. However, structures that correspond to the variable-length character string type and the BINARY type can be used.

(3) Usage examples

- ■ Structure usage example

A structure usage example is shown as follows.

```
struct {
  char    xpcode[5];
  char    xpname[17];
  char    xcolor[3];
  long    xstock;
  long    xprice;
} xrec;
   ...
EXEC SQL
  DECLARE CR3 CURSOR FOR
    SELECT PCODE,PNAME,COLOR,SQUANTITY, PRICE FROM STOCK;
   ...
EXEC SQL OPEN CR3 ;

/*  heading */
printf("    *****  STOCK TABLE LIST  *****\n\n");
printf("   PRODUCT CODE  PRODUCT NAME         COLOR  PRICE
CURRENT STOCK\n");
  printf("    ----        ----------------  --  --------
--------\n");

  EXEC SQL WHENEVER SQLERROR GOTO END;
  EXEC SQL WHENEVER NOT FOUND GOTO END;

  /* FETCH */
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec;
    printf("    %4s      %-16s  %2s  %8d  %8d\n",
        xrec.xpcode, xrec.xpname, xrec.xcolor, xrec.xprice,
xrec.xstock);
    }
END:
            ...
```

- Usage example of a structure that contains indicator variables

When you use a structure as an embedded variable and also want to use indicator variables, declare the indicator variables in a structure as well. Associate the individual members of the indicator variable structure in declaration sequence with the individual members of the embedded variable structure. An example is shown as follows.

```
  struct {
      char    xpcode[5];
      char    xpname[17];
      char    xcolor[3];
      long    xstock;
      long    xprice;
  } xrec;
  struct {
      short    xpcode_ind;
      short    xpname_ind;
      short    xcolor_ind;
      short    xstock_ind;
      short    xprice_ind;
  } xrec_ind;
       ...
  /* FETCH */
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec :xrec_ind;
      printf("    %4s      %-16s  %2s  %8d  %8d\n",
         xrec.xpcode, xrec.xpname, xrec.xcolor, xrec.xprice,
xrec.xstock);
  }
      ...
```

- Example in which a pointer to a structure is specified as an embedded variable

You can also specify a pointer to a structure as an embedded variable. The area indicated by the pointer must be allocated beforehand.

```
struct  tag_xrec {
  char    xpcode[5];
  char    xpname[17];
  char    xcolor[3];
  long    xstock;
  long    xprice;
} *xrec_ptr;
struct tag_xrec_ind {
  short    xpcode_ind;
  short    xpname_ind;
  short    xcolor_ind;
  short    xstock_ind;
  short    xprice_ind;
} *xrec_ind_ptr;
      ...
/* FETCH */
xrec_ptr = (struct tag_xrec *)malloc(sizeof(struct tag_xrec));
xrec_ind_ptr = (struct tag_xrec_ind *)
                malloc(sizeof(struct tag_xrec_ind));
  for(;;){
    EXEC SQL FETCH CR3 INTO :xrec_ptr :xrec_ind_ptr;
      printf("    %4s      %-16s  %2s  %8d  %8d\n",
          xrec_ptr->xpcode, xrec_ptr->xpname, xrec_ptr->xcolor,
             xrec_ptr->xprice, xrec_ptr->xstock);
  }
        ...
```

8.2.8 Use of pointers, structures, and structure qualifiers when the -E2 or -E3 option of the preprocessor is specified

The table below shows whether pointers, structures, and structure qualifiers can be used when you specify the preprocessor's `/E2` or `/E3` option (`-E2` or `-E3` option in the UNIX edition).

A pointer refers to a variable declared with (*type-name* `*` *variable-name*). A structure refers to a variable declared with (`struct` *structure-name variable-name*). (However, structures that specify an SQL statement, as well as `VARCHAR`- and `BINARY`-type structures, are excluded.) A structure qualifier refers to a variable that has the `(`*structure*`.`*member-variable-name*`)` `(`*structure*`->`*member-variable-name*`)` format.

Table 8-14: Use of pointers, structures, and pointer qualifiers when the -E2 or -E3 option is specified

SQL statement that specifies embedded variable or indicator variable		Pointer	Structure	Structure qualifier
Data manipulation SQL statement	CALL statement	Y	N	Y
	DECLARE CURSOR	Y	N	Y
	DELETE statement	Y	N	Y
	DESCRIBE TYPE statement	Y	N	N
	EXECUTE statement with USING specification	Y	Y	Y
	EXECUTE statement with INTO specification	Y	Y	Y
	EXECUTE statement with USING specification	Y	Y	Y
	EXECUTE statement with BY specification	Y	N	Y
	EXECUTE IMMEDIATE statement with SQL character string location	Y	N	N
	EXECUTE IMMEDIATE statement with INTO specification	Y	Y	Y
	EXECUTE IMMEDIATE statement with USING specification	Y	Y	Y
	FETCH statement with INTO specification	Y	Y	Y
	FETCH statement with USING DESCRIPTOR specification	Y	N	Y
	INSERT statement with VALUES specification	Y	Y	Y
	OPEN statement	Y	N	Y
	PREPARE statement	Y	N	N
	SELECT statement with INTO specification	Y	Y	Y
	UPDATE statement	Y	N	Y

SQL statement that specifies embedded variable or indicator variable		Pointer	Structure	Structure qualifier
	FREE LOCATOR	Y	Y	Y
	SET	Y	N	Y
	ALLOCATE CURSOR	Y	N	N
Control SQL statement	CONNECT statement	Y	N	Y
	CONNECT statement with TO specification	Y	N	Y
	SET SESSION AUTHORIZATION statement	Y	N	Y
Embedded language	GET DIAGNOSTICS	N	N	N
	COMMAND EXECUTE	N	N	N
	INSTALL JAR	Y	N	N
	REPLACE JAR	Y	N	N
	REMOVE JAR	Y	N	N
	ALLOCATE CONNECTION HANDLE	N	N	N
	FREE CONNECTION HANDLE	N	N	N
	DECLARE CONNECTION HANDLE SET	N	N	N
	GET CONNECTION HANDLE	N	N	N

Legend:

Y: Can be specified.

N: Cannot be specified.

8.3 Compiling and linking

8.3.1 Libraries for compiling and linking

When executing compiling and linking, specify a library provided by HiRDB. *Tables 8-15* and *8-16* show the libraries to be specified for compiling and linking.

Table 8-15: Libraries to be specified for compiling and linking (in non-OLTP environment)

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file

HP-UX 11.0
HP-UX 11i
HP-UX 11i V2
(PA-RISC)
	Used	For a single thread:
		`libzclts.sl`
		For a single thread in 64-bit mode:
		`libzcltk64.sl`
		For multiple threads (DCE threads):
		`libzcltm.sl`
		For multiple threads (kernel threads):
		`libzcltk.sl`
		For 64-bit mode multiple threads (kernel threads):
		`libzcltk64.sl`
			For a single thread:
			`libclts.a`
			For a single thread in 64-bit mode:
			`libcltk64.a`
			For multiple threads (DCE threads):
			`libcltm.a`
			For multiple threads (kernel threads):
			`libcltk.a`
			For 64-bit mode multiple threads (kernel threads):
			`libcltk64.a`

	Not used	For 32-bit mode:
		`libzclt.sl`
		For 64-bit mode:
		`libzclt64.sl`
			For 32-bit mode:
			`libclt.a`
			For 64-bit mode:
			`libclt64.a`

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file

HP-UX 11i V2 (IPF)
HP-UX 11i V3 (IPF)
	Used	For a single thread:
		`libzclts.so`
		For a single thread in 64-bit mode:
		`libzclts64.so`
		For multiple threads (kernel threads):
		`libzcltk.so`
		For 64-bit mode multiple threads (kernel threads):
		`libzcltk64.so`
			--

	Not used	For 32-bit mode:
		`libzclt.so`
		For 64-bit mode:
		`libzclt64.so`
			--

Solaris	Used	For a single thread:
		`libzclts.so`
		For multiple threads (Solaris threads):
		`libzcltk.so`
		`libzcltm.so`
		For 64-bit mode multiple threads (Solaris threads):
		`libzcltk64.so`
			For a single thread:
			`libclts.a`
			For multiple threads (Solaris threads):
			`libcltk.a`
			`libcltm.a`
			For 64-bit mode multiple threads (Solaris threads):
			`libcltk64.a`

	Not used	For 32-bit mode:
		`libzclt.so`
		For 64-bit mode:
		`libzclt64.so`
			For 32-bit mode:
			`libclt.a`
			For 64-bit mode:
			`libclt64.a`

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file

AIX	Used	For a single thread:
		`libzclts.a`
		For a single thread in 64-bit mode:
		`libzcltk64.a`
		For multiple threads (POSIX threads):
		`libzcltk.a`
		For 64-bit mode multiple threads (POSIX threads):
		`libzcltk64.a`
			For a single thread:
			`libclts.a`
			For a single thread in 64-bit mode:
			`libcltk64.a`
			For multiple threads (POSIX threads):
			`libcltk.a`
			For 64-bit mode multiple threads (POSIX threads):
			`libcltk64.a`

	Not used	For 32-bit mode:
		`libzclt.a`
		For 64-bit mode:
		`libzclt64.a`
			For 32-bit mode:
			`libclt.a`
			For 64-bit mode:
			`libclt64.a`

Linux	Used	For a single thread:
		`libzclts.so`
		For multiple threads (POSIX threads):
		`libzcltk.so`
			For a single thread:
			`libclts.a`
			For multiple threads (POSIX threads):
			`libcltk.a`

	Not used	`libzclt.so`	`libclt.a`

Linux (EM64T)	Used	For a single thread:
		`libzclts.so`
		For a single thread in 64-bit mode:
		`libzcltk64.so`
		For multiple threads (POSIX threads):
		`libzcltk.so`
		For 64-bit mode multiple threads (POSIX threads):
		`libzcltk64.so`
			--

	Not used	For 32-bit mode:
		`libzclt.so`
		For 64-bit mode:
		`libzclt64.so`
			--

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file

Windows	Used	`PDCLTM32.LIB`
		`PDCLTM71.LIB`
		`PDCLTM80S.LIB`
		`PDCLTM90S.LIB`#
			--

	Not used	`CLTDLL.LIB`
		`PDCLTM71.LIB`
		`PDCLTM80S.LIB`
		`PDCLTM90S.LIB`#
			--

Windows (IPF)	Used	`PDCLTM64.LIB`	--
	Not used	`PDCLTM64.LIB`	--

Windows (x64)	Used	For 32-bit mode:
		`PDCLTM80S.LIB`
		`PDCLTM90S.LIB`#
		For 64-bit mode:
		`PDCLTM64.LIB`
		`PDCLTM90S64.LIB`#
			--

	Not used	For 32-bit mode:
		`PDCLTM80S.LIB`
		`PDCLTM90S.LIB`#
		For 64-bit mode:
		`PDCLTM64.LIB`
		`PDCLTM90S64.LIB`#
			--

Legend:

--: Not applicable

#

Created only for an XDS client.

Table 8-16: Libraries to be specified for compiling and linking (in OLTP environment)

Platform	Transaction registration method	Library name	
		Shared library file	Archive file

HP-UX 11.0
HP-UX 11i
HP-UX 11i V2 (PA-RISC)
	Dynamic registration	For a single thread:
		`libzcltx.sl`
		`libzcltxs.sl` (for multiple connections)
		For multiple threads (kernel threads):
		`libzcltxk.sl`
			For a single thread:
			`libcltxa.a`
			`libcltxas.a` (for multiple connections)
			For multiple threads (kernel threads):
			`libcltxak.a`

	Dynamic or static registration	For a single thread:
		`libzclty.sl`
		`libzcltys.sl` (for multiple connections)
		For multiple threads (kernel threads):
		`libzcltyk.sl`
			For a single thread:
			`libcltya.a`
			`libcltyas.a` (for multiple connections)
			For multiple threads (kernel thread):
			`libcltyak.a`

HP-UX 11i V2 (IPF) HP-UX 11i V3 (IPF)	Dynamic registration	For a single thread: `libzcltx.so` `libzcltxs.so` (for multiple connections)	--

	Dynamic registration or static registration	For a single thread:
		`libzclty.so`
		`libzcltys.so` (for multiple connections)
		For a 64-bit mode single thread (kernel threads):
		`libzclty64.so`
		`libzcltys64.so` (for multiple connections)
		For 64-bit mode multiple threads (kernel threads):
		`libzcltyk64.so`
			--

Platform	Transaction registration method	Library name	
		Shared library file	Archive file

Solaris	Dynamic registration	For a single thread:
		`libzcltx.so`
		`libzcltxs.so` (for multiple connections)
		For multiple threads (Solaris threads):
		`libzcltxk.so`
			For a single thread:
			`libcltxa.a`
			`libcltxas.a` (for multiple connections)
			For multiple threads (Solaris threads):
			`libcltxak.a`

	Dynamic or static registration	For a single thread:
		`libzclty.so`
		`libzcltys.so` (for multiple connections)
		For multiple threads (Solaris threads):
		`libzcltyk.so`
			For a single thread:
			`libcltya.a`
			`libcltyas.a` (for multiple connections)
			For multiple threads (Solaris threads):
			`libcltyak.a`

AIX	Dynamic registration	For a single thread:
		`libzcltx.a`
		`libzcltxs.a` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltxk.a`
			For a single thread:
			`libcltxa.a`
			`libcltxas.a` (for multiple connections)
			For multiple threads (POSIX threads):
			`libcltxak.a`

	Dynamic or static registration	For a single thread:
		`libzclty.a`
		`libzcltys.a` (for multiple connections)
		For a single thread in 64-bit mode:
		`libzclty64.a`
		`libzcltys64.a` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltyk.a`
			For a single thread:
			`libcltya.a`
			`libcltyas.a` (for multiple connections)
			For multiple threads (POSIX threads):
			`libcltyak.a`

Platform	Transaction registration method	Library name	
		Shared library file	Archive file

Linux	Dynamic registration	For a single thread:
		`libzcltx.so`
		`libzcltxs.so` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltxk.so`
			For a single thread:
			`libcltxa.a`
			`libcltxas.a` (for multiple connections)
			For multiple threads (POSIX threads):
			`libcltxak.a`

	Dynamic or static registration	For a single thread:
		`libzclty.so`
		`libzcltys.so` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltyk.so`
			For a single thread:
			`libcltya.a`

Linux (EM64T)	Dynamic registration	For a single thread:
		`libzcltx.so`
		`libzcltxs.so` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltxk.so`
			--

	Dynamic registration or static registration	For a single thread:
		`libzclty.so`
		`libzcltys.so` (for multiple connections)
		For a 64-bit mode single thread (kernel threads):
		`libzclty64.so`
		`libzcltys64.so` (for multiple connections)
		For multiple threads (POSIX threads):
		`libzcltyk.so`
		For 64-bit mode multiple threads (POSIX threads):
		`libzcltyk64.so`
			--

Platform	Transaction registration method	Library name	
		Shared library file	Archive file
Windows	Dynamic registration	--	--

	Dynamic or static registration	For a single thread:
		`PDCLTX32.LIB`
		`PDCLTXS.LIB` (for multiple connections)
		For multiple threads:
		`PDCLTXM.LIB`
			--

Windows (IPF)	Dynamic registration	--	--

	Dynamic registration or static registration	For a single thread:
		`PDCLTX64.LIB`
		`PDCLTXS64.LIB` (for multiple connections)
		For multiple threads:
		`PDCLTXM64.LIB`
			--

Windows (x64)	Dynamic registration	--	--

	Dynamic registration or static registration	For a single thread:
		`PDCLTX32.LIB`
		`PDCLTXS.LIB` (for OTS)
		For multiple threads:
		`PDCLTXM.LIB`
			--

Legend:

--: Not applicable

Note

For details about dynamic registration and static registration, see the description of methods for registering HiRDB to the transaction manager in the *HiRDB Version 9 Installation and Design Guide*.

8.3.2 Compiling and linking in UNIX

You must use a compiler that conforms to the language used for the UAP in which the SQL statements are embedded to compile and link edit a post-source program created by the SQL preprocessor.

This section explains how to specify commands for compiling and linking in the UNIX environment, for each language.

(1) C

Post-source programs in C must be compiled with a compiler that conforms to the ANSI C standards. Similarly, post-source programs in C++ must be compiled with a compiler that conforms to the C++ standards. The `cc` command (lowercase) is used to activate an ANSI C compiler, and the `CC` command (uppercase) is used to activate a C++ compiler. These commands can also be used to compile and link. This is the command format for activating a compiler:

`cc` [*options*] *file-name directory distributed-library*

Note

For C++, the underlined section must be changed to `CC`.

file-name

Specifies the name of the post-source file; the file identifier must be `.c`.

directory

Specifies the include directory (directory containing the header file of the library provided by HiRDB).

distributed-library

Specifies the library provided by HiRDB. Normally, a shared library should be used. An archive library should be used only to limit the version of the library used, or if a shared library cannot be used. If the UAP uses a thread, link a multi-connection library that corresponds to that thread.

options

Specifies the following options, as necessary:

`-o`

Specifies an optional name for the object file that is to be output; when this option is omitted, the filename is `a.out`.

`-I`

Specifies that an include directory is designated; compilation does not execute if this option is omitted.

`-Wl,+s`

Specifies that different HiRDB distribution library directories are used for UAP creation and for UAP execution. Specify this option if you use shared libraries with an HP-UX edition.

When different distribution library directories are used for linkage and for execution, the `SHLIB_PATH` environment variable must be used at the time of execution to set the directory containing the distribution library.

(a) Examples of command specification in C

Examples of command specification in C are shown as follows. In these examples, the underlined text specifies the HiRDB installation directory.

- For UAPs that support the 32-bit mode

Example 1: Shared library

- The post-source filename is `sample` and an executable form filename is not specified.

```
cc -I/HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

Example 2: Archive library

- The post-source filename is `sample` and the executable form filename is `SAMPLE`.

```
aCC +DD32 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

Example 3: HP-UX (IPF)

```
aCC -Ae -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

Example 4: For multiple threads in HP-UX (IPF)

```
aCC -Ae -mt -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk
```

Example 5: Linux (EM64T)

```
gcc -m32 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt
```

Example 6: For multiple threads in Linux (EM64T)

```
gcc -m32 -D_REENTRANT -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk
```

- For UAPs that support the 64-bit mode

Example 1: Shared library

- The post-source filename is `sample` and an executable form filename is not specified.

HP-UX 11.0

```
cc +DD64 -I/HiRDB/include sample.c -L/HiRDB/client/lib lzclt64
```

HP-UX (IPF)

```
aCC -Ae +DD64 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt64
```

Multiple threads in HP-UX (IPF)

```
aCC -Ae -mt +DD64 -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk64
```

Solaris 8 and Solaris 9

```
cc -xarch=v9 -I/HiRDB/include sample.c -L/HiRDB/client/lib -lzclt64 -lnsl -lsocket
```

AIX

```
xlc -q64 -I/HiRDB/include sample.c -Wl,-L/HiRDB/client/lib -lzclt64
```

Linux (EM64T)

```
gcc -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt64
```

Multiple threads in Linux (EM64T)

```
gcc -D_REENTRANT -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzcltk64
```

Note: Use `libzcltk64.so` even though the multi-connection facility is used by a single-threaded UAP.

Example 2: Archive library

- The post-source filename is `sample` and an executable form filename is not specified.

HP-UX 11.0

```
cc +DD64 -I/HiRDB/include sample.c /HiRDB/client/lib libclt64.a
```

Solaris 8, and Solaris 9

```
cc -xarch=v9 -I/HiRDB/include sample.c -L/HiRDB/client/lib -lclt64 -lnsl -lsocket
```

AIX

```
xlc -q64 -I/HiRDB/include sample.c -Wl,-L/HiRDB/client/lib -lclt64
```

(b) Examples of command specification in C++

Examples of command specification in C++ are shown as follows. In these examples, the underlined sections specify the HiRDB installation directory.

- For UAPs that support the 32-bit mode

Example 1: Shared library

- The post-source filename is `sample` and an executable form filename is not specified.

```
CC -I/HiRDB/include sample.C -L/HiRDB/client/lib -lzclt
```

Example 2: Archive library

- Post-source filename is `sample` and the executable form filename is `SAMPLE`.

```
CC -o SAMPLE -I/HiRDB/include sample.C /HiRDB/client/lib/libclt.a
```

- For UAPs that support the 64-bit mode

Example 1: Shared library

- The post-source filename is `sample` and an executable form filename is not specified.

HP-UX 11.0

```
CC +DA2.0w -I/HiRDB/include sample.C -L/HiRDB/client/lib lzclt64
```

Solaris 8, and Solaris 9

```
CC -xarch=v9 -I/HiRDB/include sample.C -L/HiRDB/client/lib -lzclt64 -lnsl -lsocket
```

AIX

```
xlc -q64 -I/HiRDB/include sample.C -Wl,-L/HiRDB/client/lib,-lzclt64
```

Example 2: Archive library

- The post-source filename is `sample` and an executable form filename is not specified.

HP-UX 11.0

```
CC +DA2.0w -I/HiRDB/include sample.C /HiRDB/client/lib libclt64.a
```

Solaris 8, and Solaris 9

```
CC -xarch=v9 -I/HiRDB/include sample.C -L/HiRDB/client/lib -lclt64 -lnsl -lsocket
```

AIX

```
xlc -q64 -I/HiRDB/include sample.C -Wl,-L/HiRDB/client/lib,-lclt64
```

(2) COBOL

Post-source programs in COBOL must be complied with the COBOL85, COBOL2002, MicroFocus COBOL, or SUN Japanese COBOL compiler. Post-source programs in OOCOBOL must be compiled with a compiler that conforms to the OOCOBOL standards.

The `ccbl` command is used to activate a COBOL85 compiler, and the `ocbl` command is used to activate an OOCOBOL compiler. These commands can also be used to compile and link. Following is the command format for activating a compiler:
`ccbl` `[options]` *file-name* *directory* *distributed-library*

Note

For OOCOBOL, the underlined section must be changed to `ocbl`.

file-name

Specifies the name of the post-source filename; the file identifier must be `.cbl`.

directory

Specifies the include directory (directory containing the header file of the library provided by HiRDB).

distributed-library

Specifies the COBOL library provided by HiRDB.

options

Specifies the following options, as necessary:

`-Wl,+s`

Specifies that different HiRDB distribution library directories is used for UAP creation and for UAP execution. Specify this option if you use shared libraries with an HP-UX edition.

When different distribution library directories are used for linkage and for execution, the `SHLIB_PATH` environment variable must be used at the time of execution to set the directory containing the distribution library.

`-o`

Specifies an optional name for the object file that is to be output; when this option is omitted, the filename is `a.out`.

The `-Kl` and `-Xb` options must not be specified; the `-Xc` option must not be specified together with the `-Hf`, `-Hv`, or `-V3` option.

environment-variable

Specifies the following environment variable:

`CBLLIB`

Include directory.

(a) Examples of command specification in COBOL

Examples of command specification in COBOL are shown as follows. In these examples, the underlined sections specify the HiRDB installation directory. For the HP-UX (IPF) edition of COBOL2002, `ccbl` becomes `ccbl2002`.

■ For UAPs in 32-bit mode

Example 1: Shared library

- The post-source filename is `sample`.

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl sample.cbl -L/HiRDB/client/lib -lzclt
```

Example 2: Archive library

- The post-source filename is `sample`.

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl sample.cbl /HiRDB/client/lib/libclt.a
```

■ For UAPs in 64-bit mode

Example 1: Shared library

- The post-source filename is `sample`.

```
CBLLIB=/HiRDB/include
export CBLLIB
ccbl2002 sample.cbl -L/HiRDB/client/lib -lzclt64
```

(b) Examples of command specification in OOCOBOL

Examples of command specification in OOCOBOL are shown as follows. In these examples, the underlined sections specify the HiRDB installation directory.

Example 1: Shared library

- The post-source filename is `sample`.

```
CBLLIB=/HiRDB/include
export CBLLIB
ocbl sample.ocb -L/HiRDB/client/lib -lzclt
```

Example 2: Archive library

- The post-source filename is `sample`.

```
CBLLIB=/HiRDB/include
export CBLLIB
ocbl sample.ocb /HiRDB/client/lib/libclt.a
```

(3) Note

When a UAP is created with the Solaris edition of HiRDB, that UAP cannot be connected to the HiRDB server when all of the following conditions are satisfied:

- The library that the HiRDB client uses is version 07-00-/C or later, and the library that the HiRDB server uses is earlier than 07-00-/C.
- The UAP and the HiRDB server to be connected are on the same machine.

In this case, use the client library (shared library) of the HiRDB server.

8.3.3 Compiling and linking in Windows

You must use a compiler that conforms to the language used for the UAP in which the SQL statements are embedded to compile and link edit a post-source program created by the preprocessor.

For the compilation and linkage methods in the Windows environment, see the manuals for the compilers applicable to the particular languages. This section explains the compilation and linkage options for each language. The section also contains instructions for Windows (x64).

(1) C

To compile post-source programs written in C, use Microsoft Visual C++.

To set options for compilation and linkage using Microsoft Visual C++, from the **Project** menu, choose **Setup**. (The setup method differs depending on the Microsoft Visual C++ version.)

The following table shows the items to be set in **Setup**.

Table 8-17: Items set with Setup

Item	Category	Category setting	Setting value
Compiler	Code generation	Alignment of structure members	8 bytes
		Runtime library to be used	Multi-thread#
	Preprocessor	Include file path	\\<u>*HiRDB*</u>\include
Linker	Input	Library	\\<u>*HiRDB*</u>\lib\cltdll

Note

The directory where HiRDB is installed is underlined.

#: All libraries except CLTDLL are created with multiple threads.

For Windows (IPF), only the 64-bit mode client library can be used. When creating a UAP in 64-bit mode, adhere to the following conditions:

- Alignment of configuration members: 8 bytes
- Runtime library used: Multi-thread DLL
- Include file path: \\<u>*HiRDB*</u>\INCLUDE
- Linkage library: \\<u>*HiRDB*</u>\LIB\PDCLTM64.LIB

(2) COBOL

To compile post-source programs in COBOL, use a compiler that conforms to the COBOL85 or COBOL2002 standards. To compile a post-source program in OOCOBOL, use a compiler that conforms to the OOCOBOL standard.

To set options for compilation and linkage using COBOL85 (version 1.0 or subsequent versions), choose **Edit**, then **Edit Project**.

For Windows, choose **Option**, then **Compile** and **Linker**.

For COBOL2002, from the **Project Setup** menu, choose the **Linker** tab.

Table 8-18 shows the item to be set with **Edit Project** in COBOL85. Do not specify the `-Kl`, `-Xb`, `-Bb`, or `-Fb` option. Also do not specify the `-Xc` option together with `-Hf`, `-Hv`, or `-V3`. *Table 8-19* shows the item to be set with **Project Setup** in COBOL2002.

Table 8-18: Item to be set with Edit Project in COBOL85

Item	Setting item	Setting value
Linkage option setting	Import library	`\`*HiRDB*`\lib\cltdll.lib`
	Compilation option	`/NOI` (Upper-case and lower-case characters are discriminated in the file identifier.)

Note

The directory where HiRDB is installed is underlined.

Table 8-19: Item to be set with Project Setup in COBOL2002

Item	Setting item	Setting value
Link	Library specification	`\`*HIRDB*`\lib\cltdll.lib`

Note

The directory where HiRDB is installed is underlined.

COBOL85 has an option to be set for **Compilation Environment**. The table below shows the item to be set for **Compilation Environment** in COBOL85. In COBOL2002, this item is set to an environment variable.

Table 8-20: Item to be set for Compilation Environment in COBOL85

Item	Setting item	Setting value
Environment variable setting#	`CBLLIB` variable	`\`*HiRDB*`\include`

Note

The directory where HiRDB is installed is underlined.

#: For COBOL2002, the item is set for **Environment Variable**.

(3) Instruction for Windows (x64)

Windows (x64) provides both 32-bit mode and 64-bit mode client libraries. To create a UAP in 32-bit mode, specify the compilation options and library for 32-bit mode. To create a UAP in 64-bit mode, specify the compilation options and library for 64-bit mode.

The following table lists the UAP creation conditions:

UAP creation	32-bit mode	64-bit mode
Alignment of structure members	Default (8 bytes)	Default (8 bytes)
Runtime library to be used	Multi-thread DLL	Multi-thread DLL
Include file directory	\HiRDB\INCLUDE	\HiRDB\INCLUDE
Linkage library#	\HiRDB\LIB\PDCLTM80S.LIB \HiRDB\LIB\PDCLTM90S.LIB	\HiRDB\LIB\PDCLTM64.LIB \HiRDB\LIB\PDCLTM90S64.LIB

Note: Specify an HiRDB installation directory for the area indicated by underscoring.

#: Specify regardless of whether the multi-connection facility is to be used. Note that PDCLTM90S.LIB and PDCLTM90S64.LIB can be specified only for an XDS client.

UAPs that use the XA interface cannot be created for 64-bit mode.

8.3.4 Compiling and linking when the multi-connection facility is used

(1) For multi-thread UAPs

This subsection explains how to compile and link normal, non-OLTP UAPs that use the multi-connection facility.

(a) In the UNIX environment

For HP-UX 11.0, Solaris, AIX, and Linux, link the libcltk.a and libzcltk.sl libraries. The table below shows the libraries to be linked when the multi-connection facility is used. For information about the libraries that must be linked for using multiple threads, see the manual for each operating system.

Table 8-21: Libraries to be linked when the multi-connection facility is used

UAP operating system	Thread used by UAP	Library to be linked	
		Shared library file	Archive file
HP-UX 11.0 HP-UX 11i HP-UX 11i V2 (PA-RISC)	Kernel thread (native thread)	For 32-bit mode: libzcltk.sl For 64-bit mode: libzcltk64.sl	For 32-bit mode: libcltk.a For 64-bit mode: libcltk64.a
	DCE thread	libzcltm.sl	libcltm.a

UAP operating system	Thread used by UAP	Library to be linked	
		Shared library file	Archive file

HP-UX 11i V2 (IPF)
HP-UX 11i V3 (IPF)
	Kernel thread (native thread)	For 32-bit mode:
		`libzcltk.so`
		For 64-bit mode:
		`libzcltk64.so`
			--

Solaris	Solaris thread (native thread)	For 32-bit mode:
		`libzcltm.so`
		`libzcltk.so`
		For 64-bit mode:
		`libzcltk64.so`
			For 32-bit mode:
			`libcltm.a`
			`libcltk.a`
			For 64-bit mode:
			`libcltk64.a`

AIX	POSIX thread	For 32-bit mode:
		`libzcltk.a`
		For 64-bit mode:
		`libzcltk64.a`
			For 32-bit mode:
			`libcltk.a`
			For 64-bit mode:
			`libcltk64.a`

Linux	POSIX thread	`libzcltk.so`	`libcltk.a`

Linux (EM64T)	POSIX thread	For 32-bit mode:
		`libzcltk.so`
		For 64-bit mode:
		`libzcltk64.so`
			--

Legend:

--: Not applicable

■ C examples

Examples of compiling and linking when the multi-connection facility is used by a UAP written in C are shown as follows. In these examples, the underlined sections specify the HiRDB installation library.

Example 1: Linking a UAP and a shared library in HP-UX 11.0

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -D_REENTRANT -D_HP_UX_SOURCE -D_POSIX_C_SOURCE=199506L
-L/HiRDB/client/lib/ -lzcltk -lpthread
```

Example 2: Linking a UAP and a 64-bit mode shared library in HP-UX 11.0

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c +DD64 -D_REENTRANT -D_HP_UX_SOURCE
-D_POSIX_C_SOURCE=199506L
-L/HiRDB/client/lib/ -lzcltk64 -lpthread
```

Example 3: Linking a Solaris-thread UAP and a shared library in Solaris

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -D_REENTRANT -L/HiRDB/client/lib/ -lzcltk -lthread
-lnsl -lsocket
```

Example 4: Linking a POSIX-thread UAP and a shared library in Solaris

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -D_REENTRANT -D_POSIX_PTHREAD_SEMANTICS
-L/HiRDB/client/lib/ -lzcltk -lthread -lnsl -lsocket
```

Example 5: Linking a Solaris-thread UAP and a 64-bit mode shared library in Solaris

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -xarch=v9 -D_REENTRANT -L/HiRDB/client/lib/
-lzcltk64 -lthread -lnsl -lsocket
```

Example 6: Linking a POSIX-thread UAP and a 64-bit mode shared library in Solaris

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -xarch=v9 -D_REENTRANT -D_POSIX_PTHREAD_SEMANTICS
-L/HiRDB/client/lib/ -lzcltk64 -lthread -lnsl -lsocket
```

Example 7: Linking a UAP and a shared library in Linux

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -D_REENTRANT -L/HiRDB/client/lib/ -lzcltk -lthread
```

Example 8: Linking a UAP and a shared library in AIX

- The post-source file name is `sample` and an executable form file name is not specified.

```
xlc_r -I/HiRDB/include sample.c -L/HiRDB/client/lib/ -lzcltk
```

Example 9: Linking a UAP and a 64-bit mode shared library in AIX

- The post-source file name is `sample` and an executable form file name is not specified.

```
xlc_r -I/HiRDB/include sample.c -q64 -L/HiRDB/client/lib/ -lzcltk64
```

■ COBOL examples

UAPs written in COBOL must be compiled and linked with a multi-thread version (03-01 or later) of the COBOL85 compiler.

During compilation, specify the `-Mt` option (for POSIX threads, you must also specify the `-Mp` option). If an object compiled with the `-Mt` option is linked with an object compiled without the `-Mt` option, the operation is not guaranteed. For details about compiling UAPs written in COBOL, see the *COBOL85 User's Guide*.

Examples of compiling and linking when the multi-connection facility is used by a UAP written in COBOL are shown as follows. In these examples, the underlined sections specify the HiRDB installation directory.

Example 1: Using DCE threads in HP-UX 11.0

- The post-source file name is `sample` and an executable form file name is not specified.

```
setenv CBLLIB /HiRDB/include
ccbl -Mt sample.cbl -L/HiRDB/client/lib/ -lzcltm -ldce
```

Example 2: Using kernel threads in HP-UX 11.0

- The post-source file name is `sample` and an executable form file name is not specified.

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lzcltk -lpthread
```

Example 3: Linking a Solaris-thread UAP and a shared library in Solaris

- The post-source file name is `sample` and an executable form file name is not specified.

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lzcltk -lpthread
```

Example 4: Linking a POSIX-thread UAP and a shared library in Linux

- The post-source file name is `sample` and an executable form file name is not specified.

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lcltk -lpthread
```

Example 5: Linking a POSIX-thread UAP with a shared library in AIX

- The post-source file name is `sample` and an executable form file name is not specified.

```
setenv CBLLIB /HiRDB/include
ccbl -Mt -Mp sample.cbl -L/HiRDB/client/lib/ -lzcltk -lpthread
```

(b) In the Windows environment

Link `PDCLTM32.LIB` instead of `CLTDLL.LIB`. For a UAP that uses an X/Open-compliant API under OLTP, link `PDCLTXM.LIB`.

■ C

This explanation assumes that Microsoft Visual C++ Version 4.2 is used. Select **Set** from the **Project** menu, and set the individual items. The table below shows the items that can be set in **Set**. If multiple threads are to be used, see the *Operating System* manual for details about the files that must be linked.

Table 8-22: Items to be set with Set

Item	Category	Category setting	Setting value
Compiler	Code generation	Alignment of structure members	8 bytes
		Run time library to be used	Multi-thread DLL
	Preprocessor	Include file path	`\HIRDB\INCLUDE`
Linker	Input	Library	`\HIRDB\LIB\PDCLTM32.LIB`

Note

The underlined sections specify the HiRDB installation library.

■ COBOL

UAPs written in COBOL must be compiled and linked with a multi-thread version of the COBOL85 compiler. The descriptions in this subsection assume that COBOL85 Version 5.0 is being used.

During compilation, specify the `-Mt` option in the Compiler Option dialog box. If an object compiled with the `-Mt` option is linked with an object compiled without the `-Mt` option, the operation is not guaranteed. For details about compiling UAPs written in COBOL, see the *COBOL85 User's Guide*.

The following table shows the items to be specified with the **Option** menu.

Table 8-23: Items to be specified with the Option menu

Submenu	Dialog box	Setting item	Setting value
Compiler	COBOL85 Compiler Option	COBOL85 compiler option	Check the `-Mt` item.
		Environment variable setting	`CBLLIB=`*C:\HIRDB*`\INCLUDE`
Linker	Linker Option Setting	Import/user-specified library	*C:\HIRDB*`\LIB\PDCLTM32`

Note: Specify the HiRDB installation directory in the underlined sections.

(2) For single-thread UAPs

This subsection explains how to compile and link single-thread UAPs that use the multi-connection facility. HP-UX 11.0 is used as an example for explanatory purposes.

(a) Compiling and linking in HP-UX 11.0

Link the `libclts.a` or `libzclts.sl` instead of `libclt.a` or `libzclt.sl`.

During compilation, the following compilation options and libraries for multiple threads cannot be specified:

- `-D_REENTRANT`
- `-DRWSTD_MULTI_THREAD`
- `-D_THREAD_SAFE`
- `-lcma`
- `-lpthread`

Also, `pthread` headers cannot be included.

■ **C examples**

Examples of compiling and linking when the multi-connection facility is used by a single-thread UAP written in C are shown as follows. In these examples, the underlined sections specify the HiRDB installation library.

Example 1: Shared library

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -L/HiRDB/client/lib/ -lzclts
```

Example 2: Archive library

- The post-source file name is `sample` and an executable form file name is not specified.

```
cc -I/HiRDB/include sample.c -L/HiRDB/client/lib/libclts.a
```

8.4 Notes

8.4.1 Notes on UAP execution

This subsection provides notes concerning UAP execution.

(1) Notes about the character code classification

To execute UAPs, you must specify the LANG environment variable, or PDLANG or PDCLTLANG in the client environment definition according to the HiRDB server's character code classification (specified by the pdsetup command in the UNIX edition and the pdntenv command in the Windows edition). If the HiRDB server and HiRDB client use different character code classifications, an error occurs during UAP execution. The following table shows the values of LANG and PDLANG for each platform.

Table 8-24: Values of LANG and PDLANG for each platform

Character encoding type#1		HP-UX	Solaris	AIX	Linux
lang-c	LANG	C	C	C	C
	PDLANG	--	--	--	--
sjis	LANG	ja_JP.SJIS	ja_JP.PCK	Ja_JP	Any#2
	PDLANG	--	--	--	SJIS
ujis	LANG	ja_JP.eucJP	ja	ja_JP	ja_JP.eucJP#3
	PDLANG	--	--	--	--
chinese	LANG	chinese-s	Any#2	Any#2	Any#2
	PDLANG	CHINESE	CHINESE	CHINESE	CHINESE
utf-8	LANG	Any#2	Any#2	Any#2	Any#2
	PDLANG	UTF-8	UTF-8	UTF-8	UTF-8

Character encoding type[#1]		HP-UX	Solaris	AIX	Linux
chinese-gb18030	LANG	Any[#2]	Any[#2]	Any[#2]	Any[#2]
	PDLANG	CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030	CHINESE-GB18030

Legend:

--: There is no setting

#1

This is the HiRDB server's character code classification as specified in pdsetup or pdntenv.

#2

If the platform in use supports the corresponding character encoding, specify that character encoding in the LANG environment variable. If the platform does not support the character encoding, specify C.

#3

ja_JP is handled in the same way as ja_JP.eucJP.

(2) Notes about SHLIB_PATH

For executing a UAP, add $PDDIR/client/lib to SHLIB_PATH. If you are using a different platform, replace SHLIB_PATH with the corresponding environment variable for that platform.

(3) Notes about a recovery-unnecessary front-end server in the HiRDB system

If the HiRDB system includes a recovery-unnecessary front-end server and it processes a UAP that uses the X/Open XA interface to connect to the recovery-unnecessary front-end server, it will not be possible to execute that UAP's SQL statements. In such a case, you must specify the PDFESHOST and PDSERVICEGRP client environment definitions and connect to a front-end server that is not a recovery-unnecessary front-end server.

(4) Notes about Windows Server 2003 or later

Because of changes in the socket security specifications in Windows Server 2003, Windows Vista, Windows Server 2008, and Windows 7, a receive port already in use by another program might be assigned to a communication socket acquired by the HiRDB client. If a receive port that is already in use is assigned, the HiRDB server's message when connection is established is sent to the other program. As a result, the HiRDB client does not receive the message and the KFPA11732-E error (receive timeout) occurs. For this reason, it is important to take preventive measures to ensure that a receive port will not be duplicated when UAPs are executed in a Windows Server

2003 or later environment.

Tables 8-25 and *8-26* show the conditions that result in receive port duplication and the corrective measures to be taken in Windows Server 2003 or later.

Table 8-25: Conditions resulting in duplicate receive ports and corrective measures (1/2)

Client that starts later			Client that starts first					
			Library version earlier than 08-03		Library version 08-03 or later			
					PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO	
			PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted
Library version earlier than 08-03		`PDCLTRCVPORT` specified	--	--	--	--	--	--
		`PDCLTRCVPORT` omitted	--	--	--	--	--	D1
Library version 08-03 or later	PDCLTBINDLOOPBACKADDR=YES	`PDCLTRCVPORT` specified	--	--	--	--	--	--
		`PDCLTRCVPORT` omitted	--	--	--	--	--	--
	PDCLTBINDLOOPBACKADDR=NO	`PDCLTRCVPORT` specified	--	--	--	--	--	--
		`PDCLTRCVPORT` omitted	--	D1	--	--	--	--

Client that starts later			Client that starts first					
			Library version earlier than 08-03		Library version 08-03 or later			
					PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO	
			PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted
Type 4 JDBC driver	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT specified	--	--	--	--	--	--
		PDCLTRCVPORT omitted	--	--	--	--	--	D1
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT specified	--	--	--	--	--	--
		PDCLTRCVPORT omitted	--	D1	--	D1	--	D1

Table 8-26: Conditions resulting in duplicate receive ports and corrective measures (2/2)

Client that starts later		Client that starts first, HiRDB server, or other program				
		Type 4 JDBC driver				HiRDB server, or other program
		PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO		
		PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted	
Library version earlier than 08-03	PDCLTRCVPORT specified	--	--	--	--	--
	PDCLTRCVPORT omitted	--	--	--	D1	D2

Client that starts later			Client that starts first, HiRDB server, or other program				
			Type 4 JDBC driver				HiRDB server, or other program
			PDCLTBINDLOOPBACKADDR=YES		PDCLTBINDLOOPBACKADDR=NO		
			PDCLTRCVPORT specified	PDCLTRCVPORT omitted	PDCLTRCVPORT specified	PDCLTRCVPORT omitted	
Library version 08-03 or later	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT specified	--	--	--	--	--
		PDCLTRCVPORT omitted	--	--	--	D1	D2
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT specified	--	--	--	--	--
		PDCLTRCVPORT omitted	--	D1	--	D1	D2
Type4 JDBC driver	PDCLTBINDLOOPBACKADDR=YES	PDCLTRCVPORT specified	--	--	--	--	--
		PDCLTRCVPORT omitted	--	--	--	D1	D2
	PDCLTBINDLOOPBACKADDR=NO	PDCLTRCVPORT specified	--	--	--	--	D2
		PDCLTRCVPORT omitted	--	D1	--	D1	D2

Legend:

--

No preventive measure is needed because there will be no receive port duplication.

D1

The receive port will be duplicated. Specify different ranges of port numbers in the PDCLTRCVPORT client environment variable for the clients that start first and those that start later so that the port numbers used by these programs are not duplicated.

D2

The receive port will be duplicated. Specify in the PDCLTRCVPORT client environment variable for the client that starts later a port number that will not be used by the HiRDB server or other programs, thus ensuring that the port numbers used by programs will not be duplicated.

8.4.2 Executing UAPs that use an X/Open-based API (TX_function)

A UAP that uses an X/Open-based API (TX_function) uses a dedicated library. To compile and link edit such a UAP, the library dedicated to the TX_function and the library provided by HiRDB must be coupled.

Linux for AP8000 clients cannot execute UAPs that use an X/Open-based API (TX_ function).

(1) Preprocessing a UAP that uses an X/Open-based API (TX_ function)

This item describes notes on executing a UAP in a HiRDB system that is linked with TP1/LiNK (transaction control).

Linking with TP1/LiNK is possible when both the HiRDB server and the HiRDB client are for Windows.

(a) UAP preprocessing and linkage

If a UAP is to be executed in a TP1/LiNK environment, execute UAP preprocessing and linkage as described as follows.

- Preprocessing

During SQL preprocessor execution, specify one of the following options:

- /XAD: Specify this option to create a UAP written in COBOL as a DLL.
- /XA: Specify this option in all other cases.

Command specification example for C

```
PDCPP SAMPLE /XA
```

Command specification example for COBOL

```
PDCBL SAMPLE.ECB /XAD
```

- Linkage

Link the following library to the UAP:

- `%PDDIR%\CLIENT\LIB\PDCLTX32.LIB`

Do not link `CLTDLL.LIB`.

(2) Using OpenTP1

UAP compilation and linkage when OpenTP1 is used are explained here. For details about compilation and linkage using OpenTP1, see the *OpenTP1 Programming Reference C Language* manual or the *OpenTP1 Programming Reference COBOL Language* manual.

(a) C

■ **Transaction control object file creation**

When OpenTP1 is used to create a UAP that accesses HiRDB, it is necessary to create a transaction control object file with the OpenTP1 `trnmkobj` operation command; following is the specification:
`trnmkobj -o` *control-object-filename* `-r HiRDB_DB_server`

Example

- The transaction control object filename is `control`.

```
trnmkobj -o control -r HiRDB_DB_server
```

■ **Compilation and linkage**

The following is specified to compile and link a UAP that uses API.

To use a shared library:

```
/usr/bin/cc -c -l$DCDIR/include -I/HiRDB/include filename.c
  /usr/bin/cc -o UAP-executable-form-filename UAP-filename.o
  $DCDIR/spool/trnrmcmd/userobj/control-object-filename.o
  -L/HiRDB/client/lib -lzclty -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a, default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Notes

1. The directory for installing HiRDB is underlined.
2. With the HP-UX edition, the `-W1, +s` option must be specified if different directories are used for the libraries provided by HiRDB during UAP creation and UAP execution. When *different* distribution library directories are used for linkage and for execution, the `SHLIB_PATH` environment variable must be used at the time of execution to set the directory containing the distribution library.
3. Depending on the method of OLTP registration that is used, the HiRDB system provides the following four libraries for UAPs that use an X/

Open-compatible API. The library name that is specified for the linkage must match the applicable OLTP registration method.

`-lzcltx` (dynamic registration)

`-lzclty` (static/dynamic registration)

`-lzcltxs` (dynamic registration when the multi-connection facility is used)

`-lzcltys` (static/dynamic registration when the multi-connection facility is used)

In static/dynamic registration, the registration type can be switched to static registration or dynamic registration by the switch registered to TM. For details about the registration procedure, see the explanation of registering a HiRDB system in the transaction manager in the *HiRDB Version 9 Installation and Design Guide*.

4. To create a UAP that uses an X/Open-compatible API in AIX, specify `-brtl` in the linkage option.

Example

The filename (UAP name) is `sample`, UAP executable form filename is `SAMPLE`, and the transaction control object filename is `control`.

```
/usr/bin/cc -c -I$DCDIR/include -I/HiRDB/include sample.c
  /usr/bin/cc -o SAMPLE sample.o
  $DCDIR/spool/trnrmcmd/userobj/control.o
  -L/HiRDB/client/lib -lzclty -L$DCDIR/lib
  -Wl,-B,immediate -Wl,-a,default
  -lbetran -L/usr/lib -ltactk -lbsd -lc
```

To use an archive library:

```
/usr/bin/cc -c -l$DCDIR/include -I/HiRDB/include filename.c
  /usr/bin/cc -o UAP-executable-form-filename UAP-filename.o
  $DCDIR/spool/trnrmcmd/userobj/control-object-filename.o
  -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a, default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Notes

1. The directory for installing HiRDB is underlined.
2. The archive library provided by HiRDB (`-lcltxa`) must be specified before the library provided by OpenTP1 (`-lbetran`).
3. Depending on the method of OLTP registration that is used, the HiRDB system provides the following two libraries for UAPs that use an X/Open-compatible API. The library name that is specified for the linkage must match the applicable OLTP registration method.

`-lcltxa` (dynamic registration)

`-lcltya` (static/dynamic registration)

In static/dynamic registration, the registration type can be switched to static registration or dynamic registration by the switch registered to TM. For details about the registration procedure, see the explanation of registering a HiRDB system in the transaction manager in the *HiRDB Version 9 Installation and Design Guide*.

Example

The filename (UAP name) is `sample`, UAP executable form filename is `SAMPLE`, and the transaction control object filename is `control`.

```
/usr/bin/cc -c -I$DCDIR/include -I/HiRDB/include sample.c
  /usr/bin/cc -o SAMPLE sample.o
  $DCDIR/spool/trnrmcmd/userobj/control.o
  -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib
  -Wl,-B,immediate -Wl,-a,default
  -lbetran -L/usr/lib -ltactk -lbsd -lc
```

(b) COBOL

■ Transaction control object file creation

When OpenTP1 is used to create a UAP that accesses HiRDB, it is necessary to create a transaction control object file with the OpenTP1 `trnmkobj` operation command; following is the specification:
`trnmkobj -o` *control-object-filename* `-r HiRDB_DB_SERVER`

Example

- The transaction control object filename is `control`.

`trnmkobj -o control -r HiRDB_DB_SERVER`

■ Compilation and linkage

The following is specified to compile and link a UAP that uses API.

To use a shared library:

```
ccbl -o UAP-executable-form-filename -Mw filename.cbl
  $DCDIR/spool/trnrmcmd/userobj/control-object-filename.o
  -L/HiRDB/client/lib -lzclty -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a, default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Notes

1. The directory for installing HiRDB is underlined.
2. With the HP-UX edition, the `-W1, +s` option must be specified if different

directories are used for the libraries provided by HiRDB during UAP creation and UAP execution. When different distribution library directories are used for linkage and for execution, the `SHLIB_PATH` environment variable must be used at the time of execution to set the directory containing the distribution library.

3. Depending on the method of OLTP registration that is used, the HiRDB system provides the following 4 libraries for UAPs that use an X/Open-compatible API. The library name that is specified for the linkage must match the applicable OLTP registration method.

   `-lzcltx` (dynamic registration)

   `-lzclty` (static/dynamic registration)

   `-lzcltxs` (multi-connection facility is used with dynamic registration)

   `-lzcltys` (multi-connection facility is used with static/dynamic registration)

   In static/dynamic registration, the registration type can be switched to static registration or dynamic registration by the switch registered to TM. For details about the registration procedure, see the explanation of registering a HiRDB system in the transaction manager in the *HiRDB Version 9 Installation and Design Guide*.

4. To create a UAP that uses an X/Open-compatible API in AIX, specify `-brtl` in the linkage option.

Example

The filename (UAP name) is `sample`, the UAP executable form filename is `SAMPLE`, and the transaction control object filename is `control`.

```
ccbl -o SAMPLE -Mw sample.cbl
  $DCDIR/spool/trnrmcmd/userobj/control.o
  -L/HiRDB/client/lib -lzclty -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a,default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

To use an archive library:

```
ccbl -o UAP-executable-form-filename -Mw filename.cbl
  $DCDIR/spool/trnrmcmd/userobj/control-object-filename.o
  -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a,default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

Notes

1. The directory for installing HiRDB is underlined.
2. The archive library provided by HiRDB (`-lcltxa`) must be specified before

the library provided by OpenTP1 (`-lbetran`).

3. Depending on the method of OLTP registration that is used, the HiRDB system provides the following two libraries for UAPs that use an X/Open-compatible API. The library name that is specified for the linkage must match the applicable OLTP registration method.

   `-lcltxa` (dynamic registration)

   `-lcltya` (static/dynamic registration)

   In static/dynamic registration, the registration type can be switched to static registration or dynamic registration by the switch registered to TM. For details about the registration procedure, see the explanation of registering an HiRDB system in the transaction manager in the *HiRDB Version 9 Installation and Design Guide*.

Example

The filename (UAP name) is `sample`, the UAP executable form filename is `SAMPLE`, and the transaction control object filename is `control`.

```
ccbl -o SAMPLE -Mw sample.cbl
  $DCDIR/spool/trnrmcmd/userobj/control.o
  -L/HiRDB/client/lib -lcltxa -L$DCDIR/lib -Wl,-B,immediate
  -Wl,-a,default -lbetran -L/usr/lib -ltactk -lbsd -lc
```

(3) Using TPBroker for C++

UAP compilation and linkage when TPBroker for C++ is used are explained here. The UAP is assumed to use the multi-thread XA interface. For details about compilation and linkage using TPBroker for C++, see the *TPBroker User's Guide* manual. The libraries that are dedicated to the multi-thread XA interface support only C and C++.

(a) Transaction control object file creation

When TPBroker for C++ is used to create a UAP that accesses HiRDB, it is necessary to create a transaction control object file with the TPBroker for C++ `tsmkobj` operation command; following is the specification:

```
tsmkobj -o control-object-filename -r HiRDB_DB_SERVER
```

(b) Compilation and linkage

The following is specified to compile and link a UAP.

```
aCC +inst_implicit_include +DAportable -c -I$TPDIR/include
  -I$TPDIR/include/dispatch -I/HiRDB/include -D_REENTRANT
  -D_HP_UX_SOURCE -D_POSIX_C_SOURCE=199506Lfilename.c
aCC +inst_implicit_include +Daportable -o
  UAP-executable-form-filename  UAP-filename.o
  $TPDIR/otsspool/XA/control-object-filename.o
  -L/HiRDB/client/lib -lzcltxk -L$TPDIR/lib -Wl,+s -lots_r
  -lorb-r
  -Wl,-B,immediate -Wl,-a,default -L/usr/lib -lpthread
```

Notes

1. The directory for installing the HiRDB client is underlined.
2. With the HP-UX edition, the `-Wl, +s` option must be specified if different directories are used for the libraries provided by HiRDB during UAP creation and UAP execution. When different distribution library directories are used for linkage and for execution, the `SHLIB_PATH` environment variable must be used at the time of execution to set the directory containing the distribution library.
3. Depending on the method of OLTP registration that is used (dynamic registration or static registration), the HiRDB system provides two libraries (`-lzcltxk` and `-lzcltyk`) for UAPs that use an XA interface that supports multiple threads. The library name that is specified for the linkage must match the applicable OLTP registration method.

(4) Using TUXEDO

UAP compilation and linkage when TUXEDO is used are explained here. The libraries that are dedicated to the XA interface support only C and C++.

(a) UNIX

- Load module construction for the transaction manager server (TMS)

```
buildtms -r HiRDB_DB_SERVER -o TMSload-module-filename
```

- Load module construction for the TUXEDO system server

```
buildserver -r HiRDB_DB_SERVER -s service-name
  -o server-load-module-filename  -f server-filename.o
```

- Load module creation for the TUXEDO client

```
buildclient -o client-load-module-name  -f client-filename.c
```

(b) Windows

- Load module construction for the transaction manager server (TMS)

```
set LINK=/EXPORT:_imp_pdtxa_switch=pdtxa_switch
  /EXPORT:_inp_pdtxa_switch_y=pdtxa_switch_y
buildtms -r HiRDB_DB_SERVER -o TMSload-module-filename
```

- Load module construction for the TUXEDO system server

```
set LINK=/EXPORT:_imp_pdtxa_switch=pdtxa_switch
  /EXPORT:_inp_pdtxa_switch_y=pdtxa_switch_y
buildserver -r HiRDB_DB_SERVER -s service-name
  -o server-load-module-filename -f server-filename.obj
```

- Load module creation for the TUXEDO client

```
buildclient -o client-load-module-name -f client-filename.c
```

(5) Using TP1/EE (limited to UNIX)

UAP compilation and linkage when TP1/EE is used are explained here. For details about the TP1/EE commands, see the *TP1/Server Base Enterprise Option User's Guide*.

(a) C

- Object file creation for resource manager linking

  When using TP1/EE to create a UAP that accesses HiRDB, you must use a TP1/EE operation command to create an object file for linking with the resource manager. Use the `eetrnmkobj` command to create the object file.

  The specification for creating an object file for linking with the resource manager is as follows:

```
eetrnmkobj -o name-of-object-file-for-resource-manager-linking -r HiRDB_DB_SERVER  \
           -s RM-switch-name -O RM-object-file-name  \
           -i header-path-provided-by-HiRDB
```

Example

The name of the object file for linking with the resource manager is `control`, and the static registration method is used to create the object file.

```
eetrnmkobj -o control -r HiRDB_DB_SERVER -s pdtxa_switch_y  \
           -O /HiRDB/client/lib/libzcltyk.sl -i /HiRDB/include
```

- Compilation and linkage

The following is specified to compile and link a UAP that uses the multi-thread XA interface.

- Using a shared library

```
/usr/vac/bin/xlc_r -o executable-file-name $DCDIR/lib/ee_main.o
    resource-manager-linking-object -brtl -bdynamic -L/HiRDB/lib -L/HiRDB/client/lib
    -L$DCDIR/lib -lpthread -lisode -lc_r -ldl -lzcltyk -lee -lee_rm
    -lbetran2 -ltactk
```

Notes

1. The directory for installing the HiRDB client is underlined.
2. When the multi-thread XA interface is used, specify `-lzcltyk` as the library for the TP1/EE UAP, and specify a corresponding name as the library name to be specified during linkage. For details about the registration procedure, see the *HiRDB Version 9 Installation and Design Guide*.

Example

The name of the UAP executable file is `SAMPLE`, and the name of the object file for linking with the resource manager is `control`.

```
/usr/vac/bin/xlc_r -o SAMPLE $DCDIR/lib/ee_main.o control.o
     -brtl -bdynamic -L/HiRDB/lib -L/HiRDB/client/lib -L$DCDIR/lib
     -lpthread -lisode -lc_r -ldl -lzcltyk -lee -lee_rm -lbetran2 -ltactk
```

(b) COBOL

■ Creating an object file for resource manager linking

When TP1/EE is used to create a UAP that accesses HiRDB, a TP1/EE operation command must be used to create an object file for linking with the resource manager. The `eetrnmkobj` command is used for this purpose.

The following is specified to create an object file for resource manager linking:

```
eetrnmkobj -o name-of-object-file-for-resource-manager-linking -r HiRDB_DB_SERVER  \
           -s RM-switch-name -O RM-related-object-file-name  \
           -i header-path-provided-by-HiRDB
```

Example

The name of the object file for resource manager linking is `control`, and the file is created by the static registration method.

```
eetrnmkobj -o control -r HiRDB_DB_SERVER -s pdtxa_switch_y  \
          -O /HiRDB/client/lib/libzcltyk.sl -i /HiRDB/include
```

- Compilation and linkage

  For details about compilation and linkage of a UAP that uses the multi-thread XA interface, see the *TP1/Server Base Enterprise Option User's Guide*.

8.4.3 UAP execution using the Unicode functionality of COBOL2002

For UAPs that use the Unicode functionality of COBOL2002, you can use Japanese data items for storing UTF-16 character data as embedded variables.

(1) Character codes of SQL-related character data

The following table shows the character codes and character data in UAPs that use the Unicode functionality of COBOL2002.

Table 8-27: Character codes and character data in UAPs that use the Unicode functionality of COBOL2002

Character data in UAP		Character codes
SQL statements		UTF-8
Embedded variables	Alphanumeric data items	UTF-8
	Japanese data items	UTF-16LE or UTF-16BE

(2) Character code conversion performed by the HiRDB server

The following table describes the character code conversion performed by the HiRDB server when UAPs that use the Unicode functionality of COBOL2002 execute SQL statements.

Table 8-28: Character code conversion performed by the HiRDB server when UAPs that use the Unicode functionality of COBOL2002 execute SQL statements.

Character data in UAP	Character code conversion performed by the HiRDB server when SQL statements are executed
SQL statement	No conversion is performed because the client and server both use UTF-8 character codes.

Character data in UAP		Character code conversion performed by the HiRDB server when SQL statements are executed
Embedded variable	Alphanumeric data item	Character codes are converted if the character set of character data used by the server that is subject to assignment and comparison processing is UTF-16.#
	Japanese data item	Character codes are converted if the character set of character data used by the server that is subject to assignment and comparison processing is the default character set (UTF-8).# If the pre-conversion data contains single-byte characters, the single-byte characters (double-byte characters in UTF-16) are stored. Data in embedded variables is treated as having the character data type (`CHAR` or `VARCHAR`) for which the character set name, `UTF-16LE` or `UTF-16BE`, has been specified according to the `-XU16` option.

#

For details about the data types that can be converted (assigned and compared), see the manual *HiRDB Version 9 SQL Reference*.

(3) SQL execution conditions

In order to execute the SQL statements by a UAP that uses the Unicode functionality of COBOL2002, all the following conditions must be satisfied:

- The HiRDB server's default character set is UTF-8.
- The `-XU16` option is specified when the SQL preprocessor is executed.
- No embedded SQL statement contains characters in JIS X0213 level 3 or 4 kanji codes.
- `NOUSE` (default value) is specified in the `PDCLTCNVMODE` client environment definition.

(a) HiRDB server's default character set

If you specify `utf-8` as the character code classification in the `pdntenv` command (in the UNIX edition, `pdsetup` command), the HiRDB server's default character set is set to UTF-8.

(b) Preprocessing options

For details about the `-XU16` option, see *Table 8-5 Preprocessing options (for COBOL in the UNIX environment)* or *Table 8-11 Preprocessing options (for COBOL in the Windows environment)*.

(c) Limitations to the characters in embedded SQL statements

An error will occur during preprocessing if an embedded SQL statement in the UAP source contains characters in JIS X0213 level 3 or 4 kanji codes. For character data stored in embedded variables, you must use characters in the range supported by the Unicode functionality of COBOL2002.

(d) Client environment definition

You must specify `NOUSE` in the `PDCLTCNVMODE` client environment definition in order to prevent the HiRDB client from performing character code conversion.

8.4.4 Converting UAPs created with XDM/RD or UNIFY2000

If a UAP created with XDM/RD or UNIFY2000 possesses SQL compatibility, it can use HiRDB by executing the SQL preprocessor (a UAP that does not possess SQL compatibility must be rewritten). For details about the SQLs used by HiRDB, see *1.2.2 List of SQL statements usable in HiRDB*. For details about SQLs, see the *HiRDB Version 9 SQL Reference* manual.

The following table shows the portability of UAPs from XDM/RD or UNIFY2000.

Table 8-29: UAP transferability from XDM/RD or UNIFY2000

Creation system	How to operate database	Transferability	Transfer condition
XDM/RD	Embedding SQL statements	T	Re-execution of SQL preprocessor
	Module type	--	NA
	CALL type	--	NA
UNIFY2000	Embedding SQL/A statements	T	Re-execution of SQL preprocessor
	Using RHLI	--	NA

T: Can be transferred

--: Cannot be transferred

NA: Not applicable

8.4.5 Actions required when HiRDB is upgraded

The APIs provided by HiRDB are upgradeable. In general, this means that there is no need to update UAPs when you upgrade your HiRDB.

However, if the following conditions are applicable, you must preprocess, compile, and link all UAPs again:

- You will be using new HiRDB features.
- A UAP created in C language or COBOL will be executed by a HiRDB client that uses a different version of the OS.

Chapter

9. Java Stored Procedures and Java Stored Functions

This chapter explains the procedures for creating and executing Java stored procedures and Java stored functions that code procedures in Java. Note that Linux for AP8000 clients cannot use Java stored procedures and Java stored functions.

This chapter contains the following sections:

9.1 Overview
9.2 Procedure from external Java stored routine creation to execution
9.3 Sample external Java stored routine programs
9.4 Notes about Java program creation
9.5 Notes about testing and debugging
9.6 Notes about JAR file creation

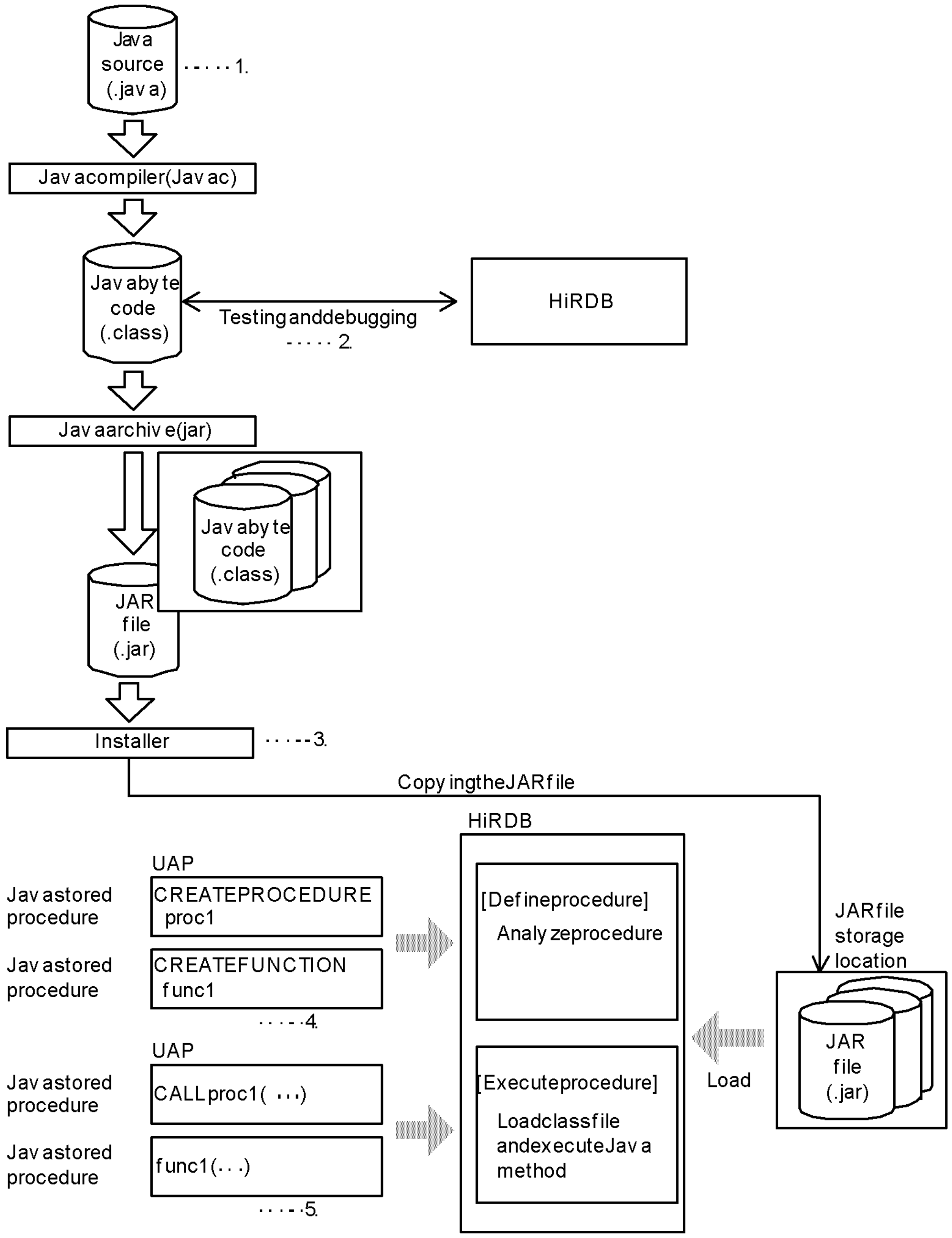
9.1 Overview

Stored procedures and stored functions that are coded in Java are called *Java stored procedures* and *Java stored functions*, respectively. They are referred to collectively in this chapter as *external Java stored routines*.

External Java stored routines cannot be used on all HiRDB operation platforms. For details, see the section that describes environments in which Java stored procedures and Java stored functions can be used in the *HiRDB Version 9 System Operation Guide*.

The following figure shows the procedure from creation to execution of an external Java stored routine.

Figure 9-1: Procedure from creation to execution of an external Java stored routine



Explanation

1. Create an external Java stored routine. For details, see *9.2.1 Creating an external Java stored routine*.
2. Test and debug the Java stored routine as a client AP. For details, see *9.2.1 Creating an external Java stored routine*.
3. Register the JAR file in HiRDB. For details, see *9.2.2 Registering the JAR file*.
4. Define the external Java stored routine. For details, see *9.2.3 Defining the external Java stored routine*.
5. Execute the external Java stored routine. For details, see *9.2.4 Executing the external Java stored routine*.

■ Features of external Java stored routines

1. **There is no overhead for communication between the server and a client.**

   Because external Java stored routines are processed at the server in the same manner as SQL stored procedures and SQL stored functions, there is no communication overhead between server and client.

2. **The procedure or function itself can be coded in Java.**

   Because Java is used as the programming language, more advanced control is available than with SQL.

3. **Java stored routines are supported by different types of DBMS.**

   Because Java is a platform-independent programming language, a program created in Java can be run in a variety of DBMSs that support external Java stored routines.

4. **Debugging is easy.**

   To debug an SQL stored procedure or an SQL stored function, you need to execute it at the server. In contrast, an external Java stored routine can be debugged at the client, including testing of database accesses, as long as the Java debugger is installed at the client.

■ Preparations for external Java stored routine execution

To execute an external Java stored routine, you must have first installed the JDBC driver. For details about installing the JDBC driver, see *17.1 Installation and environment setup*.

9.2 Procedure from external Java stored routine creation to execution

To create and execute an external Java stored routine:

1. Create the external Java stored routine.
2. Register the JAR file.
3. Define the external Java stored routine.
4. Execute the external Java stored routine.
5. Re-register and delete the JAR file.

9.2.1 Creating an external Java stored routine

To code an external Java stored routine:

1. Code the Java program (create a Java file).
2. Compile (create a Class file).
3. Test and debug.
4. Archive in JAR format (create the JAR file).

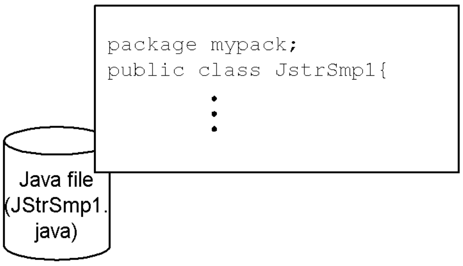
(1) Coding a Java program (creating a Java file)

Code the program that is to be registered as an external Java stored routine.

For notes about Java program coding, see *9.4 Notes about Java program creation*.

The following figure shows an example of Java program coding.

Figure 9-2: Example of Java program coding



```
package mypack;
public class JstrSmp1{
        ⋮
```

(2) Compiling (creating a Class file)

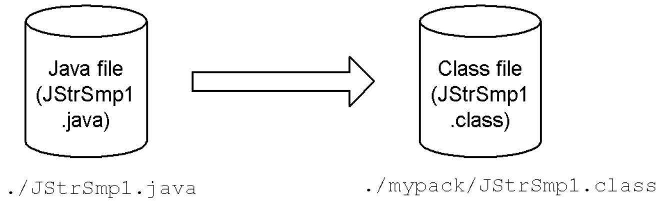
Create a Class file from the Java file using the `javac` command.

The following figure shows an example of compilation.

Figure 9-3: Example of compilation

Command entry

```
% javac -d ./ JStrSmp1.java
```



Explanation

If you specified `package` for the Java file, specify the `-d` option during compilation. When the Java file is compiled, a directory with the specified package name is created and a Class file is created in that directory.

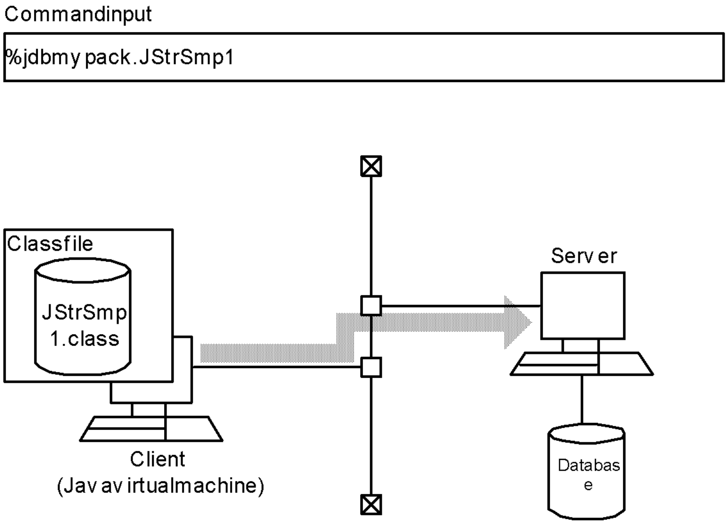
(3) Testing and debugging

Execute the compiled file on the client's Java virtual machine to test and debug it.

For notes about testing and debugging, see *9.5 Notes about testing and debugging*.

The following figure provides an overview of testing and debugging.

Figure 9-4: Overview of testing and debugging



Commandinput

```
%jdbmy pack.JStrSmp1
```

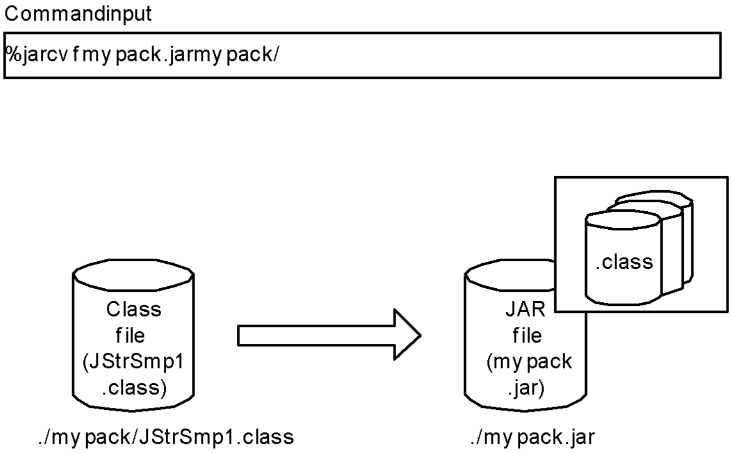
(4) Archiving in JAR format (creating a JAR file)

Use the `jar` command to create a JAR file from multiple Class files.

For notes about JAR file creation, see *9.6 Notes about JAR file creation*.

The following figure shows an example of archiving in the JAR format.

Figure 9-5: Example of archiving in JAR format



Commandinput

```
%jarcv fmy pack.jarmy pack/
```

9.2.2 Registering the JAR file

Register (copy) the created JAR file onto the server.

There are three ways to do this:

- **Executing SQL "INSTALL JAR"**

  Use a UAP or the database definition utility to specify and execute `INSTALL JAR`.

- **Executing the pdjarsync command**

  Execute the `pdjarsync` command (specifying the `-I` option).

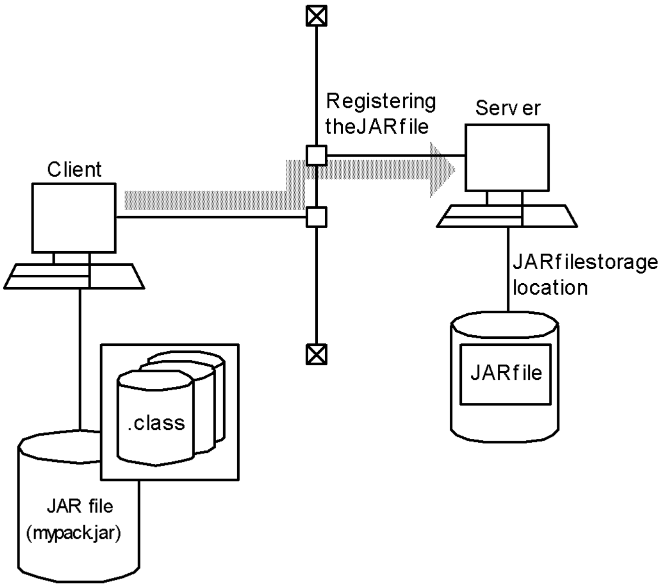
  Only the HiRDB administrator can execute the `pdjarsync` command.

- **Calling a Java method for installation**

  Call a Java method for installation, which is `Jdbh_JARReInstall` in the `Jdbh_JARAccss` class, to register the JAR file.

The following figure provides an overview of JAR file registration.

Figure 9-6: Overview of JAR file registration



Reference note:

You use the REPLACE JAR SQL statement to re-register a JAR file. You use the REMOVE JAR SQL statement to delete a JAR file. The HiRDB administrator must use the pdjarsync command to re-register or delete a JAR file.

Note that only the user who registered a JAR file or the HiRDB administrator can re-register or delete it.

9.2.3 Defining the external Java stored routine

You use CREATE PROCEDURE or CREATE FUNCTION to define an external Java stored routine. CREATE PROCEDURE or CREATE FUNCTION defines the association of a Java method with the procedure name or function name.

- Java stored procedure

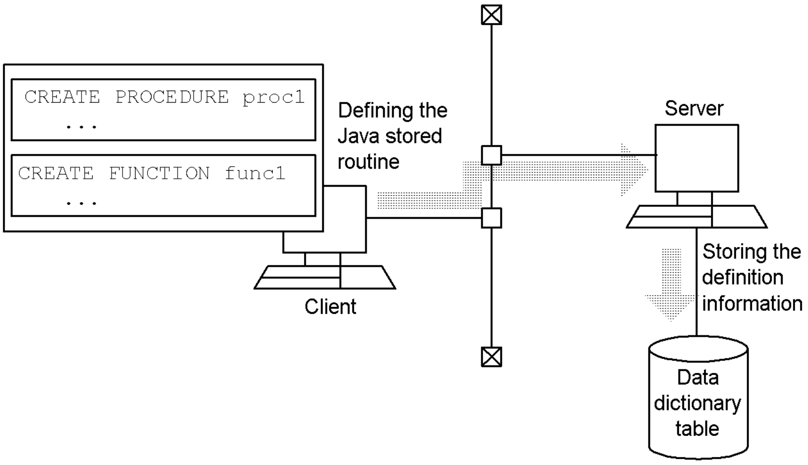
  Use CREATE PROCEDURE to register a Java method as a Java stored procedure.

- Java stored function

  Use CREATE FUNCTION to register a Java method as a Java stored function.

The following figure shows an example of an external Java stored routine definition.

Figure 9-7: Example of an external Java stored routine definition



Defining public routines

To use an external Java stored routine defined by another user, you must specify its owner's authorization identifier and the routine identifier when you call the stored routine from within the UAP.

However, if the external Java stored routine was defined as a *public routine* by executing CREATE PUBLIC PROCEDURE or CREATE PUBLIC FUNCTION, then there is no need to specify the owner's authorization identifier when the stored routine is called from within a UAP (only the routine identifier needs to be specified).

Redefining external Java stored routines

You use ALTER PROCEDURE or ALTER ROUTINE to redefine an external Java stored routine, such as when you need to make changes to the Java program.

Deleting external Java stored routines

You use DROP PROCEDURE or DROP FUNCTION to delete an external Java stored routine.

To delete a public routine, use DROP PUBLIC PROCEDURE or DROP PUBLIC FUNCTION. Note that a public routine can be deleted only by the user who defined it or by a user with the DBA privilege.

9.2.4 Executing the external Java stored routine

You use the CALL statement or a function call to execute an external Java stored routine. When the CALL statement or SQL statement specifying a function call executes, a Java method is called as the external Java stored routine and executed at the server's Java virtual machine.

- Java stored procedures

  Use the CALL statement to execute a Java method as a Java stored procedure.

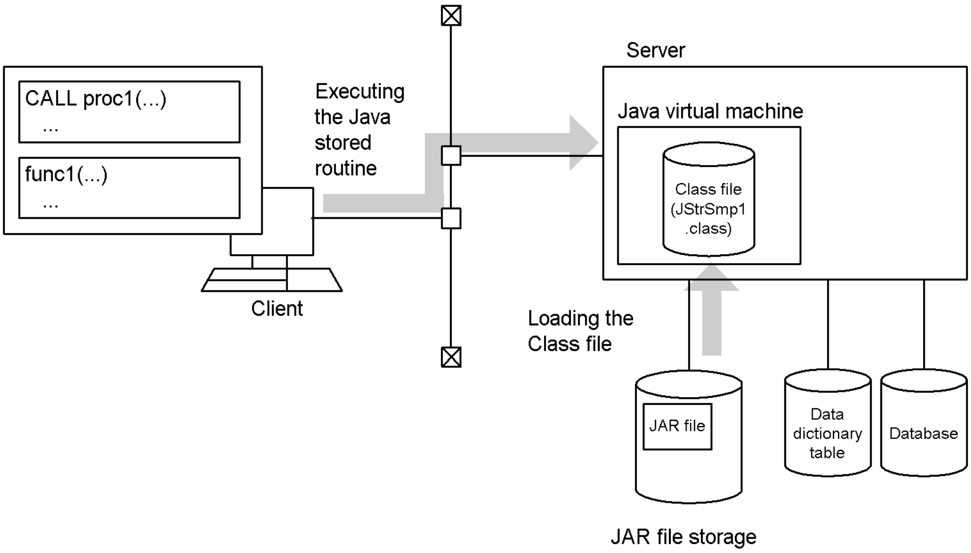
- Java stored functions

  Use an SQL statement specifying a function call to execute a Java method as a Java stored function.

For details about the CALL statement and function calls, see the *HiRDB Version 9 SQL Reference* manual.

The following figure shows an example of executing an external Java stored routine.

Figure 9-8: Example of execution of an external Java stored routine



9.3 Sample external Java stored routine programs

9.3.1 Sample program

This section presents an example of a stored procedure that uses the `SELECT` statement to retrieve BLOB data stored in the `pics` table, zips the data (compresses the data in the ZIP format), then returns it to the calling program.

■ Definition of the Java stored procedure

```
CREATE PROCEDURE get_pic   ...........................................1
  (IN pic_num INTEGER, OUT pic_data BLOB)   ...........................1
  LANGUAGE JAVA   .....................................................2
  EXTERNAL NAME 'mypack.jar:JStrPics.getZippedPic(int, byte[][])'   ....3
  PARAMETER STYLE JAVA;   ............................................4
```

Explanation

1. Defines the procedure name and parameters.
2. Specifies `LANGUAGE`.
3. Associates with the Java method.
4. Specifies `PARAMETER STYLE`.

■ Body of the Java stored procedure

```
import java.sql.*;
import java.io.*;
import java.util.zip.*;

public class JStrPics{   .............................................1
  public static void getZippedPic(int jpic_num, byte[][] jpic_data)   ..2
    throws SQLException, IOException{   ..............................3
    Connection con = DriverManager.getConnection(   ...................4
                     "jdbc:hitachi:hirdb");         ...................4

    PreparedStatement pstmt = con.prepareStatement   ..................5
        ("select p_name,p_data from pics where p_num = ?");  ..........5
```

```
pstmt.setInt(1, jpic_num);   .....................................5
ResultSet rs = pstmt.executeQuery();   ...........................6

String name;   ...................................................7
byte[] srcPic;   .................................................7

while(rs.next()){
  name = rs.getString("p_name");   ...............................8
  srcPic = rs.getBytes("p_data");   ..............................9
}

ByteArrayOutputStream baos = new ByteArrayOutputStream();   .......10
ZipOutputStream zos = new ZipOutputStream(baos);   ................10
ByteArrayInputStream bais = new ByteArrayInputStream(srcPic);   ...10
ZipEntry ze = new ZipEntry(name);   ..............................10
zos.putNextEntry(ze);   ..........................................10

int len = 0;   ...................................................10
byte[] buff = new byte[1024];   ..................................10
while((len = bais.read(buff)) != -1){   ..........................10
zos.write(buff, 0, len);   .......................................10
}   ..............................................................10

zos.closeEntry();   ..............................................11
bais.close();   ..................................................11
zos.close();   ...................................................11

jpic_data[0] = baos.toByteArray();   .............................12
baos.close();   ..................................................12
```

```
    return;   ..........................................................13
  }
}
```

Explanation

1. Defines the class name.
2. Defines the method name and parameter name.
3. Defines an action to be taken in the event of an exception.
4. Obtains the `Connection` object. (Because the Java stored procedure runs within the calling program's connection, there is no increase in the number of users connected to HiRDB.)
5. Preprocesses the `SELECT` statement.
6. Executes the `SELECT` statement and obtains the result set.
7. Declares variables.
8. Obtains the value of the `p_name` column from the result set.
9. Obtains the value of the `p_data` column from the result set.
10. Compresses the data in the `srcPic` array in the ZIP format and stores it in the `zos` stream.
11. Closes the input and output streams.
12. Specifies the `byte` column of the `baos` stream in the method's `OUT` parameter.
13. End of method execution.

■ Execution of the Java stored procedure

```
import java.sql.*;
import java.io.* ;

public class Caller{   ..................................................1
  public static void main(String[] args)   ..............................2
    throws SQLException, IOException{   .................................3
    Connection con = DriverManager.getConnection(    ......................4
                     "jdbc:hitachi:hirdb","USER1","PASS1");   ............4
    CallableStatement cstmt = con.prepareCall("{call get_pic(?,?)}");   ..5
    cstmt.setInt(1, 10);   ..............................................5
    cstmt.registerOutParameter(2, java.sql.Types.LONGVARBINARY);   .......5


    cstmt.executeUpdate();   ............................................6
    byte[] getPic = cstmt.getBytes(2);   ................................7


    ......


  }
}
```

Explanation

This sample Java application program calls a Java stored procedure.

1. Defines the class name.
2. Defines the method name and parameter name.
3. Defines an action to be taken in the event of an exception.
4. Obtains the `Connection` object. (Because the connection to HiRDB is established by obtaining a `Connection` object, there is an increase in the number of users.)
5. Preprocesses the `CALL` statement.
6. Executes the `CALL` statement.
7. Obtains the `OUT` parameter of the `byte` array type.

9.3.2 Sample external Java stored routines provided with HiRDB

(1) Sample 1

This sample Java stored procedure obtains a calendar of the specified year and month.

- **External Java procedure (filename: sample1.java)**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/************************************************************************/
/* name = HiRDB 06-00 sample program of Java stored procedure 1         */
/************************************************************************/
import java.lang.*;
import java.util.*;

/************************************************************************/
/* name = sample_1 class                                                */
/************************************************************************/
public class sample1 {
    /*==================================================================*/
    /* name = main method for debugging                                 */
    /*==================================================================*/
    public static void main(java.lang.String[] args) {
        java.lang.Integer year = new Integer(args[0]);
        java.lang.Integer month = new Integer(args[1]);
        java.lang.String calendar[] = new String[1];
        calendar(year, month, calendar);
        System.out.println(calendar[0]);
    }

    /*====================================================================*/
    /* name = sample_1 method                                             */
    /*====================================================================*/
    public static void calendar(java.lang.Integer year,
                               java.lang.Integer month,
                                java.lang.String[] calendar) {
```

```
        int DayOfWeek;          // first day of the week in the specified month
        int week;               // For linefeed control
        int wyear = year.intValue();    // Year work
        int wmonth = month.intValue();  // Month work

        // Creating the calendar header
        calendar[0] = "         " +  wyear + " / " + wmonth + "\n";
        calendar[0] += "Sun Mon Thu Wed Tue Fri Sat\n";

        // Generating the calendar object
        Calendar target_cal = new GregorianCalendar(wyear, wmonth - 1, 1);

        // Calculating the first day of the week in the specified month
        DayOfWeek = target_cal.get(Calendar.DAY_OF_WEEK);

        // Specifying spaces up to the first day of the week
        for (week = 1; week < DayOfWeek; week++) {
            calendar[0] += "    ";
        }
        // Specifying the date
        for (;
             target_cal.get(Calendar.MONTH) == wmonth - 1;
             target_cal.add(Calendar.DATE, 1), week++) {
            // Adjusting spaces according to the date and digits
            if (target_cal.get(Calendar.DATE) < 10) {
                calendar[0] += "  " + target_cal.get(Calendar.DATE);
            } else {
                calendar[0] += " " + target_cal.get(Calendar.DATE);
            }
            // Specifying padding characters between dates

            if (week == 7) {

                calendar[0] += "\n";

                week = 0;

            } else {

                calendar[0] += " ";

            }

        }


        return;

    }

}
```

The following is an example of defining and executing a Java stored procedure using the external Java procedure shown above:

- **Compiling the Java file (for HP-UX)**

```
javac sample1.java
```

- **Creating the JAR file (for HP-UX)**

```
jar -cvf sample1.jar sample1.class
```

- **Registering the JAR file in HiRDB (using INSTALL JAR SQL statement)**

```
INSTALL JAR 'sample1.jar' ;
```

- **Defining the Java stored procedure**

```
CREATE PROCEDURE calendar(IN pyear INT, IN pmonth INT, OUT
                                        calendar VARCHAR(255))
  LANGUAGE JAVA
  EXTERNAL NAME 'sample1.jar:sample1.calendar(java.lang.Integer,
                 java.lang.Integer,java.lang.String[])
                 returns void'
  PARAMETER STYLE JAVA
end_proc;
```

- **Executing the Java stored procedure**

```
CALL calendar(?,?,?)
```

(2) Sample 2

This example accepts the specified date as a processing range and updates the total for the `goods_no` column in that range.

The example assumes that the table is defined as follows:

```
CREATE TABLE master_t1 (goods_no int,total_quantity dec(17,2))
CREATE TABLE tran_t1(goods_no int,quantity_1
dec(17,2),entrydate date)
```

- **External Java procedure (filename: sample2.java)**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/************************************************************************/
/* name = HiRDB 06-00 Java stored sample 2                              */
/************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;

/************************************************************************/
/* name = sample_2 class                                                */
/************************************************************************/
public class sample2 {
    /*==================================================================*/
    /* name = main method for debugging                                 */
    /*==================================================================*/
    public static void main(String args[]) throws SQLException {
        java.sql.Date fromdate = Date.valueOf("1996-06-01");
        java.sql.Date todate = Date.valueOf("1996-06-30");
        try {
        // Registering the Driver class
            Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
        } catch (ClassNotFoundException ex) {
                System.out.println("\n*** ClassNotFoundException caught ***\n");
                ex.printStackTrace();
                System.out.println ("");
                System.out.println("\n*************************************\n");
                return;
        }

        jproc1(fromdate, todate);
    }
    /*==================================================================*/
    /* name = sample_2 method                                           */
    /*==================================================================*/
    public static void jproc1(java.sql.Date fromdate, java.sql.Date todate)
                                                         throws SQLException {

        java.lang.Integer x_goods_no;
        java.math.BigDecimal x_quantity_1, x_total_quantity;
        try {
            // Creating a connection object (CONNECT not issued within the Java
                                                          procedure)
            java.sql.Connection con =
              DriverManager.getConnection("jdbc:hitachi:hirdb");

            con.setAutoCommit(false);  // Suppressing automatic commit

            // SELECT (stmt1) preprocessing
            java.sql.PreparedStatement stmt1 =
              con.prepareStatement("SELECT goods_no, quantity_1
                                                   , entrydate FROM tran_t1
```

```
  WHERE entrydate BETWEEN ? AND ? ORDER BY entrydate");
// SELECT (stmt2) preprocessing (outside the loop)
java.sql.PreparedStatement stmt2 =
  con.prepareStatement("SELECT total_quantity FROM master_t1
  WHERE goods_no = ?");

// INSERT (stmt3) preprocessing (outside the loop)
java.sql.PreparedStatement stmt3 =
  con.prepareStatement("INSERT INTO master_t1 VALUES(?, ?)");

// UPDATE (stmt4) preprocessing (outside the loop)
java.sql.PreparedStatement stmt4 =
  con.prepareStatement("UPDATE master_t1 SET total_quantity = ?
  WHERE goods_no = ?");

 // Specifying SELECT (stmt1) input parameters
stmt1.setDate(1, fromdate);
stmt1.setDate(2, todate);

// Executing SELECT (stmt1)
java.sql.ResultSet rs1 = stmt1.executeQuery();
while (rs1.next()) {
    // Obtaining the retrieval result of SELECT (stmt1)
    x_goods_no = (Integer)rs1.getObject("goods_no");
    x_quantity_1 = rs1.getBigDecimal("quantity_1");

    // Specifying SELECT (stmt2) input parameter
    stmt2.setObject(1, x_goods_no);
```

```
                // Executing SELECT (stmt2)
                java.sql.ResultSet rs2 = stmt2.executeQuery();

                // Checking whether or not goods_no has been registered to
                                                             determine action
                if (!rs2.next()) {      // Not registered ==> Add a new entry
                    // Closing the SELECT (stmt2) cursor before updating
                    rs2.close();

                    // Specifying INSERT (stmt3) input parameters
                    stmt3.setObject(1, x_goods_no);
                    stmt3.setBigDecimal(2, x_quantity_1);

                    // Executing INSERT (stmt3)
                    stmt3.executeUpdate();
                } else {                // Registered ==> Update the
                                                          existing entry
                    // Obtaining the current value
                    x_total_quantity = rs2.getBigDecimal("total_quantity");

                    // Incrementing
                    x_total_quantity = x_total_quantity.add(x_quantity_1);

                    // Closing SELECT (stmt2) cursor before updating
                    rs2.close();

                    // Specifying UPDATE (stmt4) input parameters
                    stmt4.setBigDecimal(1, x_total_quantity);
                    stmt4.setObject(2, x_goods_no);
                    stmt4.executeUpdate() ;


                }
            }

            // Closing SELECT (stmt1) cursor
            rs1.close();

            // Releasing each statement object
            stmt1.close();
            stmt2.close();
            stmt3.close();
            stmt3.close();

            // Disconnecting
            con.close();

        } catch (SQLException ex) { // SQL error handling procedure
            SQLException fast_ex = ex;

            System.out.println("\n***** SQLException caught *****\n");
            while (ex != null) {
                System.out.println ("SQLState: " + ex.getSQLState ());
                System.out.println ("Message:  " + ex.getMessage ());
                System.out.println ("Vendor:   " + ex.getErrorCode ());
                ex.printStackTrace();
                ex = ex.getNextException ();
                System.out.println ("");
            }
            System.out.println("*******************************\n");


            throw fast_ex;

        }

        return;

    }

}
```

The following is an example of defining and executing a Java stored procedure using the external Java procedure shown above:

- **Compiling the Java file (for HP-UX)**

```
javac sample2.java
```

- **Creating the JAR file (for HP-UX)**

```
jar cvf sample2.jar sample2.class
```

- **Registering the JAR file in HiRDB (using INSTALL JAR SQL statement)**

```
INSTALL JAR 'sample2.jar' ;
```

- **Defining the Java stored procedure**

```
CREATE PROCEDURE jproc1(IN fromdate DATE, IN todate DATE)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample2.jar:sample2.jproc1(java.sql.Date, java.sql.Date)
                 returns void'
  PARAMETER STYLE JAVA
end_proc;
```

- **Executing the Java stored procedure**

```
CALL jproc1(IN ?,IN ?)
```

(3) Sample 3

This example compresses and decompresses BLOB data using `gzip` and `ungzip`.

- **External Java function (filename: sample3.java)**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */

/* LICENSED MATERIAL OF HITACHI,LTD. */
```

```
/************************************************************************/
/* name = HiRDB 06-00 Java stored sample 3                              */
/************************************************************************/
import java.util.zip.*;
import java.io.*;

public class sample3 {
    private final static int BUFF_SIZE = 4096;

    /*==================================================================*/
    /* name = main method for debugging                                 */
    /*==================================================================*/
    public static void main(String[] args) throws IOException {
        // Obtaining input data
        String sin = args[0];
        byte[] bin = args[0].getBytes();
        System.out.println("input data : " + sin);

        // GZIP(BLOB)
        byte[] bwork = gzip(bin);
        System.out.println("gzip(BLOB)  : " +
                           bin.length +  "=>" + bwork.length +
                           "(" + (bwork.length * 100 / bin.length) + "%): " +
                           "");

        // GUNZIP(BLOB)
        byte[] bout = gunzip(bwork);
        System.out.println("gunzip(BLOB): " +
                           bwork.length +  "=>" + bout.length +
                           "(" + (bout.length * 100 / bwork.length) + "%): " +
                           new String(bout));

        return;
    }

    /*==================================================================*/
    /* name = sample_3 method [gzip(BLOB)]                              */
    /*==================================================================*/
    public static byte[] gzip(byte indata[]) {
        // Creating a stream for output of compressed data
        ByteArrayOutputStream baos = new ByteArrayOutputStream();

        // Output of compressed data
        try {
            GZIPOutputStream zos = new GZIPOutputStream(baos);
            zos.write(indata, 0, indata.length);
            zos.close();
            baos.close();
        } catch (IOException ex) {
            System.out.println("gzip(BLOB): IOException: " + ex);
        ex.printStackTrace();
        }

        // Creating a byte array after compressing return value
        byte[] outdata = baos.toByteArray();
        return outdata;
    }
```

```
/*==================================================================*/
/* name = sample_3 method [gunzip(BLOB)]                            */
/*==================================================================*/
public static byte[] gunzip(byte[] indata) {
    int rlen;                    // Actual input/output length
    byte[] buff = new byte[BUFF_SIZE]; // Input/output buffer

    // Creating a stream for input of compressed data
    ByteArrayInputStream bais = new ByteArrayInputStream(indata);
    // Creating a stream for output of decompressed data
    ByteArrayOutputStream baos = new ByteArrayOutputStream();

    // Input of compressed data and output of decompressed data
    try {
        GZIPInputStream zis = new GZIPInputStream(bais);

        while ((rlen = zis.read(buff, 0, buff.length)) >= 0) {
          baos.write(buff, 0, rlen);
        }

        zis.close();
        bais.close();
        baos.close();
    } catch (IOException ex) {
        System.out.println("gunzip(BLOB): IOException: " + ex);
    ex.printStackTrace();
    }

    // Creating a byte array after decompressing return value
```

```
        byte[] outdata = baos.toByteArray();


        return outdata;
    }
}
```

The following is an example of defining and executing a Java stored procedure using the external Java function shown above:

- **Compiling the Java file (for HP-UX)**

```
javac sample3.java
```

- **Creating the JAR file (for HP-UX)**

```
jar -cvf sample3.jar sample3.class
```

- **Registering the JAR file in HiRDB (using INSTALL JAR SQL statement)**

```
INSTALL JAR 'sample3.jar' ;
```

- **Defining the Java stored procedure**

```
CREATE FUNCTION gzip(indata BLOB(1M)) RETURNS BLOB(1M)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample3.jar:sample3.gzip(byte[]) returns byte[]'
  PARAMETER STYLE JAVA
end_proc;
CREATE FUNCTION gunzip(indata BLOB(1M)) RETURNS BLOB(1M)
  LANGUAGE JAVA
  EXTERNAL NAME 'sample3.jar:sample3.gunzip(byte[]) returns byte[]'
  PARAMETER STYLE JAVA
end_proc;
```

- **Executing the Java stored procedure**

```
INSERT INTO t1 values(10, ?, gzip(? AS BLOB(1M)))
      :
SELECT c1, c2, gunzip(c3), length(c2), length(c3) from t1
```

(4) Sample 4

This example uses a dynamic result set to return the result of retrieving two tables.

- **External Java procedure (filename: sample4rs.java)**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/*************************************************************************/
/* name = HiRDB 06-00 Java stored Result Set connection job              */
/*************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;

/*************************************************************************/
/* name = Result Set connection class (procedure side)                   */
/*************************************************************************/
public class sample4rs {
    /*===================================================================*/
    /* name = main method for debugging                                  */
    /*===================================================================*/
    public static void main(String args[]) throws SQLException {
        java.lang.Integer p1 = new Integer(10);
    int[]   cr_cnt = null;
    java.sql.ResultSet[] rs1 = null;
    java.sql.ResultSet[] rs2 = null;

    try {
        // Registering Driver class
        Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
    } catch (ClassNotFoundException ex) {
        System.out.println("\n***** ClassNotFoundException caught *****\n");
        ex.printStackTrace();
        System.out.println ("");
        System.out.println("********************************\n");
        return;
    }

    rs_proc(p1, cr_cnt, rs1, rs2);
    }

    /*==================================================================*/
    /* name = Result Set connection method                              */
    /*==================================================================*/
    public static void rs_proc(java.lang.Integer p1,int icnt_cr[],
                    java.sql.ResultSet[] rs1,
                    java.sql.ResultSet[] rs2) throws SQLException {
        java.lang.Integer x_goods_no;
        java.math.BigDecimal x_quantity_1, x_total_quantity;

        try {
            // Creating a connection object (CONNECT not issued within
                                             Java procedure)
            java.sql.Connection con =
                DriverManager.getConnection("jdbc:hitachi:hirdb");

            con.setAutoCommit(false);  // Suppressing automatic commit

            // SELECT (stmt1) preprocessing
            java.sql.PreparedStatement stmt1 =
                con.prepareStatement("SELECT c1, c2 FROM rs_t1 WHERE c1 > ?");

            // Specifying SELECT (stmt1) input parameter
            stmt1.setInt(1, p1.intValue());
```

```
       // SELECT (stmt2) preprocessing
       java.sql.PreparedStatement stmt2 =
           con.prepareStatement("SELECT c1, c2 FROM rs_t2 WHERE c1 > 10");

       // Executing SELECT (stmt1)
       rs1[0] = stmt1.executeQuery();

       // Executing SELECT (stmt2)
       rs2[0] = stmt2.executeQuery();

       // Number of dynamic result sets
       icnt_cr[0] = 2;

       // Executing SELECT (stmt2) (retrieving only one row)
       rs2[0].next();

    } catch (SQLException ex) { // SQL error handling procedure
       SQLException fast_ex = ex;

       System.out.println("\n***** SQLException caught *****\n");
       while (ex != null) {
           System.out.println ("SQLState: " + ex.getSQLState ());
           System.out.println ("Message:  " + ex.getMessage ());
           System.out.println ("Vendor:   " + ex.getErrorCode ());
           ex.printStackTrace();
           ex = ex.getNextException ();
           System.out.println ("");
       }
```

```
            System.out.println("******************************\n");
            throw fast_ex;
        }

        return;
    }
}
```

■ UAP (sample4ap.java)

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2000,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/*************************************************************************/
/* name = HiRDB 06-00 Java stored Result Set connection object           */
/*************************************************************************/
import java.lang.*;
import java.math.*;
import java.sql.*;

/*************************************************************************/
/* name = Result Set connection class (CALL side)                        */
/*************************************************************************/
public class sample4ap {
    /*===================================================================*/
    /* name = main method for debugging                                  */
    /*===================================================================*/
    public static void main(String args[]) throws SQLException {
        try {

        // Registering Driver class
                Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
        } catch (ClassNotFoundException ex) {
            System.out.println("\n***** ClassNotFoundException caught *****\n");
            ex.printStackTrace();
            System.out.println ("");
            System.out.println("******************************\n");
            return;
        }
        rs_call();
    }
    /*==================================================================*/
    /* name = Result Set connection method                              */
    /*==================================================================*/
    public static void rs_call() throws SQLException {
        java.lang.Integer xc1;
        java.lang.String xc2;
        int cr_cnt[] = new int[1];

        try {
            // Creating a connection object (CONNECT not issued within
                                             Java procedure)
            java.sql.Connection con =
               DriverManager.getConnection
                             ("jdbc:hitachi:hirdb", "\"USER1\""
                                                      , "\"PASS1\"");

            con.setAutoCommit(false);  // Suppressing automatic commit

            // CALL (stmt1) preprocessing
            java.sql.CallableStatement stmt1 =
                con.prepareCall("{CALL rs_proc(?,?)}");

            // Specifying CALL (stmt1) input parameters
            stmt1.setInt(1, 10);
            stmt1.registerOutParameter(2, java.sql.Types.INTEGER);

            // Executing CALL (stmt1)
            stmt1.execute();

            // Obtaining CALL (stmt1) output parameter
            cr_cnt[0] = stmt1.getInt(2);

            System.out.println("cr_cnt=" + cr_cnt[0] + "\n");

            // Obtaining dynamic result set
            java.sql.ResultSet rs = stmt1.getResultSet();

            while (rs.next()) {
                // Obtaining SELECT (stmt1) retrieval result
                xc1 = (Integer)rs.getObject("c1");
                xc2 = (String)rs.getObject("c2");
                System.out.println("xc1=" + xc1 + ",xc2=" + xc2 + "\n");
            }

            // Closing the cursor
            rs.close();

            if (stmt1.getMoreResults()) {
                rs = stmt1.getResultSet();
```

```
                while (rs.next()) {
                    // Obtaining SELECT (stmt1) retrieval result
                    xc1 = (Integer)rs.getObject("c1");
                    xc2 = (String)rs.getObject("c2");
                    System.out.println("xc1=" + xc1 + ",xc2=" + xc2 + "\n");
                }
            }

            // Closing the cursor
            rs.close();

            // Releasing each statement object
            stmt1.close();

            // Disconnecting
            con.close();

        } catch (SQLException ex) { // SQL error handling procedure
            SQLException fast_ex = ex;

            System.out.println("\n***** SQLException caught *****\n");
            while (ex != null) {
                System.out.println ("SQLState: " + ex.getSQLState ());
                System.out.println ("Message:  " + ex.getMessage ());
                System.out.println ("Vendor:   " + ex.getErrorCode ());
                ex.printStackTrace();
                ex = ex.getNextException ();
                System.out.println ("");
            }
            System.out.println("*******************************\n");


            throw fast_ex;
        }


        return;
    }
}
```

■ Defining the Java stored procedure

```
CREATE PROCEDURE rs_proc(IN p1 INT,OUT cr_cnt INT)
  DYNAMIC RESULT SETS 2
  LANGUAGE JAVA
  EXTERNAL NAME 'sample4.jar:sample4rs.rs_proc(java.lang.Integer, int[]
                                                                , java.sql.
ResultSet[], java.sql.ResultSet[]) returns void'
  PARAMETER STYLE JAVA
end_proc;
```

9.4 Notes about Java program creation

This section describes the points to be observed when creating a Java program. There are the following limitations to the specification of control processing in Java:

- No thread can be created.
- A GUI cannot be used.
- Connection cannot be established with another DBMS.
- File manipulation is not supported.
- Do not change the Java Runtime Environment security policy.

9.4.1 Using a Type 2 or 4 JDBC driver

If you run a Java stored procedure by using a Type 2 or 4 JDBC driver, it is important that you check the settings described in *18.12 Migration from a Type2 JDBC driver*. If a driver name, a protocol name that is set in the URL when HiRDB is connected, a sub-protocol name, and a sub-name are specified in a Java stored procedure, whether a Type2 JDBC driver or a Type4 JDBC driver is used is determined on the basis of specified information.

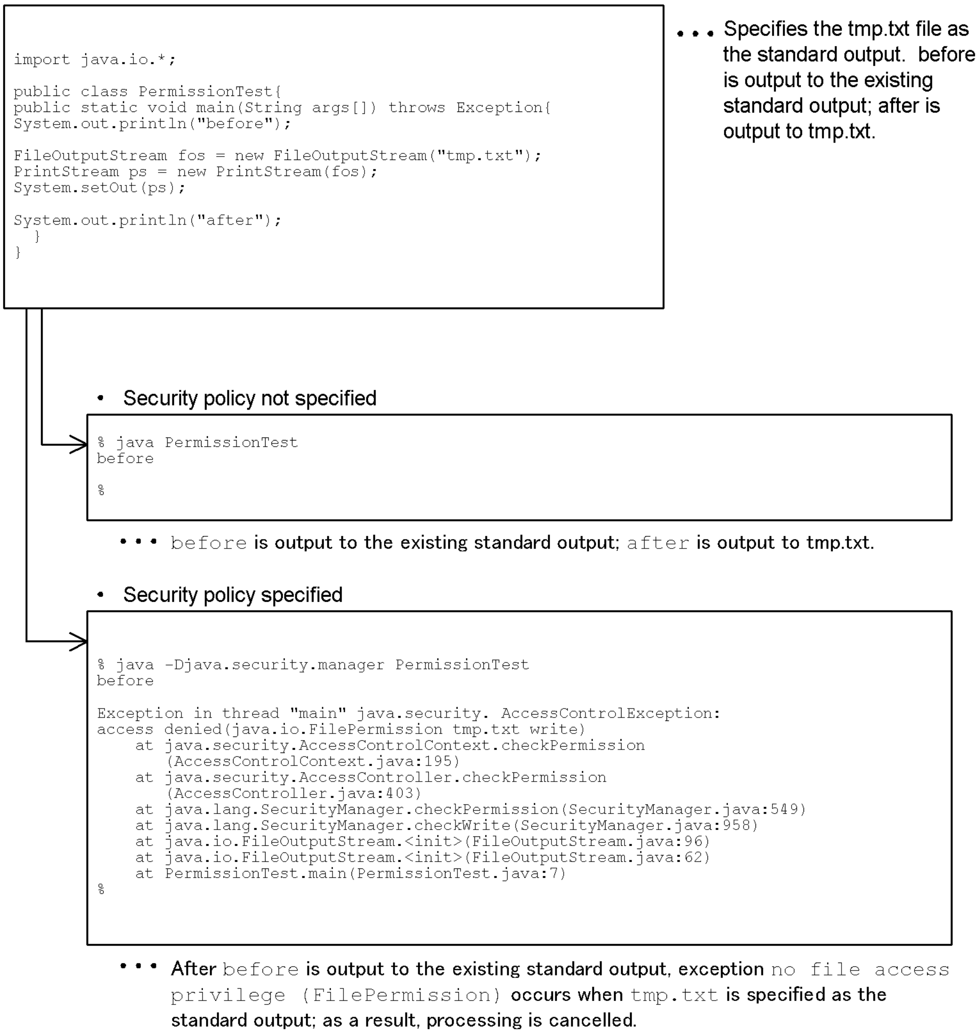
9.4.2 Unsupported methods

On a Java virtual machine, you can limit available methods by specifying the access privilege with the security policy. The Java virtual machine in HiRDB does not allow any method to be executed without the access privilege.

For the specification of the access privilege with the security policy and a list of unsupported methods, see the manual provided with JDK.

The following figure shows method execution control using the security policy.

Figure 9-9: Method execution control using security policy



```
import java.io.*;

public class PermissionTest{
public static void main(String args[]) throws Exception{
System.out.println("before");

FileOutputStream fos = new FileOutputStream("tmp.txt");
PrintStream ps = new PrintStream(fos);
System.setOut(ps);

System.out.println("after");
  }
}
```

• • • Specifies the tmp.txt file as the standard output. before is output to the existing standard output; after is output to tmp.txt.

- Security policy not specified

```
% java PermissionTest
before

%
```

• • • `before` is output to the existing standard output; `after` is output to tmp.txt.

- Security policy specified

```
% java -Djava.security.manager PermissionTest
before

Exception in thread "main" java.security. AccessControlException:
access denied(java.io.FilePermission tmp.txt write)
    at java.security.AccessControlContext.checkPermission
       (AccessControlContext.java:195)
    at java.security.AccessController.checkPermission
       (AccessController.java:403)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
    at java.lang.SecurityManager.checkWrite(SecurityManager.java:958)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:96)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:62)
    at PermissionTest.main(PermissionTest.java:7)
%
```

• • • After `before` is output to the existing standard output, exception `no file access privilege (FilePermission)` occurs when `tmp.txt` is specified as the standard output; as a result, processing is cancelled.

9.4.3 Package, class, and method definitions

This section describes the points to be observed when defining packages, classes, and methods. For details about the packages, classes, and methods, see the manual provided with JDK.

(1) Package

1. Specification of a package name is optional.

2. If you specify the package name, the length of the *package-name.class-name* character string must be no longer than 255 characters.
3. You cannot use either of the following package names:
    - Package name existing in JRE
    - Package name provided by HiRDB

(2) Class

1. A class name must be no longer than 255 characters.
2. Define a class in the format `public class` <*class-name*>.

(3) Method

1. A method name must be no longer than 255 characters.
2. Define a method as follows:

    Java stored procedure
    `public static void` <*method-name*>

    Java stored function
    `public static` <*return-value*> <*method-name*>
3. If there is a possibility of an exception occurring in the method, you must either declare the exception in the `throw` section or specify `try.catch`. For Java stored procedures, there is a possibility of an `SQLException` exception occurring in nearly all JDBC methods.
4. A method can reference a class included in the Java platform core API or a class included in the JAR file that contains the current method.

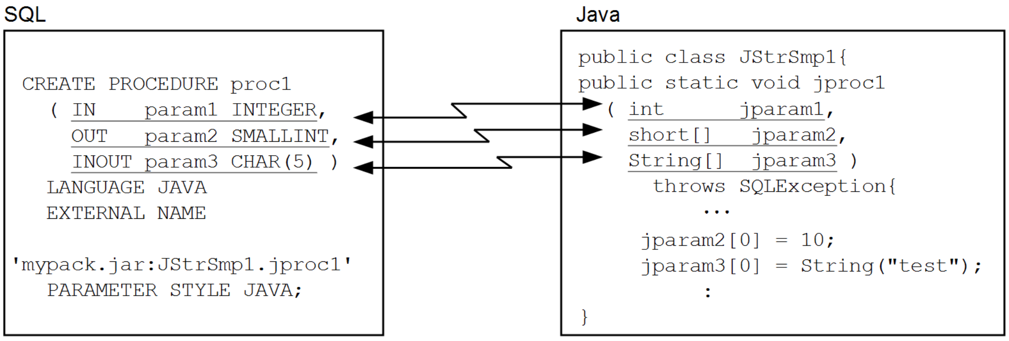
9.4.4 Parameter input/output mode mapping (Java stored procedures only)

This section explains the mapping of the SQL parameter input/output mode (`IN`, `OUT`, or `INOUT`) with Java stored procedures. You cannot specify a parameter input/output mode for a Java stored function.

For details about mapping, see the type mapping in the *HiRDB Version 9 SQL Reference* manual.

The following figure shows an example of parameter input/output mode mapping.

Figure 9-10: Example of parameter input/output mod mapping



SQL

```
CREATE PROCEDURE proc1
  ( IN    param1 INTEGER,
    OUT   param2 SMALLINT,
    INOUT param3 CHAR(5) )
  LANGUAGE JAVA
  EXTERNAL NAME
'mypack.jar:JStrSmp1.jproc1'
  PARAMETER STYLE JAVA;
```

Java

```
public class JStrSmp1{
public static void jproc1
  ( int      jparam1,
    short[]  jparam2,
    String[]  jparam3 )
      throws SQLException{
            ...
     jparam2[0] = 10;
     jparam3[0] = String("test");
           :
}
```

(1) IN parameter

For a parameter defined as an `IN` parameter with SQL, a Java program uses the corresponding data type as is.

Suppose that the `IN` parameter is defined as an `INTEGER` type with the `CREATE PROCEDURE` SQL statement. With a Java program, it is defined as the corresponding `int` type or `java.lang.Integer` type (`param1` and `jparam1` in *Figure 9-10*).

(2) OUT or INPUT parameter

For a parameter defined as an `OUT` or `INOUT` parameter with SQL, a Java program defines it as the array type of the corresponding data type. The `OUT` and `INOUT` parameters are implemented in this manner because a parameter is to be passed as a one-element array of the corresponding data type with the pointer representation method in the Java language.

Suppose that the `OUT` parameter is defined as `SMALLINT` type with the `CREATE PROCEDURE` SQL statement. With a Java program, it is defined as the array type of the corresponding `short` or `java.long.Short` type (`param2` and `jparam2`; `param3` and `jparam3` in *Figure 9-10*). To return a value to the `OUT` or `INOUT` parameter, set the value at the beginning of the array (`jparam2` and `jparam3` in *Figure 9-10*).

9.4.5 Results-set return facility (Java stored procedures only)

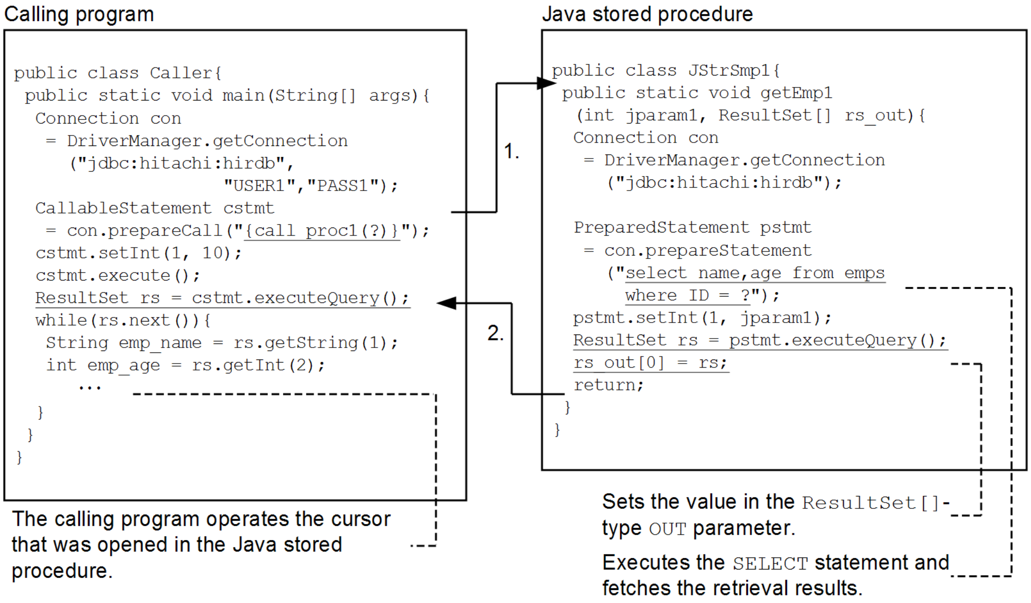
You can use the results-set return facility by specifying 1 or a greater value in the `DYNAMIC RESULT SETS` clause in `CREATE PROCEDURE` during Java stored procedure definition. The results-set return facility is not supported for Java stored functions.

(1) What is the results-set return facility?

The results-set return facility enables the Java stored procedure caller to reference the cursor that is acquired by the execution of the `SELECT` statement within the Java stored procedure.

The following figure provides an overview of the results-set return facility.

Figure 9-11: Overview of the results-set return facility (for a Java stored procedure)



Calling program

```
public class Caller{
 public static void main(String[] args){
  Connection con
   = DriverManager.getConnection
     ("jdbc:hitachi:hirdb",
                    "USER1","PASS1");
  CallableStatement cstmt
   = con.prepareCall("{call proc1(?)}");
  cstmt.setInt(1, 10);
  cstmt.execute();
  ResultSet rs = cstmt.executeQuery();
  while(rs.next()){
   String emp_name = rs.getString(1);
   int emp_age = rs.getInt(2);
      ...
  }
 }
}
```

Java stored procedure

```
public class JStrSmp1{
 public static void getEmp1
   (int jparam1, ResultSet[] rs_out){
  Connection con
   = DriverManager.getConnection
     ("jdbc:hitachi:hirdb");

  PreparedStatement pstmt
   = con.prepareStatement
     ("select name,age from emps
      where ID = ?");
  pstmt.setInt(1, jparam1);
  ResultSet rs = pstmt.executeQuery();
  rs_out[0] = rs;
  return;
 }
}
```

The calling program operates the cursor that was opened in the Java stored procedure.

Sets the value in the `ResultSet[]`-type `OUT` parameter.

Executes the `SELECT` statement and fetches the retrieval results.

Explanation:
1. Calls the method that was associated by `CREATE PROCEDURE`.
2. Returns the result set.

(2) Calling-source languages supporting the results-set return facility

The calling-source languages that support the results-set return facility are as follows:

- Java
- C
- C++
- COBOL#
- OOCOBOL

#: COBOL can be used if the RDB file input/output facility is not used.

(3) Example of using the results-set return facility

This example obtains columns `rank`, `name`, and `age`, which satisfy condition `rank<10` in tables `emps_1` and `emps_2` within a Java stored procedure. The caller receives two result sets to manipulate them.

■ Defining the Java stored procedure

```
CREATE PROCEDURE proc2(IN param1 INTEGER)   ...........1
  DYNAMIC RESULT SETS 2   .............................2
  LANGUAGE JAVA   .....................................3
  EXTERNAL NAME   .....................................4
    'mypack.jar:JStrSmp1.getEmp2(int, ResultSet[]
                                    , ResultSet[])'  ..4
  PARAMETER STYLE JAVA;   .............................5
```

Explanation

1. Defines the procedure name and parameters.
2. Specifies the number of retrieval result sets to be returned.
3. Specifies LANGUAGE.
4. Associates with the Java method.
5. Specifies PARAMETER STYLE.

- **Java stored procedure**

```
import java.sql.*;   ................................1


public class JStrSmp1{   ............................2
  public static void getEmp2   ......................3
    (int jparam1, ResultSet[] rs1_out
                , ResultSet[] rs2_out)   ............4
          throws SQLException {   ...................4
    Connection con = DriverManager.getConnection (  ...5
                     "jdbc:hitachi:hirdb");   ........5
    con.setAutoCommit(false);   .....................6


    PreparedStatement pstmt1 = con.prepareStatement  ..7
        ("select rank,name,age from emps_1 where
                      rank < ?   .....................7
         order by rank");   .........................7
    pstmt1.setInt(1, jparam1);   .....................7
    ResultSet rs1 = pstmt1.executeQuery();   .........8
```

```
     rs1_out[0] = rs1;    ..............................9
     PreparedStatement pstmt2 = con.prepareStatement  ..10
          ("select rank,name,age from emps_2 where
                          rank < ?   .....................10
           order by rank");   ..........................10
     pstmt2.setInt(1, jparam1);   .....................10
     ResultSet rs2 = pstmt2.executeQuery();   ....... ..11
     rs2_out[0] = rs2;   ......................... ...12
     return;   ................................ ......13
   }
}
```

Explanation

1. Imports the `java.sql` package.
2. Defines the class name.
3. Defines the method name.
4. Defines the parameter name (the second and third arguments are for returning result sets).
5. Obtains the `Connection` object.
6. Suppresses automatic commit processing.
7. Preprocesses the `SELECT` statement.
8. Executes the `SELECT` statement.
9. Sets the obtained result set `rs1` in the second argument of the `ResultSet[]` type.
10. Preprocesses the `SELECT` statement.
11. Executes the `SELECT` statement.
12. Sets the obtained result set `rs1` in the third argument of the `ResultSet[]` type.
13. Terminates the call and returns the result sets.

- **Executing the Java stored procedure (caller)**

```
import java.sql.*;  ......................................................1


public class Caller{  ..................................................2
  public static void main(String[] args)
                        throws SQLException {  .........................3
    Connection con = DriverManager.getConnection(  .......................4
                     "jdbc:hitachi:hirdb","USER1","PASS1");  ............4
                     ,"PASS1");  .......................................4
    CallableStatement cstmt = con.prepareCall
                     ("{call proc2(?)}");  ..............................5
    cstmt.setInt(1, 10);  ...............................................5

    ResultSet rs;  .....................................................6
    int emp_rank;  .....................................................6
    String emp_name;  ..................................................6
    int emp_age;  ......................................................6

    if(cstmt.execute()){  ..............................................7
       rs = cstmt.getResultSet();  .....................................8

       System.out.println("*** emps_1 ***");  ..........................9
       while(rs.next()){  ..............................................9
         emp_rank = rs.getInt(1);  .....................................9
         emp_name = rs.getString(2);  ..................................9
         emp_age = rs.getInt(3);  ......................................9
         System.out.println("RANK =" + emp_rank +  .......................9
           "  NAME = " + emp_name + "  AGE =
                               " + emp_age);  ..........................9
```

```
          }
     }


     if(cstmt.getMoreResults()){   ...................................10
        rs= cstmt.getResultSet();   ..................................11
        System.out.println("*** emps_2 ***");   ......................12
        while(rs.next()){   ..........................................12
          emp_rank = rs.getInt(1);   .................................12
          emp_name = rs.getString(2);   ..............................12
          emp_age = rs.getInt(3);   ..................................12

          System.out.println("RANK =" + emp_rank +   .................12
            "  NAME = " + emp_name + "  AGE =
            " + emp_age);   ..........................................12
        }
        rs.close();   ................................................13
     }
  }
}
```

Explanation

1. Imports the `java.sql` package.
2. Defines the class name.
3. Defines the method name.
4. Obtains the `Connection` object.
5. Preprocesses the `CALL` statement.
6. Declares variables.
7. Executes the `CALL` statement.
8. Obtains the result set.

9. Outputs information obtained from the first result set.
10. Checks to see if there are any more result sets.
11. Obtains the next result set.
12. Outputs information obtained from the second result set.
13. Closes the result sets.

(4) Notes about using the results-set return facility

(a) When defining a Java stored procedure with CREATE PROCEDURE

1. In the `DYNAMIC RESULT SETS` clause, specify the maximum number of result sets to be returned from within the Java stored procedure. If a value of 0 is specified, the system does not use the results-set return facility.
2. For a parameter in `CREATE PROCEDURE`, do not specify the `OUT` parameter of `ResultSet[]` type that is specified for a parameter in the Java stored procedure.
3. When using `EXTERNAL NAME` to define correspondence with a Java program, include arguments of the `ResultSet[]` type.

(b) When creating a caller's method

1. In the `CALL` statement's parameters, do not include a parameter of the `ResultSet[]` type in the method for Java stored procedure.
2. If there is more than one result set to be returned, to receive the second or subsequent result set, use the `getMoreResult` method (to determine whether there are more result sets) and the `getResultSet` method (to receive the next result).

(c) When creating a method for a Java stored procedure

Specify the retrieval result (`ResultSet`) in the `OUT` parameter of the `ResultSet[]` type without closing it.

9.4.6 Connection in a Java stored procedure

You can create only one active connection within a Java stored procedure. Database manipulation using a connection object is not available if the database and JDBC resources are released by the garbage collector, or the `close()` method is used to explicitly release the database and JDBC resources before the Java stored procedure is terminated.

9.4.7 Releasing the result sets

To release a result set object, use the `close()` method. Implicit release using the garbage collector does not release the resources until the Java stored procedure is terminated.

9.5 Notes about testing and debugging

Because external Java stored routines are based on the architecture of running a normal Java program on a DBMS server, the methods of testing and debugging them are the same as for Java applications.

After creating a Java program, you need to test it and debug it so that the Java program runs successfully as a stored procedure or stored function. This section describes the points to be observed when testing and debugging a Java program.

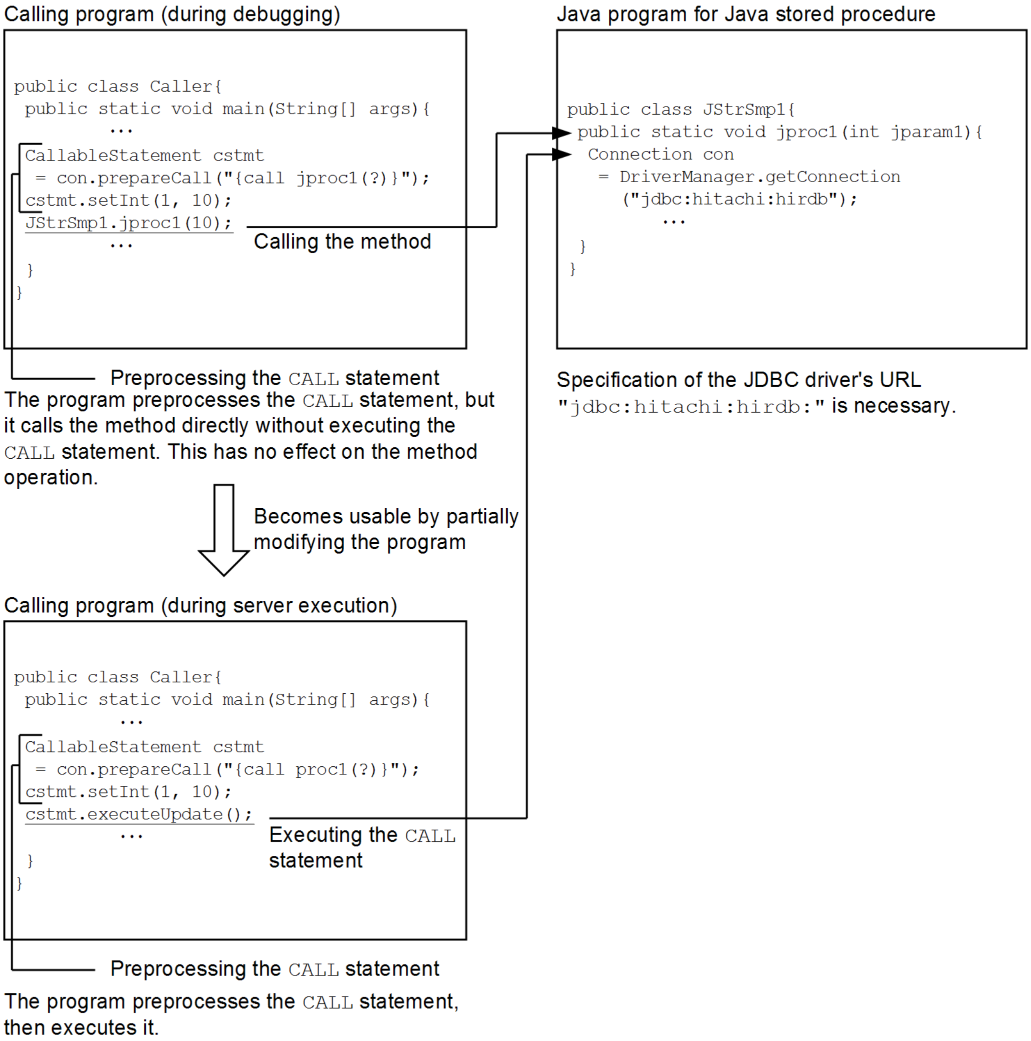
9.5.1 Java program for a Java stored procedure

When testing or debugging a Java program for a Java stored procedure, note the following:

1. You can use a Java program for a Java stored procedure during server execution without having to modify the debugged program.
2. A caller Java program directly calls the method of the Java program for a Java stored procedure during debugging. During server execution, it uses the `CALL` statement.
3. Available methods may be different because the Java virtual machine environment is different between debugging and server execution. For details about the methods that cannot be executed, see *9.4.2 Unsupported methods*.

The following figure shows the procedure for testing and debugging a Java program for a Java stored procedure.

Figure 9-12: Procedure for testing and debugging a Java program for a Java stored procedure



Calling program (during debugging)

```
public class Caller{
 public static void main(String[] args){
         ...
 CallableStatement cstmt
  = con.prepareCall("{call jproc1(?)}");
 cstmt.setInt(1, 10);
 JStrSmp1.jproc1(10);
         ...
 }
}
```

Calling the method

Preprocessing the CALL statement

The program preprocesses the CALL statement, but it calls the method directly without executing the CALL statement. This has no effect on the method operation.

Becomes usable by partially modifying the program

Java program for Java stored procedure

```
public class JStrSmp1{
 public static void jproc1(int jparam1){
  Connection con
   = DriverManager.getConnection
     ("jdbc:hitachi:hirdb");
          ...
 }
}
```

Specification of the JDBC driver's URL "jdbc:hitachi:hirdb:" is necessary.

Calling program (during server execution)

```
public class Caller{
 public static void main(String[] args){
         ...
 CallableStatement cstmt
  = con.prepareCall("{call proc1(?)}");
 cstmt.setInt(1, 10);
 cstmt.executeUpdate();
         ...
 }
}
```

Executing the CALL statement

Preprocessing the CALL statement

The program preprocesses the CALL statement, then executes it.

9.5.2 Java program for a Java stored function

When testing or debugging a Java program for a Java stored function, note the following:

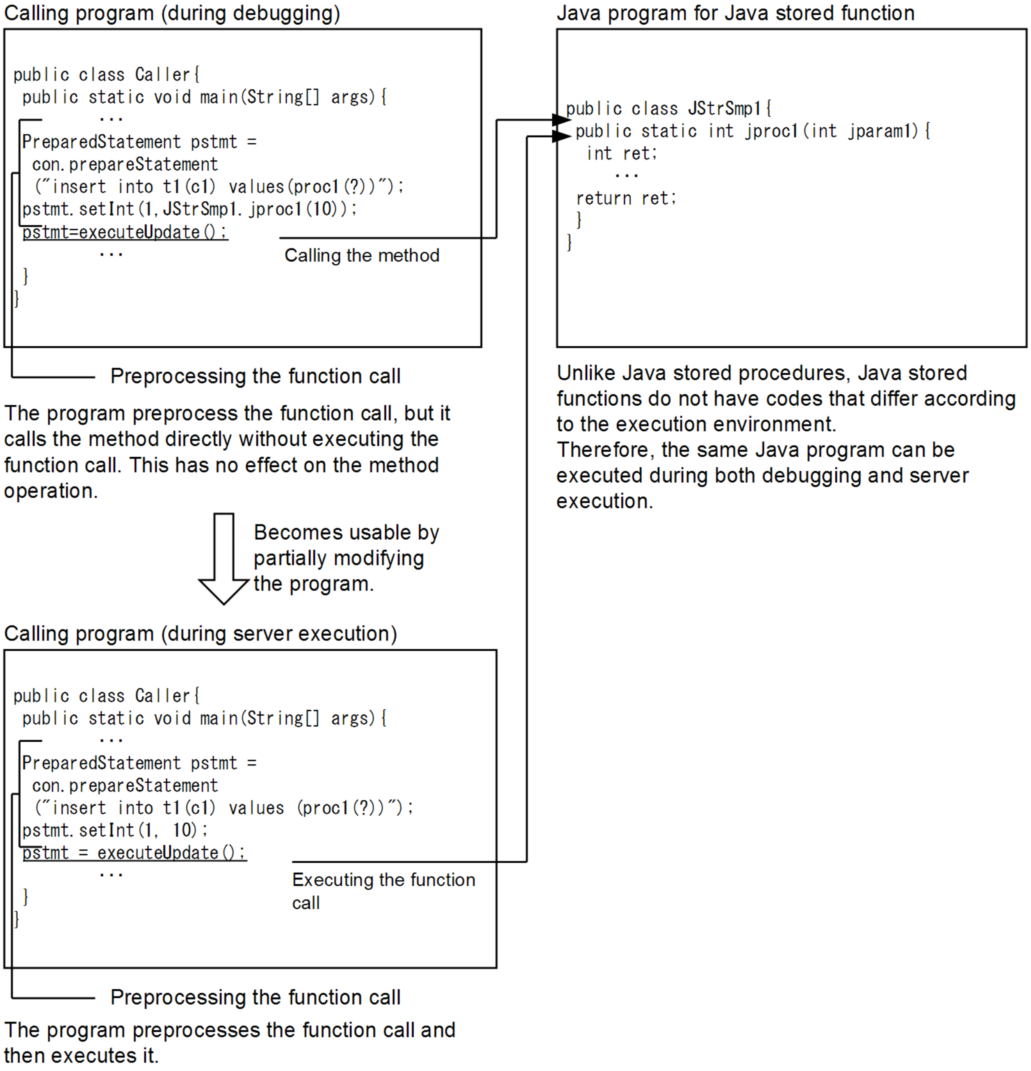
1. You can use a Java program for a Java stored function during server execution without having to modify the debugged program.
2. A caller Java program directly calls the method of the Java program for a Java

stored function during debugging. During server execution, it uses a function call.

3. Available methods may be different because the Java virtual machine environment is different between debugging and server execution. For details about the methods that cannot be executed, see *9.4.2 Unsupported methods*.

The following figure shows the procedure for testing and debugging a Java program for a Java stored function.

Figure 9-13: Procedure for testing and debugging a Java program for a Java stored function



Calling program (during debugging)

```
public class Caller{
 public static void main(String[] args){
          ...
 PreparedStatement pstmt =
  con.prepareStatement
  ("insert into t1(c1) values(proc1(?))");
 pstmt.setInt(1,JStrSmp1.jproc1(10));
 pstmt=executeUpdate();
          ...
 }
}
```

The program preprocess the function call, but it calls the method directly without executing the function call. This has no effect on the method operation.

Becomes usable by partially modifying the program.

Calling program (during server execution)

```
public class Caller{
 public static void main(String[] args){
          ...
 PreparedStatement pstmt =
  con.prepareStatement
  ("insert into t1(c1) values (proc1(?))");
 pstmt.setInt(1, 10);
 pstmt = executeUpdate();
          ...
 }
}
```

The program preprocesses the function call and then executes it.

Java program for Java stored function

```
public class JStrSmp1{
 public static int jproc1(int jparam1){
  int ret;
      ...
 return ret;
 }
}
```

Unlike Java stored procedures, Java stored functions do not have codes that differ according to the execution environment.
Therefore, the same Java program can be executed during both debugging and server execution.

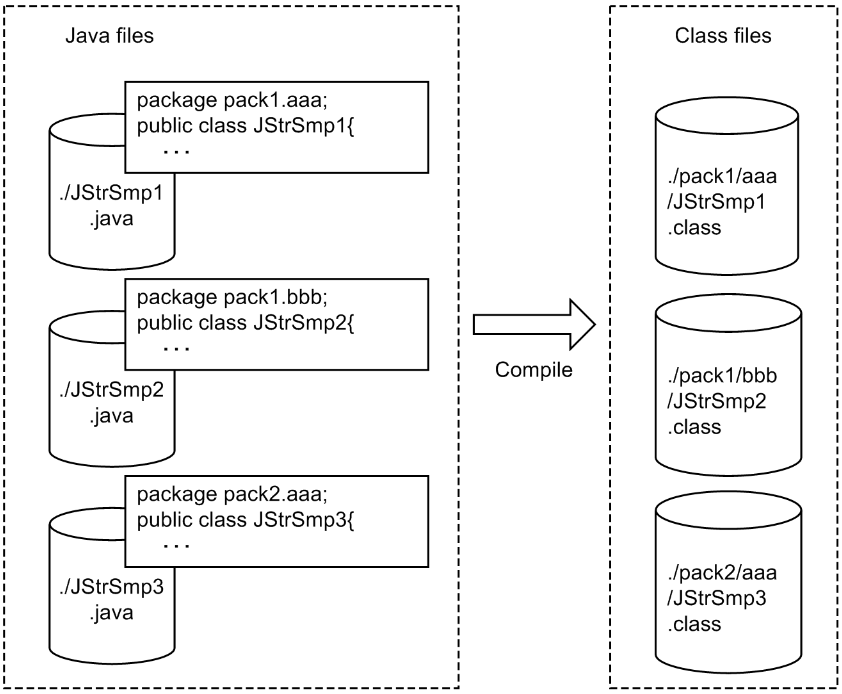
9.6 Notes about JAR file creation

This section describes the points to be observed when creating a JAR file.

Java has a concept called *package* to classify and manage programs by their function. A package is actually expressed as a directory structure; therefore, a Class file is created in the directory with the package name after compilation.

The following figure shows the location at which Class files are created.

Figure 9-14: Location at which Class files are created



When creating Java files, you can integrate and compress the files, including the directory structure.

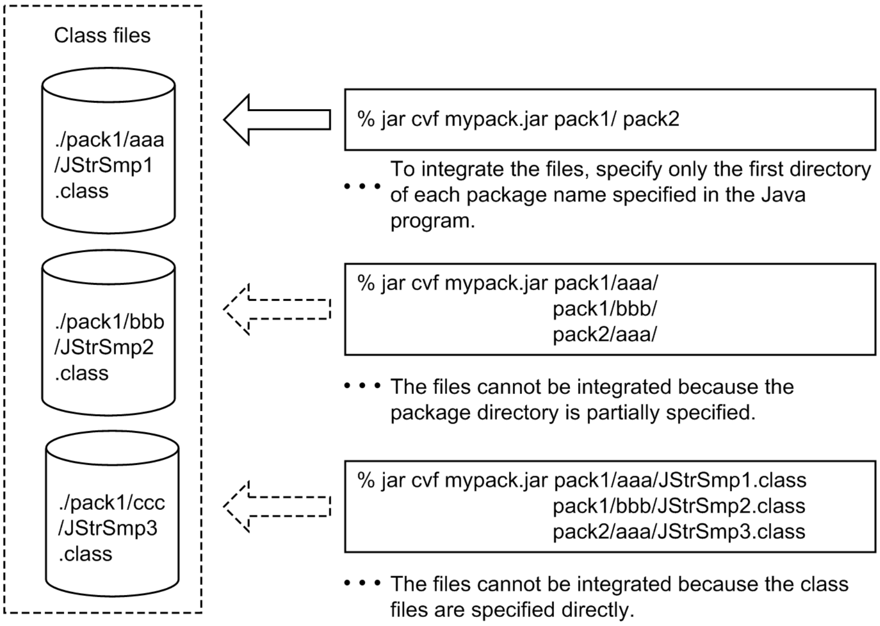
You can integrate not only Class files but also Java files at the same time. When obtaining the Java program source with a specified Class from the JAR files registered in HiRDB by conducting a retrieval specifying `GET_JAVA_STORED_ROUTINE_SOURCE`, you need to integrate the Java files at the same time. For details about the `GET_JAVA_STORED_ROUTINE_SOURCE` specification, see the *HiRDB Version 9 SQL Reference* manual.

9.6.1 Integrating Class files

If a package is specified during Java program creation, integration in the JAR file takes place for each package directory. Do not specify individual Class files only.

The following figure shows an example of integrating Class files in JAR files.

Figure 9-15: Example of integrating Class files in JAR files



```
% jar cvf mypack.jar pack1/ pack2
```

• • • To integrate the files, specify only the first directory of each package name specified in the Java program.

```
% jar cvf mypack.jar pack1/aaa/
                     pack1/bbb/
                     pack2/aaa/
```

• • • The files cannot be integrated because the package directory is partially specified.

```
% jar cvf mypack.jar pack1/aaa/JStrSmp1.class
                     pack1/bbb/JStrSmp2.class
                     pack2/aaa/JStrSmp3.class
```

• • • The files cannot be integrated because the class files are specified directly.

The Class files with the same name can be integrated in the same JAR file if their packages are different.

9.6.2 Integrating Java files

The following describes the points to be observed when integrating Java files:

1. To retrieve a Java program source corresponding to a Class file, the Java file must be integrated at the same time as the Class file.
2. A Java file to be integrated in a JAR file can be stored in any directory.
3. If multiple packages have the same Class filename, you can retrieve each Java program source by storing the corresponding Java file in a different directory. The following shows an example:

   Example

   If the Class files consist of the following packages, `pack1.aaa.JStrAAA` and `pack2.ccc.JStrAAA,` they will have the same Class filename:

```
./pack1/aaa/JstrAAA.class

./pack1/bbb/JstrBBB.class

./pack2/ccc/JstrAAA.class
```

There is no need to manage the Java files with the same directory structure, but if there are multiple files with the same name, they cannot be stored under the same directory. In this case, individual files can be stored as follows:

```
./src1/JStrAAA.java

./src1/JStrBBB.java

./src2/JStrAAA.java
```

This example cannot identify each Java file corresponding to a specified Class file; therefore, all Java files with the specified name are retrieved. For example, a retrieval of `JStrAAA.java` results in both `pack1.aaa.JStrAAA.java` and `pack2.ccc.JStrAAA.java`. A retrieval of `JStrBBB.java` results in `pack1.bbb.JStrBBB.java`.

Chapter

10. C Stored Procedures and C Stored Functions

This chapter explains the procedures for creating and executing C stored procedures and C stored functions that code procedures in C language.

10.1 Overview
10.2 Procedure from external C stored routine creation to execution
10.3 Sample external C stored routine programs
10.4 Limitations to C program creation

10.1 Overview

Stored procedures and stored functions coded in C language are called *C stored procedures* and *C stored functions*, respectively. They are referred to collectively in this chapter as *external C stored routines*.

Features of external C stored routines

- **There is no overhead for communication between the server and a client.**

  Because external C stored routines are processed at the server in the same manner as SQL stored procedures and SQL stored functions, there is no communication overhead between server and client.

- **The procedure or function itself can be coded in C language.**

  Because C instructions can be specified directly, processing can be coded more flexibly than with SQL control statements.

- **Debugging is easy.**

  To debug an SQL stored procedure or an SQL stored function, you need to execute it at the server. In contrast, an external C stored routine can be debugged at the client, as long as the C debugger is installed at the client.

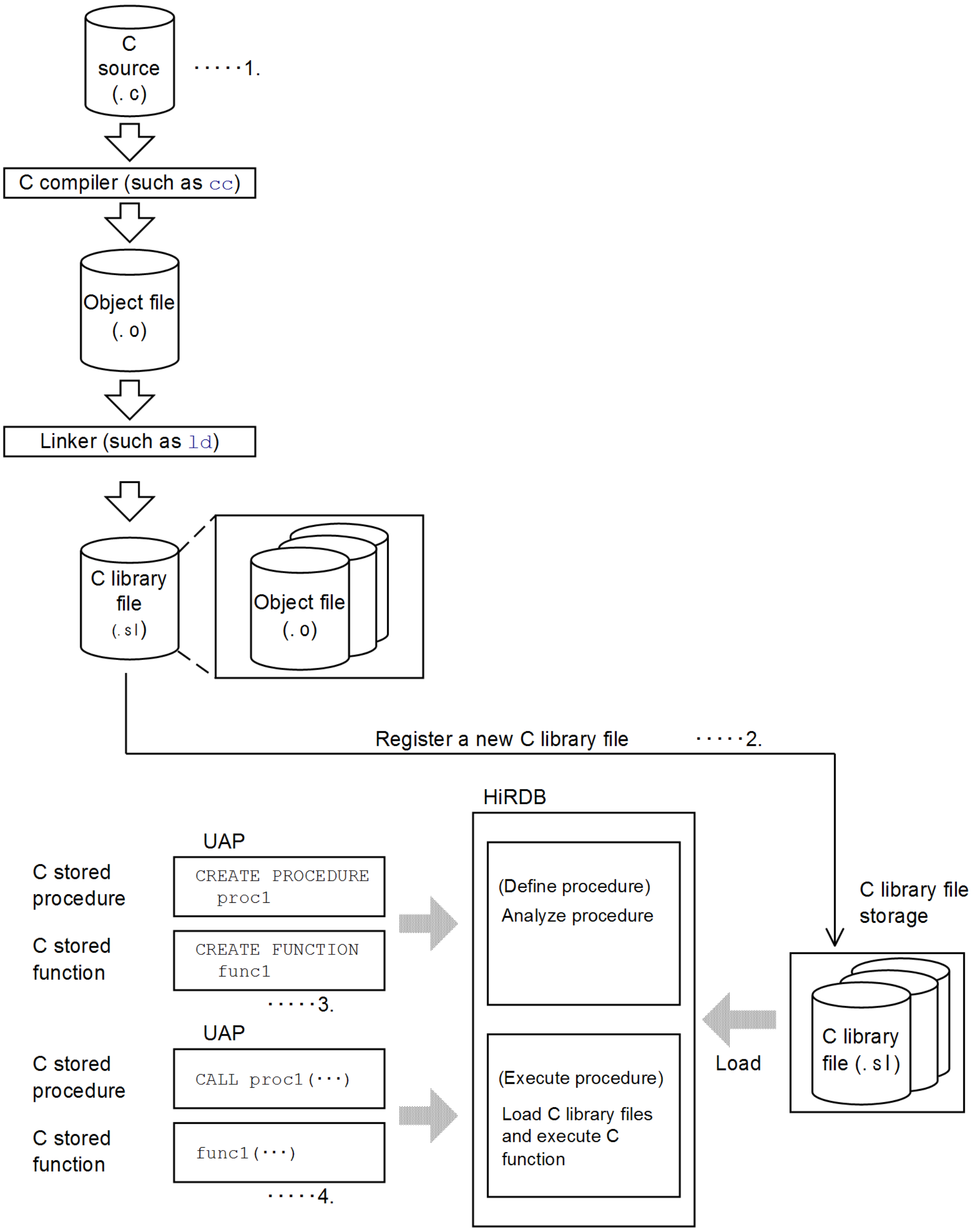
10.2 Procedure from external C stored routine creation to execution

To create and execute an external C stored routine:

1. Create an external C stored routine.
2. Register the C library file.
3. Define the external C stored routine.
4. Execute the external C stored routine.

The following figure shows the procedure from external C stored routine creation to execution.

Figure 10-1: Procedure from external C stored routine creation to execution



Note

The C library file extension depends on the OS.

Explanation

1. Creation of an external C stored routine. For details, see *10.2.1 Creating an external C stored routine*.
2. Registration of the C library file into the HiRDB system. For details, see *10.2.2 Registering the C library file*.
3. Definition of the external C stored routine. For details, see *10.2.3 Defining the external C stored routine*.
4. Execution of the external C stored routine. For details, see *10.2.4 Executing the external C stored routine*.

10.2.1 Creating an external C stored routine

To create an external C stored routine:

1. Code a C program (create a C file).
2. Compile (create an object file).
3. Link (create a C library file).

(1) Coding a C program (creating a C file)

You code a C program that is to be registered as an external C stored routine.

(a) C program creation rules

This subsection discusses the rules for creating C programs. Also see *10.4 Limitations to C program creation*.

- Specify the C language's default call convention (`__cdecl`).
- For the case where the program terminates normally, set `00000` in the output parameter for `SQLSTATE`.

  Example: `memcpy(sqlstate, "00000", 5);`
- For the case where the program terminates abnormally, set `38`*XYY* in the output parameter for `SQLSTATE`, where the values of *X* and *Y* must be in the following ranges:

  X: `I` to `Z`

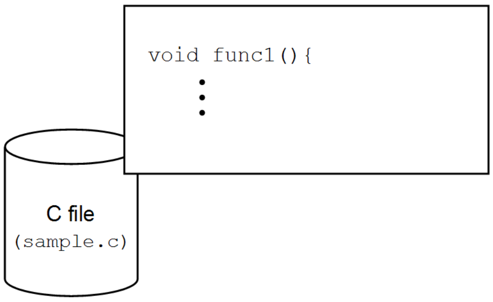
  Y: `0` to `9` or `A` to `Z`

  Example 1: `memcpy(sqlstate, "38I01", 5);`

  Example 2: `memcpy(sqlstate, "38ZCB", 5);`
- For the case where the program terminates abnormally, set a message text that indicates the cause of the error.

The following figure shows an example of C program coding.

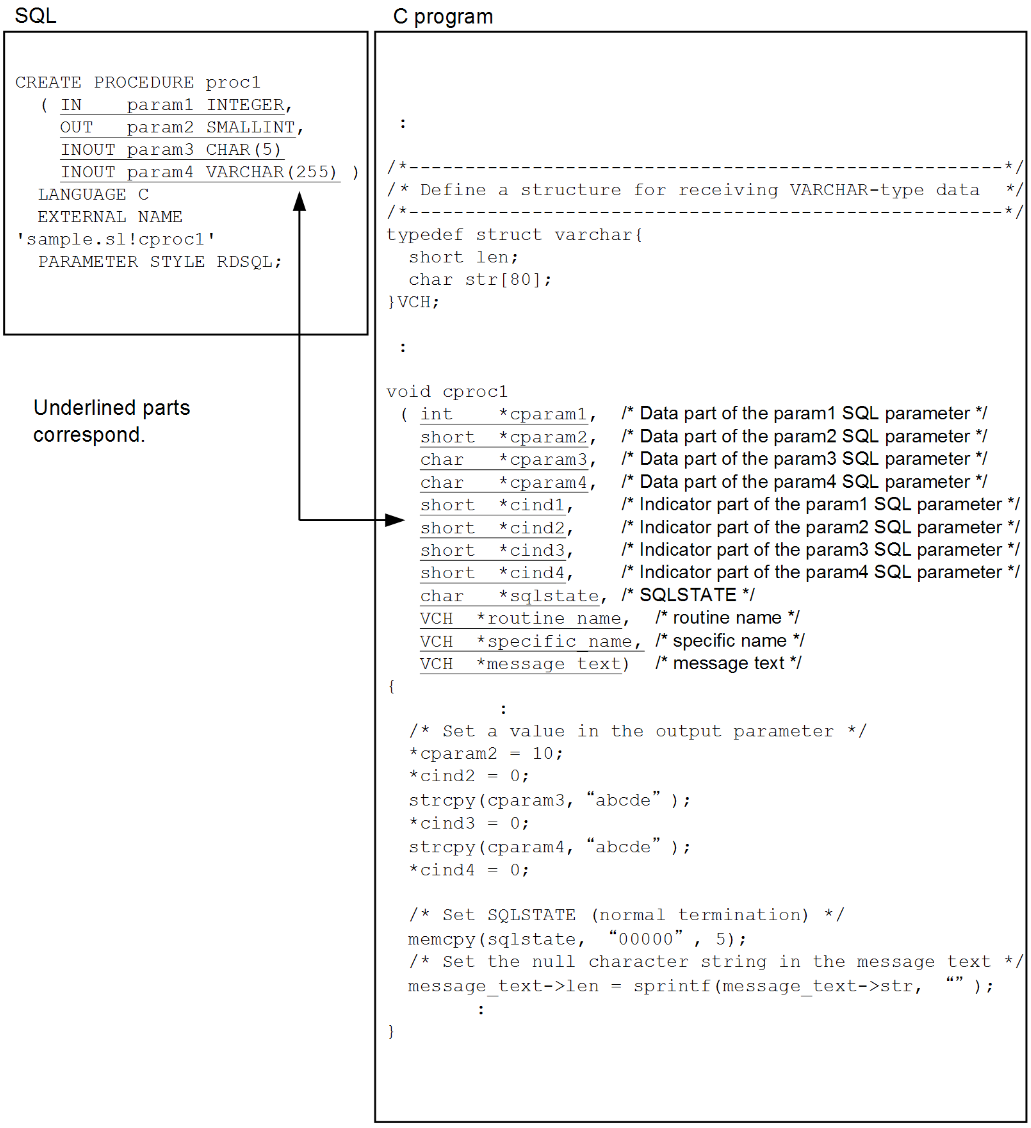
Figure 10-2: Example C program coding



(b) Mapping of parameter input/output modes

The following figure shows a sample mapping of SQL parameter input/output modes (`IN`, `OUT`, and `INOUT`) in an external C stored routine. For details about mapping, see type mapping in the manual *HiRDB Version 9 SQL Reference*.

Figure 10-3: Example of mapping of parameter input/output modes (external C stored routine)



SQL

```
CREATE PROCEDURE proc1
  ( IN     param1 INTEGER,
    OUT    param2 SMALLINT,
    INOUT param3 CHAR(5)
    INOUT param4 VARCHAR(255) )
  LANGUAGE C
  EXTERNAL NAME
'sample.sl!cproc1'
  PARAMETER STYLE RDSQL;
```

Underlined parts correspond.

C program

```
 :

/*-----------------------------------------------------*/
/* Define a structure for receiving VARCHAR-type data  */
/*-----------------------------------------------------*/
typedef struct varchar{
  short len;
  char str[80];
}VCH;

 :

void cproc1
 ( int     *cparam1,   /* Data part of the param1 SQL parameter */
   short   *cparam2,   /* Data part of the param2 SQL parameter */
   char    *cparam3,   /* Data part of the param3 SQL parameter */
   char    *cparam4,   /* Data part of the param4 SQL parameter */
   short   *cind1,     /* Indicator part of the param1 SQL parameter */
   short   *cind2,     /* Indicator part of the param2 SQL parameter */
   short   *cind3,     /* Indicator part of the param3 SQL parameter */
   short   *cind4,     /* Indicator part of the param4 SQL parameter */
   char    *sqlstate,  /* SQLSTATE */
   VCH  *routine_name,    /* routine name */
   VCH  *specific_name,   /* specific name */
   VCH  *message_text)    /* message text */
{
          :
  /* Set a value in the output parameter */
  *cparam2 = 10;
  *cind2 = 0;
  strcpy(cparam3, “abcde” );
  *cind3 = 0;
  strcpy(cparam4, “abcde” );
  *cind4 = 0;

  /* Set SQLSTATE (normal termination) */
  memcpy(sqlstate,  “00000” , 5);
  /* Set the null character string in the message text */
  message_text->len = sprintf(message_text->str,  “” );
        :
}
```

(2) Compiling (creating an object file)

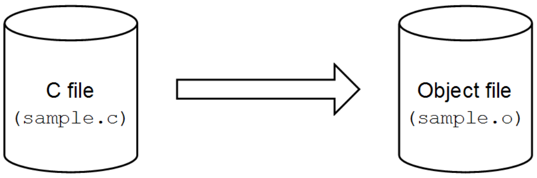
You use a method such as the `cc` command to create an object file from the C file.

The figure below shows an example of compilation. For details about the compiler options, see the OS documentation.

Figure 10-4: Example of compilation (external C stored routine)

Entered command

```
% cc +z -c sample.c
```



Note

For details about the compiler options, see the applicable OS's documentation.

(3) Linking (creating a C library file)

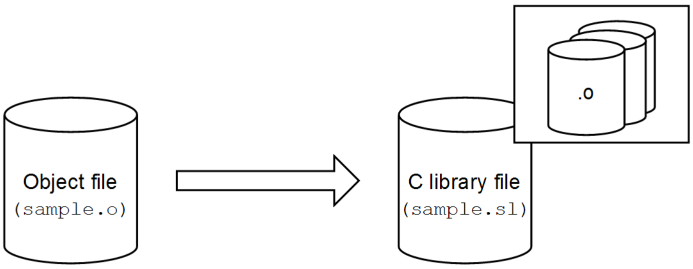
You use a method such as the `ld` command to create a C library file from multiple object files. If you use library functions, link the required libraries by using, for example, the `-l` option.

The following figure shows an example of linking.

Figure 10-5: Example of linking

Entered command

```
% ld -b -o sample.sl sample.o
```



Note 1

The C library file extension depends on the OS.

Note 2

For details about the linker options, see the documentation for the applicable OS.

(4) Examples of C library file creation for different OSs

This subsection presents examples of C library file creation from a C file (sample.c) for the different OSs. Creation of C library files varies according to the OS. For details, see the applicable OS's documentation.

The examples presented here assume that the compiler and linker are on the specified path.

(a) In HP-UX

Create a C library file named sample.sl from sample.c:

1. Create an object file by using the cc command with the +z option specified.

   ```
   $ cc -c +z sample.c
   ```

2. Create a C library file by using the ld command with the -b option specified.

   ```
   $ ld -b -o sample.sl sample.o
   ```

In 64-bit mode:

If you use the C library file in a 64-bit-mode HiRDB system, compile the file in 64-bit mode. In this case, specify the +DD64 compiler option.

In the POSIX library edition:

If you use the C library file in a 64-bit-mode HiRDB system, make sure that the following conditions are satisfied, because the file must support multiple threads:

- Specify the following options during compilation:

  ```
  -D_REENTRANT -D_POSIX_C_SOURCE=199506L
  -D_THREAD_SAFE -D_HPUX
  ```

- Use thread-safe functions.

(b) In Solaris

Create a C library file by using the cc command with the -G option specified. The following example creates a C library file named sample.so from sample.c:

```
$ cc -G sample.c -o sample.so
```

In 64-bit mode:

If you use the C library file in a 64-bit-mode HiRDB system, compile the file in 64-bit mode. In this case, specify the -xarch=v9 compiler option.

(c) In Linux

Create a C library file by using the gcc command with the -shared option specified. The following example creates a C library file named sample.so from sample.c:

```
$ gcc -shared -fPIC -o sample.so sample.c
```

(d) In AIX

Create a C library file named `sample.so` from `sample.c`:

1. Create an object file by using the `xlc` command.

   ```
   $ xlc -c -o sample.o sample.c
   ```

2. Create a C library file by using the `xlc` command with the `-G` option specified.

   ```
   $ xlc -G -bexpall -o sample.so sample.o
   ```

In 64-bit mode:

If you use the C library file in a 64-bit-mode HiRDB system, compile and link the file in 64-bit mode. In this case, specify the `-q64` compiler option and `-b64` linker option.

In the POSIX library edition:

If you use the C library file in a 64-bit-mode HiRDB system, make sure that the following conditions are satisfied, because the file must support multiple threads:

- Compile the C file by using the `xlc_r` command.
- Use thread-safe functions.

(e) In Windows

Create a C library file (DLL file) named `sample.dll` from `sample.c`:

1. Create an object file by using the `cl` command.

   ```
   cl /MD /c sample.c
   ```

2. Create a C library file (DLL file) by using the `link` command. Also create a module definition file (`sample.def`) and specify the function to be exported.

   ```
   link /dll /def:sample.def sample.obj
   ```

Notes

- Do not specify the base address (default load address) of the DLL to be created. If the base address is specified, address contention might occur between the DLLs of HiRDB and of the system, resulting in an overload of DLL loading processing.
- You must use a multi-thread DLL edition (/MD) of the Microsoft Visual C++ runtime library. If any other library edition is used, area management processing might fail, resulting in abnormal termination of server processes.

10.2.2 Registering the C library file

Register (copy) the created C library file into the HiRDB server. C library files can be registered only by the UAP's developer or the HiRDB administrator.

- **The UAP developer registers the file**

  Specify the INSTALL CLIB SQL statement in the UAP and then execute the UAP.

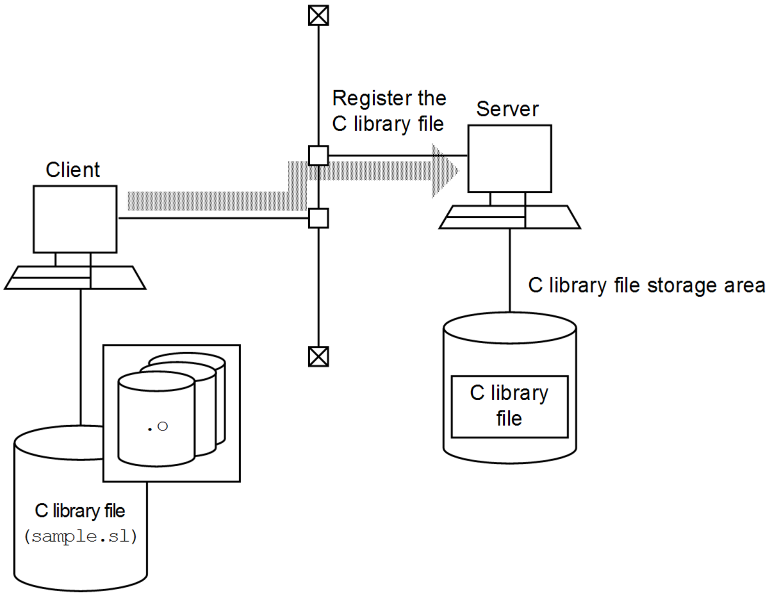
- **The HiRDB administrator registers the file**

  Execute the pdclibsync command (with the -I option specified). Only the HiRDB administrator can execute the pdclibsync command.

The C library file will be stored in the directory specified in the pd_c_library_directory operand.

The following figure provides an overview of C library file registration.

Figure 10-6: Overview of C library file registration



Reference note:

You use the REPLACE CLIB SQL statement to re-register a C library file. You use the REMOVE CLIB SQL statement to delete a C library file. The HiRDB administrator must use the pdclibsync command to re-register or delete a C library file.

Note that only the user who registered a C library file or the HiRDB administrator can re-register or delete it.

10.2.3 Defining the external C stored routine

You use CREATE PROCEDURE or CREATE FUNCTION to define an external C stored routine. CREATE PROCEDURE or CREATE FUNCTION defines the association of a C function with the procedure name or function name.

- C stored procedure

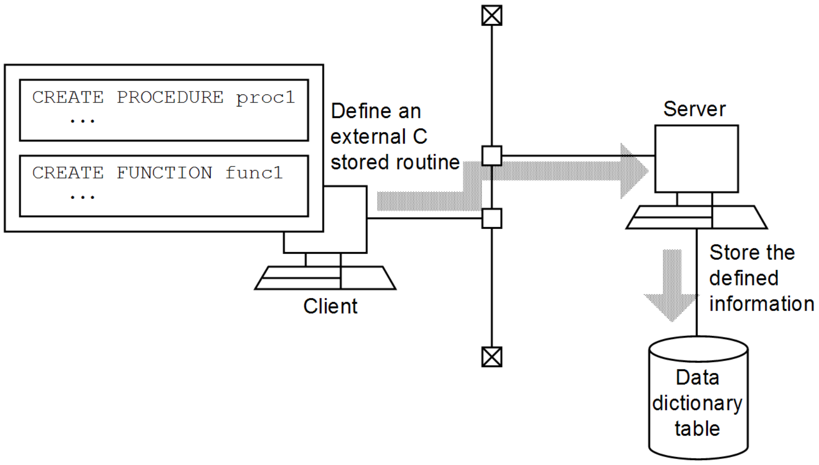
  Use CREATE PROCEDURE to register a C function as a C stored procedure.

- C stored function

  Use CREATE FUNCTION to register a C function as a C stored function.

The following figure shows an example of external C stored routine definition.

Figure 10-7: Example of an external C stored routine definition



Defining public routines

To use an external C stored routine defined by another user, you must specify its owner's authorization identifier and the routine identifier when you call the stored routine from within a UAP.

However, if the external C stored routine was defined as a *public routine* by executing CREATE PUBLIC PROCEDURE or CREATE PUBLIC FUNCTION, then there is no need to specify the owner's authorization identifier when the stored routine is called from within a UAP (only the routine identifier needs to be specified).

Redefining external C stored routines

You use ALTER PROCEDURE or ALTER ROUTINE to redefine an external C stored routine, such as when you need to make changes to the C program.

Deleting external C stored routines

You use DROP PROCEDURE or DROP FUNCTION to delete an external C stored routine.

To delete a public routine, use `DROP PUBLIC PROCEDURE` or `DROP PUBLIC FUNCTION`. Note that a public routine can be deleted only by the user who defined it or by a user with the DBA privilege.

10.2.4 Executing the external C stored routine

You use the `CALL` statement or a function call to execute an external C stored routine. When the `CALL` statement or an SQL statement specifying a function call executes, the C function is called as the external C stored routine and executed on the server machine.

- C stored procedure

  Use the `CALL` statement to execute a C function as a C stored procedure.

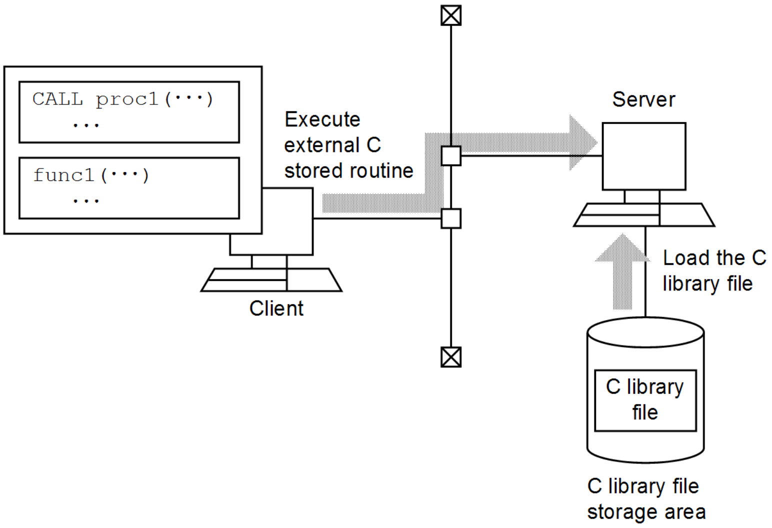
- C stored function

  Use an SQL statement specifying a function call to execute a C function as a C stored function.

For details about the `CALL` statement and function calls, see the *HiRDB Version 9 SQL Reference* manual.

The following figure shows an example of executing an external C stored routine.

Figure 10-8: Example of external C stored routine execution



10.3 Sample external C stored routine programs

This sample external C function acquires the fraction part from a real number.

- **Body of the C function (file name: sample1.c)**

```
/* ALL RIGHTS RESERVED,COPYRIGHT (C)2007,HITACHI,LTD. */
/* LICENSED MATERIAL OF HITACHI,LTD. */
/************************************************************************/
/* name = HiRDB 08-03 sample program of C stored function 1  */
/************************************************************************/
#include <math.h>
#include <stdio.h>

/*---------------------------------------------------------------------*/
/* Define structure for receiving VARCHAR-type data      */
/*---------------------------------------------------------------------*/
typedef struct varchar{
  short len;
  char str[80];
}VCH;

/************************************************************************/
/* name = sample1                                                   */
/************************************************************************/
 void func_modf(double *value, double *ret,
                short *ind1, short *ind_ret,
                char *sqlstate, VCH *routine_name,
                VCH *specific_name, VCH *message_text )
{
    double int_value;

    /* Set fraction part obtained by calling modf function as return value  */
    *ret = modf(*value, &int_value);

    /* Set indicator part of return value (non-null value) */
    *ind_ret = 0;

    /* Set SQLSTATE (normal termination) */
    memcpy(sqlstate, "00000", 5);

    /* Set null character string in message text */
    message_text->len = sprintf(message_text->str, "");
}
```

The following is an example of defining and executing a C stored procedure using the C function shown above:

- **Compiling the C file**

  This example is for an HP-UX (32-bit mode). For examples for other OSs, see *10.2.1(4) Examples of C library file creation for different OSs*.

```
cc +z -c sample1.c
```

Creating the C library file

This example is for an HP-UX (32-bit mode). For examples for other OSs, see *10.2.1(4) Examples of C library file creation for different OSs*.

```
ld -b -o sample1.sl sample1.o
```

Registering the C library file (using the INSTALL CLIB SQL statement)

```
INSTALL CLIB 'sample1.sl' ;
```

Defining the C stored function

```
CREATE function func_modf( parm1 FLOAT ) RETURNS FLOAT
  LANGUAGE C
  EXTERNAL NAME 'sample1.sl!func_modf'
  PARAMETER STYLE RDSQL;
```

Executing the C stored function

```
select func_modf(double_value) from t1
```

10.4 Limitations to C program creation

The following limitations must be observed when you code control processing in C language:

- None of the following functions can be used; using them might result in serious adverse effects on the HiRDB operation:
    - Process manipulation functions, such as `fork()`, `exit()`, `abort()`, and `exec()`
    - sleep(),select(),wait()
    - Stack manipulation functions (such as `setjmp()` and `longjmp()`)
    - Shared memory manipulation functions
    - Semaphore manipulation functions
    - Socket manipulation functions
    - System resource manipulation functions (such as `setrlimit`)
    - mmap(),munmap()
    - gethostent(),sethostent(),endhostent(),gethostbyname(),gethostbyaddr(),herror ()
    - tempnam(),tmpnam()
    - pstat()
    - system()
- Do not create threads.
- Do not use GUI.
- SQL statements cannot be specified.
- Do not manipulate files.
- Do not use special files (such as PIPE).
- Do not use the standard input, standard output, or standard error output.
- Do not use recursive function calls.
- None of the following names is permitted as global variable and function names:
    - Names beginning with the upper-case letters `SQL`, `Y`, or `Z`
    - Names beginning with the lower-case letters `p_`, `pd`, `yy`, or `z`
    - Names beginning with the lower-case letter `_p`

  - Names beginning with the lower-case letters `da`, `dbr`, or `dp`
- Do not set or change environment variables.
- Do not perform signal manipulation.
- Do not change the system date or time.
- If you allocate memory, make sure that the memory is released when the routine terminates.
- The stack size cannot exceed 4,096 bytes.

Chapter
11. UAP Troubleshooting

This chapter explains the collection of historical information for UAP execution and the collection of error information to be used for troubleshooting. It also describes the types of UAP errors and the recovery methods.

This chapter contains the following sections:

11.1 Gathering error information
11.2 UAP error recovery

11.1 Gathering error information

When an error occurs in a UAP, the troubleshooting functions can be used to investigate the cause of the error. The troubleshooting functions are as follows:

- SQL tracing
- Client error log facility
- Facility for output of extended SQL error information
- UAP statistical report facility
- Command trace facility
- SQL trace dynamic acquisition facility
- Reconnect trace facility

11.1.1 SQL tracing

This function collects in an SQL trace file the SQL trace information for an executed UAP.

If an SQL error occurs during UAP execution, the SQL trace information can be used to identify the SQL statement that caused the error.

When the current SQL trace file becomes full, a new file is swapped in and the previous information in that file can be overwritten.

(1) Collecting SQL trace information

SQL trace information is collected by setting values in the `PDCLTPATH` and `PDSQLTRACE` environment variables during client environment definition. For details about the client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

This subsection describes the output destination and file names of SQL traces.

- ■ Output destination

  SQL trace files are output to the directory specified in `PDCLTPATH`. If `PDCLTPATH` is not specified, the SQL trace files are output to the current directory.

- ■ File names

  The following table shows the files names that are assigned.

Table 11-1: Whether an X/Open-compliant API (TX_ function) is used and the file names that are assigned

Use of TX_function	File names that are assigned
No	`pdsql1.trc` and `pdsql2.trc`
Yes	`pdsql`*xxxxx*`-1.trc` and `pdsql`*xxxxx*`-2.trc`

xxxxx: Process ID during UAP execution

The following table lists the SQL trace file names for each library type.

Table 11-2: SQL trace file names for each library type

Library type#	Whether TX_ function is used	PDXATRCFILEMODE environment variable	Client library	
			Version earlier than 08-05	Version 08-05 or later
Regular library	--	--	`pdsql1.trc` or `pdsql2.trc`	`pdsql1.trc` or `pdsql2.trc`
Single-thread edition of XA library	No	LUMP	`pdsql1.trc` or `pdsql2.trc`	`pdsql1.trc` or `pdsql2.trc`
		SEPARATE	`pdsql`*xxxxx*`-1.trc` or `pdsql`*xxxxx*`-2.trc`	`pdsql1.trc` or `pdsql2.trc`
	Yes	LUMP	`pdsql1.trc` or `pdsql2.trc`	`pdsql1.trc` or `pdsql2.trc`
		SEPARATE	`pdsql`*xxxxx*`-1.trc` or `pdsql`*xxxxx*`-2.trc`	`pdsql`*xxxxx*`-1.trc` or `pdsql`*xxxxx*`-2.trc`
Multi-thread edition of XA library	No	--	`pdsql1.trc` or `pdsql2.trc`	`pdsql1.trc` or `pdsql2.trc`
	Yes	LUMP	`pdsql1.trc` or `pdsql2.trc`	`pdsql1.trc` or `pdsql2.trc`
		SEPARATE	`pdsql`*xxxxx*`-1.trc` or `pdsql`*xxxxx*`-2.trc`	`pdsql`*xxxxx*`-1.trc` or `pdsql`*xxxxx*`-2.trc`

Legend:

--: Not applicable

#

The following table shows the library types:

Library type	Library names in UNIX environment#1	Library names in Windows environment

Regular library	`libzclt.sl,ibzclt64.sl,`
	`libzclts.sl,libzclts64.sl,`
	`libzcltk.sl,libzcltk64.sl,`
	`libzcltm.sl,libclt.a,`
	`libclt64.a,libclts.a,`
	`libclts64.a,libcltk.a,`
	`libcltk64.a,libcltm.a`
		`cltdll.dll,pdcltm32.dll,`
		`pdcltp32.dll,`
		`pdcltm50.dll,`
		`pdcltm71.dll,`
		`pdcltm80s.dll,`
		`pdcltm64.dll,`
		`pdcltm90s.dll,`
		`pdcltm90s64.dll`

Single-thread edition of XA library	`libzcltx.sl,libzclty.sl,`
	`libzcltx64.sl,libzclty64.sl,`
	`libzcltxs.sl,libzcltys.sl,`
	`libcltxa.a,libcltya.a,`
	`libcltxas.a,libcltyas.a`
		`pdcltx32.dll,`
		`pdcltxs.dll,pdcltx64.dll,`
		`pdcltxs64.dll`

Multi-thread edition of XA library	`libzcltxk.sl,libzcltyk.sl,`
	`libzcltxk64.sl,`
	`libzcltyk64.sl,`
	`libcltxak.a,libclttyak.a`
		`pdcltxm5.dll,`
		`pdcltxm64.dll,`
		`pdclto32.dll`#2`,pdclto64.dll`#2

#1

The suffix for shared libraries depends on the platform. It is `.so` in Solaris and Linux, and `.a` in AIX.

#2

Created as a regular library for an XDS client.

(2) Examining SQL trace information

SQL trace information is output after the execution of SQL statements is completed. An example of output of SQL trace information is shown as follows, followed by an explanation.

Output example

```
                           [20]                   [19] [22]
** UAP TRACE (CLT:VV-RR(Mmm dd yyyy) SVR:VV-RR US) WIN32(WIN32) **

  USER APPLICATION PROGRAM FILE NAME : XXXXXXXX [1]
  USERID : YYYYYYYY [2]
  UAP START TIME : YYYY/MM/DD HH:MM:SS [3]
  UAP ENVIRONMENT : [4]
    LANG(ja_JP.SJIS)
    USER("hirdb")
    HOST(h9000vr5)
    NAMEPORT(20281)
    FESHOST()
    SVCGRP()  SVCPORT()  SRVTYPE()
    SWAIT(600)  CWAIT(0)  SWATCH(0)

    BLKF(1)  RDABLKF(-1)  LOCKLMT(0)  ISLLVL(2)  DBLOG(ALL)  DFLNVAL(NOUSE)
    AGGR(1024)  DLKPRIO(64)  EXWARN(NO)  VWOPTMODE(0)
    LOCKSKIP(NO) CLTGRP(A)  DSQLOBJCACHE(YES)  PLGIXMK(NO)
    CLTRCVPORT(5000)  CLTRCVADDR(192.134.35.4)  PLGPFSZ(8192)
    PLGPFSZEXP(8192)  SPACELVL(-1)  STJTRNOUT()
    OPTLVL("SELECT_APSL","RAPID_GROPING")
    ADDITIONALOPTLVL("COST_BASE_2","APPLY_HASH_JOIN")
    UAPREPLVL()  REPPATH()
    TRCPATH()

    IPC(MEMORY)  SENDMEMSIZE(16)  RECVMEMSIZE(32)
    HASHTBLSIZE(128)  CMMTBFDDL(NO)  PRPCRCLS( )
    SQLTRCOPENMODE(SQL)  AUTOCONNECT(ON)  CWAITTIMEWRNPNT(-1)  TCPCONOPT(0)
    WRTLNFILSZ(-1) WRTLNCOMSZ(1024)
    WRTLNPATH( ) UAPENVFILE( )
    TP1SERVICE(NO) AUTORECONNECT(NO) RCCOUNT(0) RCINTERVAL(0)
    KALVL(0) KATIME(0) CLTCNVMODE(NOUSE)
    PRMTRC(YES) PRMTRCSIZE(256) BESCONHOLD() BESCONHTI(-1)
    BLKBUFFSIZE(0) BINARYBLKF(NO) FORUPDATEEXLOCK(NO)
    CNSTRNTNAME() SQLTEXTSIZE(4096) RCTRACE(-1)
    FESGRP()
    NBLOCKWAITTIME(0) CONNECTWAITTIME(300) DBBUFLRU(YES)
    UAPEXERLOGUSE() UAPEXERLOGPRMSZ() HJHASHINGMODE(TYPE1)
    DDLDEAPRP(NO) DELRSVWDFILE() HATRNQUEUING()
    ODBSPLITSIZE(100) NODELAYACK(NO) CURSORLVL(0)
    TAAPINFPATH() TAAPINFMODE(0) TAAPINFSIZE(409600)
    JETCOMPATIBLE(NO) SUBSTRLEN() BLKFUPD() ARYERRPOS()
    CALCMDWAITTIME(0) BLKFERRBREAK(NO) XAAUTORECONNECT(NO)
    CLTBINDLOOPBACKADDR(NO)
    STANDARDSQLSTATE() LCKWAITTIME(-1) DDLDEAPRPEXE(NO)
  CONNECTION STATUS : [5]
    CURHOST(dcm3500)  CURPORT(4439)  SRVNAME(fes1)
    CNCTNO(1)  SVRPID(8945)  CLTPID(9155)  CLTTID( ) CLTCNCTHDL(0x0)
```

```
 [6]  [7]    [8]  [9]      [10] [11] [12] [13]  [14]          [15]          [16] [23]
CNCT CLPID CLTID NO      OP   SEC  SQL  SQL   START-TIME   END-TIME     OP   EXEC-TIME
NO                      CODE NO   CODE WARN                             TION
---- ----- ----- -- ---- ---  ---- ----- ------------ ------------ ---- -----------
   1  9155     0      1 CNCT  0     0 WC040 16:03:55.720 16:03:58.080 0001     2356125
   1  9155     0      2 AUI2  1     0 -0000 16:03:58.630 16:03:59.400 M000      769651

 *SQL* INSERT INTO STOCK(GNO,GNAME,PLAN,PRICE,QUANTITY,DISCOUNT) VALUES(?,?,
.........17
 ?,?,?,?)
...............................................................17

   1  9155     0      3 SET   2     0 -0000 16:04:00.820 16:04:01.540 M000      719825

 *SQL* SELECT GNO,GNAME,PLAN,PRICE,QUANTITY,DISCOUNT FROM STOCK
.....................17

   1  9155     0      4 OPEN  2     0 -0000 16:04:02.090 16:04:02.800 M000      709123
   1  9155     0      5 FETC  2  -204 -0000 16:04:03.080 16:04:03.790 M000      708902
   1  9155     0      6 SET   2     0 W8800 16:04:04.060 16:04:04.830 M000      765147

 *SQL(AUTHID)* INSERT INTO TBL01 VALUES('12345',12345)  ..........................17


   1  9155     0      7 SAUT  0     0 -0000 16:04:04.834 16:04:04.835 M000         912

 *USER* hirdb01
.........................................................18

   1  9155     0      8 AUI2  3     0 -0000 16:05:05.110 16:05:05.121 M000        9456

 *SQL* INSERT INTO TBL01 VALUES(?,100)  ........................................21
 *PARAM* NO=       1 COD=c5 XDIM=   1 SYS=     0 LEN=      15 IND=       0   .......21
        DATA=30 35 2d 30 35 00 00 00 00 00 00 00 00 00 00     *05-05.........*   ..21

   1  9155            9 DISC  0     0 -0000 16:05:55.110 16:05:56.660 M004     1547893
```

1. UAP name: Displays the name specified in the PDCLTAPNAME environment variable.

2. Authorization identifier: Displays the authorization identifier of the user who executed the UAP.

3. UAP start time: Displays the time at which execution of the UAP started.

4. UAP execution environment: Displays the values of the environment variables when the UAP was executed.

   If the client environment definition is omitted, -1 is displayed for SWATCH, RDABLKF, SPACELVL, HASHTBLSIZE, CWAITTIMEWRNPNT, WRTLNFILSZ, BESCONHTI, RCTRACE, UAPEXERLOGPRMSZ, and LCKWAITTIME.

5. UAP execution status: Displays the status of the connection with the server when

the UAP was executed:

- CURHOST: Connection-destination host name
- CURPORT: Connection port number
- SRVNAME: Front-end server name or single-server name
- CNCTNO: Connection serial number
- SVRPID: Connected server's process number
- CLTPID: UAP process number

  If connection is established from a Type4 JDBC driver, 0 is displayed.
- CLTTID: UAP thread number

  If connection is established from a Type4 JDBC driver, 0 is displayed.
- CLTCNCTHDL: Connection handle

If information cannot be obtained, an invalid value may be displayed (Windows).

6. Connection serial number: Displays the connection serial number. Connection serial numbers are assigned sequentially each time the server accepts CONNECT.
7. Displays the process number of the UAP.

   If connection is established from a Type4 JDBC driver, 0 is displayed. If the correct process number cannot be obtained, an invalid value may be displayed (Windows).
8. UAP thread number: Displays the UAP thread number when the UAP is running in a multi-thread environment. If the UAP is not running in a thread, or connection was established from a Type4 JDBC driver, 0 is displayed. A thread number that cannot be allocated can sometimes be displayed as an invalid value.
9. SQL counter: Displays the SQL counter values. Each time an SQL statement is accepted, the counter value is incremented (from 1 through 999999, after which the counter value returns to 1).
10. Operation code: Displays the operation code that corresponds to each SQL statement.

    The following table shows the SQL statements that correspond to the displayed operation codes.

Operation code	Corresponding SQL statement
ALCR	ALLOCATE CURSOR statement

Operation code	Corresponding SQL statement
AUI2	DELETE statement (static SQL), INSERT statement (static SQL), UPDATE statement (static SQL), LOCK statement (static SQL), PURGE TABLE statement (static SQL), single-row SELECT statement (static SQL), FREE LOCATOR statement (static SQL)
AUI3	Assignment statement (static SQL)
AUX	EXECUTE statement
AUXI	EXECUTE IMMEDIATE statement, all definition SQL statements
AUXO	EXECUTE statement (INTO specified)
CALL	CALL statement
CLIN	INSTALL CLIB
CLOS	CLOSE statement
CLRM	REMOVE CLIB
CLRP	REPLACE CLIB
CMIT	COMMIT statement
CNCT	CONNECT statement
CPRP	Commit prepare#
DESC	DESCRIBE statement (OUTPUT specified)
DEST	DESCRIBE TYPE statement
DISC	DISCONNECT statement, COMMIT statement (RELEASE specified)
DISR	ROLLBACK statement (RELEASE specified)
DIST	Disconnect + Tran Check#
DSCM	Used by the system.
DSET	DEALLOCATE PREPARE statement
DSPR	Used by the system.
DSRL	Used by the system.
FETC	FETCH statement
GETD	GET DIAGNOSTICS
HVAR	DESCRIBE statement (INPUT specified)

Operation code	Corresponding SQL statement
JARI	INSTALL JAR
JARR	REPLACE JAR
JARU	REMOVE JAR
OPEN	OPEN statement (dynamic SQL)
OPN2	OPEN statement (static SQL)
OPNR	OPEN statement (dynamic SQL (multiple cursors))
RENV	Used by the system.
RNCN	CONNECT statement (TO specified)
RNDS	DISCONNECT statement (TO specified)
ROLL	ROLLBACK statement
RSDC	DESCRIBE statement (OUTPUT and RESULT SET specified)
RSFT	FETCH statement (RESULT SET specified)
RSCL	CLOSE statement (RESULT SET specified)
SAUT	SET SESSION AUTHORIZATION statement
SET	PREPARE statement
SINF	Used by the system.
SOPT	Used by the system.
SVLS	Used by the system.
THRE	Used by the system.
THSU	Used by the system.
TRCK	Used by the system.
TRC2	Used by the system.
TRST	Used by the system.
TSCM	Used by the system.
TSRL	Transfer Rollback#
TSPR	Transfer Prepare#

#: Applicable only when the XA interface is used.

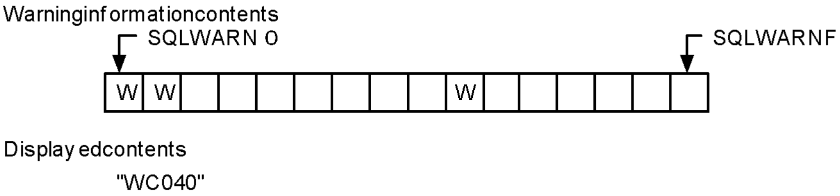
11. Section number: Displays a number for verifying SQL statement correspondence; this number is assigned automatically by the SQL preprocessor.

12. `SQLCODE`: Displays the `SQLCODE` that occurs as a result of SQL statement execution.

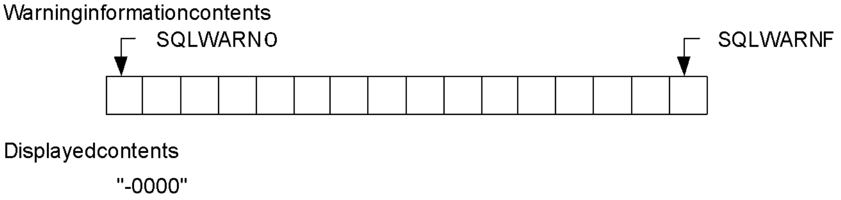
13. `SQLWARN`: Displays warning information (in hexadecimal). Starting from the left, one bit each is allocated to warning information `SQLWARN0` through `SQLWARNF`. A 16-bit value is obtained by setting each bit to `1` if the warning flag is set and to `0` if it is not set. This obtained value is displayed as a 4-digit hexadecimal number.

    `W` is displayed at the beginning if at least one warning flag is set; `-` is displayed if no warning flags are set.

    Example 1



    Example 2



14. SQL statements execution request receipt time: Displays the time at which the SQL execution request was received (in *HH:MM:SS:mmm* format).

15. SQL statement execution request termination time: Displays the time at which the SQL statement execution request was terminated (in *HH:MM:SS:mmm* format).

16. Information used by the system: Displays the information used by the system. If the first byte is `M`, memory is used for process-to-process communication. The other part of the information is used by the HiRDB developer for maintenance purposes.

17. SQL statement: Displays the SQL statement, but only when the operation code is `SET`, `AUXI`, `AUI2`, or `OPN2`.

The maximum SQL statement length that can be output is 4,096 bytes (the PDSQLTEXTSIZE environment variable can be used to change this value); any excess part is truncated. If the -A option was specified during preprocessing, or the /A option was used during preprocessing to specify the authorization identifier to be assumed when the authorization identifier in the SQL statement is omitted, *SQL* is displayed as *SQL (*assumed-authorization-identifier*)*.

18. New user identifier: Displays a new user identifier if the user identifier was changed during a single connection. This information is also displayed if the user identifier change operation fails.

19. Platform for UAP:

Platform	Character string to be displayed
HP-UX 11.0	HP32
HP-UX 11.0 (64-bit mode)	HP64
Solaris	SOL
Solaris (64-bit mode)	SOL64
AIX	AIX
AIX (64-bit mode)	AIX64
Linux	LINUX
Windows	WIN32
HP-UX (IPF) 32-bit mode	HPI32
HP-UX (IPF) 64-bit mode	HPI64
Linux (EM64T)	LINX64
Windows (IPF) 64-bit mode	WINI64
Windows (x64) 64-bit mode	WINX64
Type4 JDBC driver	Type4

20. Library creation date: Displays the creation date of the linked library in the following format:

Mmm: Month (first three letters of the month in English with the first letter in upper case). For example, June is displayed as Jun.

dd: Date

yyyy: Year

21. Parameter trace: Displays input parameter information, output parameter information, and retrieved data when PDPRMTRC=YES, IN, OUT, or INOUT is specified in the client environment definitions.

    The parameter information data is displayed up the length specified in PDPRMTRCSIZE (or 256 bytes if omitted), and any excess part is discarded. For details, see *(4) Parameter trace output examples*.

    NO

    Parameter number

    COD

    Data type code

    XDIM

    Number of array elements

    SYS

    Length of one element, including gaps

    LEN

    Data length

    IND

    Value of indicator variable

    ARRAY NUM

    Number of elements in repetition array

    ROW NUM

    Number of execution rows in SQL that uses embedded variables in an array

    DATA

    Data (dump format)

22. Linked library name

Library name	Displayed characters
libzclt.sl, libclt.a	UNIX, UNIX_32
libzclts.sl, libclts.a	UNIX_S, UNIX_32S
libzcltm.sl, libcltm.a	UNIX_M, UNIX_32M
libzcltk.sl, libcltk.a	UNIX_K, UNIX_32K

Library name	Displayed characters
libzcltx.sl, libcltxa.a	UNIX_XA, UNIX_XA_32
libzcltxs.sl, libcltxas.a	UNIX_XA_S, UNIX_XA_32S
libzcltxm.sl, libcltxam.a	UNIX_XA_M, UNIX_XA_32M
libzcltxk.sl, libcltxak.a	UNIX_XA_K, UNIX_XA_32K
libzclt64.sl, libclt64.a	UNIX_64
libzcltk64.sl, libcltk64.a	UNIX_64K
libzclts64.sl	UNIX_64S
libzcltx64.sl, libzclty64.sl	UNIX_XA_64
libzcltxk64.sl, libzcltyk64.sl	UNIX_XA_64K
libzcltxs64.sl, libzcltys64.sl	UNIX_XA_64S
CLTDLL.DLL	WIN_32
PDCLTM32.DLL	WIN_M32
PDCLTM50.DLL	WIN_M50
PDCLTM71.DLL	WIN_M71
PDCLTM80S.DLL	WIN_M80S
PDCLTP32.DLL	WIN_P32
PDCLTX32.DLL	WIN_XA_32
PDCLTXM.DLL	WIN_XA_32M
PDCLTXS.DLL	WIN_XA_32S
PDCLTXM5.DLL	WIN_XA_50M
PDCLTM64.DLL	WIN_M64
PDCLTX64.DLL	WIN_XA_64
PDCLTXM64.DLL	WIN_XA_64M
PDCLTXS64.DLL	WIN_XA_64S
PDJDBC2.JAR	Type4

23. SQL runtime: Displays the SQL runtime in microseconds when PDSQLEXECTIME=YES is specified in the client environment definitions.

(3) Making a backup of an SQL trace file

If the SQL trace file becomes full while SQL trace information is being output, HiRDB stops writing to that file and outputs SQL trace information to another SQL trace file. Any information that already exists in the switched-in SQL trace file is overwritten in chronological order by the new SQL trace information. To prevent that information from being lost, copy the contents of the SQL trace file into a backup file whenever execution of a UAP is completed.

To determine the SQL trace file that is being used currently, check the most recent update dates/times of the files. The SQL trace file that was updated most recently is the current file.

For a Windows edition HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX edition HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

(4) Parameter trace output examples

Output examples of representative parameter traces are shown below.

(a) INSERT statement (with null value and repetition column)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    7  1088  2060      1 CNCT    0    0 -0000 18:47:21.435 18:47:21.755 0000
    7  1088  2060      2 AUI2    1    0 -0000 18:47:21.765 18:47:21.765 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 65                                       *...e            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=         0
             ARRAY NUM=    5
           0 DATA(    0)=00 01 61                                          *..a             *
           0 DATA(    1)=00 07 62 62 62 62 62 62 62                        *..bbbbbbb       *
           0 DATA(    2)=00 04 63 63 63 63                                 *..cccc          *
           0 DATA(    3)=00 09 64 64 64 64 64 64 64 64 64                  *..ddddddddd     *
           0 DATA(    4)=00 0a 65 65 65 65 65 65 65 65 65 65               *..eeeeeeeeee    *
    7  1088  2060      3 AUI2    2    0 -0000 18:47:21.785 18:47:21.795 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 66                                       *...f            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=         0
             ARRAY NUM=    5
           0 DATA(    0)=00 01 61                                          *..a             *
          -1 DATA(    1)=
           0 DATA(    2)=00 04 63 63 63 63                                 *..cccc          *
          -1 DATA(    3)=
           0 DATA(    4)=00 4f 65 65 65 65 65 65 65 65 65 65 65 65 65 65  *.Oeeeeeeeeeeeeee*
                         65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65  *eeeeeeeeeeeeeeee*
                         --- SAME 3 LINES ---
                         65                                                *e               *
    7  1088  2060      4 AUI2    3    0 -0000 18:47:21.805 18:47:21.815 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 67                                       *...g            *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=        -1
             DATA=
    7  1088  2060      5 DISC    0    0 -0000 18:47:21.825 18:47:21.825 0000
```

Explanation

This is an output example of parameter trace information when INTEGER and VARCHAR(10) repetition column (five elements) values are inserted with the INSERT statement. The values are output in the sequence in which the ? parameters are specified.

1. For input parameters, *INPRM* is displayed. However, when PDPRMTRC=YES, *PARAM* is displayed.
2. For a repetition column, the number of repeated elements is displayed in ARRAY NUM.
3. The number before each DATA clause is the indicator variable of each

element in the repetition column.

4. The number in parentheses in each DATA clause is the repetition column element number.

5. For VARCHAR-type data, the first 2 bytes of DATA is the data length area (the first 4 bytes for BINARY-type data, and the first 8 bytes for BLOB-type data). When PDPRMTRC is YES, the size of the output information is the sum of the defined length and the data area length. When PDPRMTRC is IN, OUT, or INOUT, the size of the output information is the sum of the actual data length and the data area length.

6. If the indicator variable is a negative value, only the information up to DATA= is displayed.

7. If the data extends beyond one line, --- SAME *x* LINES --- (*x* is the number of lines) is output. However, when PDPRMTRC=YES, all data is output.

(b) Single-row SELECT statement

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   12  1492  2260      1 CNCT    0    0 -0000 19:18:31.914 19:18:32.135 0000
   12  1492  2260      2 AUI2    1    0 -0000 19:18:32.135 19:18:32.145 0000
     *SQL* SELECT C2,C3 FROM TBL02 WHERE C1=? AND C4=?
     *OUTPM* NO=    1 COD=c4 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=41 41 41 41 41 41 41 41 41 41                *AAAAAAAAAA      *
     *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 08 61 61 61 61 61 61 61 61                *..aaaaaaaa      *
     *INPRM* NO=    3 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 65                                  *...e            *
     *INPRM* NO=    4 COD=f4 XDIM=    1 SYS=    0 LEN=         2 IND=         0
             DATA=00 62                                        *.b              *
   12  1492  2260      3 DISC    0    0 -0000 19:18:32.155 19:18:32.155 0000
```

Explanation

This is an output example of parameter trace information when PDPRMTRC=INOUT is specified. The retrieval data information is output first in retrieval item sequence, and the input parameter information is output later in the specification sequence.

1. This is the retrieval data information. This information is not output when PDPRMTRC=IN. When PDPRMTRC=YES, *PARAM* is displayed instead of *OUTPM*.

2. This is the input parameter information. This information is not output when PDPRMTRC=OUT. When PDPRMTRC=YES, *PARAM* is displayed instead of *INPRM*.

(c) Stored procedure execution (CALL statement)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   16  1456  2188      1 CNCT    0    0 -0000 19:43:00.486 19:43:00.797 0000
   16  1456  2188      2 CALL    1    0 -0000 19:43:00.797 19:43:00.807 0000
     *SQL* CALL PROC1(IN?,OUT?,INOUT?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 78                                      *...x            *
     *INPRM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 09 63 63 63 63 63 63 63 63 63                 *..ccccccccc     *
     *OUTPM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 dc                                      *....            *
     *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 09 63 63 63 63 63 63 63 63 63                 *..ccccccccc     *
   16  1456  2188      3 DISC    0    0 -0000 19:43:00.829 19:43:00.829 0000
```

Explanation

1. This is the `IN` parameter. When `PDPRMTRC=OUT`, this information is not output.
2. This is the input parameter of the `INOUT` parameter. However, the contents of the `DATA` clause become output data.
3. This is the `OUT` parameter. This information is not output when `PDPRMTRC=IN` or `YES`.
4. This is the output parameter of the `INOUT` parameter. This information is not output when `PDPRMTRC=IN` or `YES`.

(d) Retrieval (FETCH statement)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      1 CNCT    0    0 -0000 14:49:54.326 14:49:54.696 0000
    6   668  1664      2 OPN2    1    0 -0000 14:49:54.736 14:49:54.746 0000
     *SQL* SELECT*FROM TBL03
    6   668  1664      3 FETC    1    0 -0000 14:49:54.746 14:49:54.746 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 78                                       *...x            *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=41 41 41 41 41 41 41 41 41 41                     *AAAAAAAAAA      *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=00 08 61 61 61 61 61 61 61 61                     *..aaaaaaaa      *
    6   668  1664      4 FETC    1    0 -0000 14:49:54.756 14:49:54.756 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 96                                       *....            *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=43 43 43 43 43 43 43 43 43 43                     *CCCCCCCCCC      *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=00 06 63 63 63 63 63 63                           *..cccccc        *
    6   668  1664      5 FETC    1    0 -0000 14:49:54.756 14:49:54.766 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 b4                                       *....            *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=44 44 44 44 44 44 44 44 44 44                     *DDDDDDDDDD      *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=        0
             DATA=00 09 64 64 64 64 64 64 64 64 64                  *..ddddddddd     *
    6   668  1664      6 FETC    1  100 -0000 14:49:54.766 14:49:54.766 0000
    6   668  1664      7 CLOS    1    0 -0000 14:49:54.776 14:49:54.776 0000
    6   668  1664      8 CMIT    0    0 -0000 14:49:54.776 14:49:54.776 0000
```

Explanation

This is an output example of parameter trace information for the FETCH statement. A parameter trace is not output when PDPRMTRC=IN or YES.

1. If the SQLCODE of the FETCH statement is a value other than 0, a parameter trace is not output.

(e) Retrieval (FETCH facility using arrays)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      9 OPN2    2    0 -0000 14:49:54.786 14:49:54.786 0000
    *SQL* SELECT*FROM TBL03
    6   668  1664     10 FETC    2    0 -0000 14:49:54.786 14:49:54.796 0002
    *ROW NUM = 2*
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=         4 IND=         0
           0 DATA(    0)=00 00 00 78                                      *...x             *
           0 DATA(    1)=00 00 00 96                                      *....             *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=        10 IND=         0
           0 DATA(    0)=41 41 41 41 41 41 41 41 41 41                    *AAAAAAAAAA       *
           0 DATA(    1)=43 43 43 43 43 43 43 43 43 43                    *CCCCCCCCCC       *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=        10 IND=         0
           0 DATA(    0)=00 08 61 61 61 61 61 61 61 61                    *..aaaaaaaa       *
           0 DATA(    1)=00 06 63 63 63 63 63 63                          *..cccccc         *
    6   668  1664     11 FETC    2  100 -0000 14:49:54.796 14:49:54.806 0001
    *ROW NUM = 1*
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=         4 IND=         0
           0 DATA(    0)=00 00 00 b4                                      *....             *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=        10 IND=         0
           0 DATA(    0)=44 44 44 44 44 44 44 44 44 44                    *DDDDDDDDDD       *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=        10 IND=         0
           0 DATA(    0)=00 09 64 64 64 64 64 64 64 64 64                 *..ddddddddd      *
    6   668  1664     12 CLOS    2    0 -0000 14:49:54.806 14:49:54.816 0000
    6   668  1664     13 DISC    0    0 -0000 14:49:54.826 14:49:54.846 0000
```

Explanation

This is an output example of parameter trace information for the FETCH facility using arrays. A parameter trace is not output when PDPRMTRC=IN or YES.

1. ROW NUM displays the number of array elements (number of retrieval rows).
2. The number before each DATA clause is the indicator variable of each array element.
3. The number in parentheses in each DATA clause is the array element number.
4. If the SQLCODE of the FETCH statement is a value other than 0, parameter trace information is output for the number of rows returned from the server.

11.1.2 Client error log facility

When an error occurs during communication between a client and the HiRDB server or in the XA interface specified by X/Open, error information is collected as a client error log in a client error log file.

When the current client error log file becomes full, a new file is swapped in and the oldest information in that file is overwritten.

(1) How to collect client error log information

You can collect client error logs by specifying appropriate values in `PDCLTPATH` and `PDUAPERLOG` in the client environment definitions. For details about client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

The two client error log files in which information is to be collected are created under a specified directory. The files that are created depend on whether or not an X/Open-compliant API (`TX_function`) is used.

The following table shows the relationship between use of an X/Open-compliant API (`TX_` function) and the error log files that are created.

Table 11-3: Relationship between use of an X/Open-compliant API (`TX_` function) and error log files that are created

Use of TX_function	Client error log files that are created
No	`pderr1.trc` and `pderr2.trc`
Yes	`pderr`*xxxxx*`-1.trc` and `pderr`*xxxxx*`-2.trc`

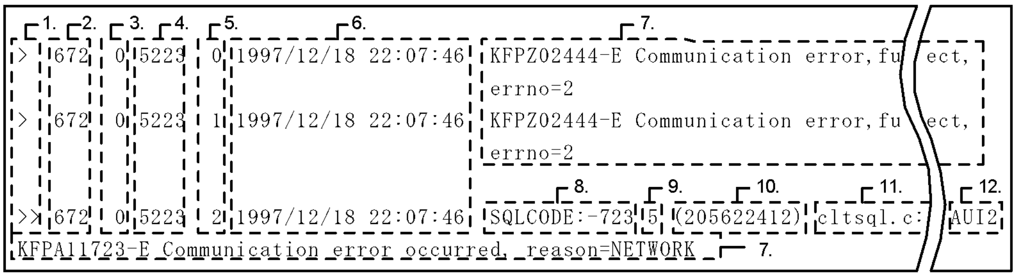
xxxxx: Process ID during UAP execution

(2) How to interpret the client error log information

Client error log information is output whenever an error occurs during SQL statement execution, during communication, and during execution of an X/Open-compliant XA interface function.

An example of client error log information that is output is shown below, followed by an explanation of the output items.

Output example



```
>   672  0 5223  0 1997/12/18 22:07:46 KFPZ02444-E Communication error,fu    ect,
                                       errno=2
>   672  0 5223  1 1997/12/18 22:07:46 KFPZ02444-E Communication error,fu    ect,
                                       errno=2
>>  672  0 5223  2 1997/12/18 22:07:46 SQLCODE:-723 5 (205622412) cltsql.c:   AUI2
KFPA11723-E Communication error occurred, reason=NETWORK
```

1. Client error log lead identifier: `>>` is displayed for an error that occurred during SQL execution; `>` is displayed for any other error.
2. UAP process number: Displays the process number of the UAP where the error

occurred. If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

3. UAP thread number: Displays the UAP thread number when the UAP in which the error occurred is running in a multi-thread environment. Displays 0 if the UAP is not running in a multi-thread environment. The correct thread number cannot be assigned, and an invalid numeric value can sometimes be displayed as a result.
4. Server process number: Displays the process number at the server that is connected.
5. Client error log counter: Displays a value provided by the error log counter. Each time error log information is accepted, the counter value is incremented by 1 (from 0 through 65535).
6. Collection date and time: Displays the date and time the client error log information was collected (in *YYYY/MM/DD HH:MM:SS* format).
7. Log data: Displays the error information (error message).
8. SQLCODE: Displays the SQLCODE when the client error log is for an SQLCODE that is to be returned to the UAP.
9. SQL counter: Displays the SQL counter value for the SQL statement in which the error occurred. If the SQL counter value consists of more than five digits, only the five leading digits are displayed. For details about the SQL counter, see the output example explanation in Section *11.1.1 SQL tracing*.
10. Error collection time: Displays (in milliseconds) the amount of time used to collect the client error log information.
11. Error detection location: Displays the name of the source file and the row number where the error was detected.
12. Operation code: Displays the operation code of the SQL statement in which the error occurred.

(3) Making a backup of a client error log file

If the client error log file becomes full while client error log information is being output, HiRDB stops writing to that file and resumes output of client error log information to the other client error log file. Any information that already exists in the switched-in client error log file is overwritten (beginning with the oldest information) by the new client error log information. To prevent information from being lost by overwriting, you should copy the contents of the client error log file into a backup file whenever execution of a UAP is completed.

To determine the client error log file that is being used currently, check the most recent update dates/times of files. The client error log file that was updated most recently is the current file.

For a Windows edition HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX edition HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

11.1.3 Facility for output of extended SQL error information

(1) What is the facility for output of extended SQL error information

The facility for output of extended SQL error information performs the following functions:

- Outputs the affected SQL statement and parameter information as information for the client error log facility. (The information produced when an SQL statement and parameter information are added to the client error log facility's information is called *SQL error information*.)
- Outputs SQL error information to the server as well. (The file to which SQL error information is output is called the *SQL error report file*.)

(2) Benefits

- ■ Centralized management of SQL error information

  If an SQL error occurs, SQL error information is output on the server side, as well as on the client side. Since SQL error information for multiple clients can be output to the SQL error report file of one server, centralized management of **`SQL error`** information is possible.

- ■ Output of the affected SQL statement and parameter information

  The SQL statement affected by the error and the related parameter information are output. The affected SQL statement can be investigated from this information.

(3) Usage method

When you use the facility for output of extended SQL error information, specify the following system definitions or client environment definitions:

- ■ Whether or not the facility for output of extended SQL error information is to be used

  Use the `pd_uap_exerror_log_use` operand or `PDUAPEXERLOGUSE` to set whether or not the facility for output of extended SQL error information is to be used. Specify the `pd_uap_exerror_log_use` operand to set a value for the entire HiRDB system, and specify `PDUAPEXERLOGUSE` to set a value for each application.

- ■ Output destination directory and maximum size of the SQL error report file

  Use the `pd_uap_exerror_log_dir` operand to set the output directory of the

SQL error report file. Use the `pd_uap_exerror_log_size` operand to set the maximum size of the SQL error report file.

- Maximum data length of the parameter information output to the client error log file or the SQL error report file

  Use the `pd_uap_exerror_log_param_size` operand or `PDUAPEXERLOGPRMSZ` to set the maximum data length of the parameter information output to the error log file or the SQL error report file. Specify the `pd_uap_exerror_log_param_size` operand to set a value for the entire HiRDB system, and specify `PDUAPEXERLOGPRMSZ` to set a value for each application.

(4) Interpreting SQL error information

(a) Output format of the SQL error report file

The output format of the SQL error report file is shown below.

Output format

```
** UAP ERROR INFORMATION aa...aa bbbbbbbbbbbbbbbbbbbbbbbbbb ** [1]

* UAP INFORMATION * [2]
  UAP_NAME(cc...cc) USERID(dd...dd)
  IPADDR(ee...ee) CLTPID(ff...ff) THRDID(gg...gg)
  START_TIME(hhhhhhhhhhhhhhhhhhh)

* SERVER INFORMATION * [3]
  HOST(ii...ii) PORT(jj...jj) PLATFORM(kk...kk)
  SVRNAME(ll...ll) SVRPID(mm...mm)

* SQL INFORMATION * [4]
  OPTIMIZE_LEVEL(nn...nn) ADDITIONAL_OPTIMIZE_LEVEL(oo...oo)
  ISOLATION_LEVEL(pp...pp)

CNCTNO     SQL-       OP   SEC  SQL   SQL   OP   ERROR
           COUNTER    CODE NO   CODE  WARN  TION COUNTER
---------- ---------- ---- ---- ----- ----- ---- -----
rrrrrrrrrr  ssssssssss  tttt  uuuu  vvvvv  wwwww xxxx  yyyyy

START-TIME      END-TIME        EXEC-TIME
--------------- --------------- -----------------
zzzzzzzzzzzzzzz   AAAAAAAAAAAAAAA  BB...BB

* SQL MESSAGE * [5]
  "CC...CC" [DD...DD]

* SQL STATEMENT * [6]
```

```
  "EE...EE"

* PARAMETER * [7]
*ELM NO= FFFFF*
*GGGGG* NO=HHHHH COD=III XDIM=JJJJJ SYS=KKKKK LEN=LLLLLLLLLLL
IND=MMMMMMMMMMM
          ARRAY NUM=NNNNN
        DATA=OO...OO
```

Explanation

1. Title of SQL error report file
2. UAP information
3. Server information
4. SQL information
5. SQL message
6. SQL statement
7. Parameter information

aa...aa

Displays the HiRDB version in the format shown below. (The maximum size of the displayed characters is 8 bytes.)

vv-*rr*-*zz*

If there is no -*zz* value, -*zz* is not output.

bbbbbbbbbbbbbbbbbbbbbbbbbb

Displays the date and time that the error information was output. The output format is shown below. (The maximum size of the displayed characters is 26 bytes.)

YYYY/*MM*/*DD* *hh*:*mm*:*ss*.*uuuuuu*

YYYY: Year

MM: Month

DD: Day

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

cc...cc

Displays the UAP name that was specified in the `PDCLTAPNAME` client environment definition. (The maximum size of the displayed characters is 30 bytes.)

dd...dd

Displays the authorization identifier of the connected user. (The maximum size of the displayed characters is 8 bytes.)

ee...ee

Displays the IP address of the UAP. (The maximum size of the displayed characters is 15 bytes.)

ff...ff

Displays the UAP process number. (The maximum size of the displayed characters is 10 bytes.)

If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

gg...gg

Displays the UAP thread number if the UAP is operating in multiple threads. (The maximum size of the displayed characters is 11 bytes.) If the UAP is not operating in multiple threads, `0` is displayed.

An incorrect number may be displayed if the correct thread number cannot be obtained. If the client version is 07-01 or earlier, `*` is displayed.

hhhhhhhhhhhhhhhhhhh

Displays the UAP execution time in the format shown below. (The maximum size of the displayed characters is 19 bytes.)

YYYY/*MM*/*DD* *hh*:*mm*:*ss*

YYYY: Year

MM: Month

DD: Day

hh: Hour

mm: Minute

ss: Second

ii...ii

Displays the name of the host in which the server process is operating. (The maximum size of the displayed characters is 30 bytes.)

jj..jj

Displays the communication port number of the server process. (The maximum size of the displayed characters is 5 bytes.)

kk...kk

Displays the platform supported by the client library. (The maximum size of the displayed characters is 6 bytes.)

For details about the output information, see the UAP operation platform in *11.1.1(2) Examining SQL trace information*. If the client version is 07-01 or earlier, `*` is output.

ll...ll

Displays the server name of the single server or the front-end server. (The maximum size of the displayed characters is 8 bytes.)

mm...mm

Displays the process number of the server process. (The maximum size of the displayed characters is 10 bytes.)

nn...nn

Displays the value of the SQL optimization option in decimal format. (The maximum size of the displayed characters is 10 bytes.)

oo...oo

Display the value of the SQL extension optimizing option in decimal format. (The maximum size of the displayed characters is 10 bytes.)

pp...pp

Displays the value of the data guarantee level. (The maximum size of the displayed characters is 10 bytes.)

rrrrrrrrrr

Displays the connection sequence number each time the server accepts `CONNECT`. (The maximum size of the displayed characters is 10 bytes.) The displayed connection sequence number is right-justified and padded with leading single-byte space characters.

ssssssssss

Displays the incremented SQL counter value each time an SQL statement is accepted. (The maximum size of the displayed characters is 10 bytes.) The displayed SQL counter value is right-justified and padded with leading single-byte space characters.

tttt

Displays the operation code for the SQL statement. (The maximum size of the displayed characters is 4 bytes.)

uuuu

Displays the section number of the SQL statement. (The maximum size of the displayed characters is 4 bytes.) The displayed section number is right-justified and padded with leading single-byte space characters. If an error occurs during execution of a control SQL, `****` is displayed.

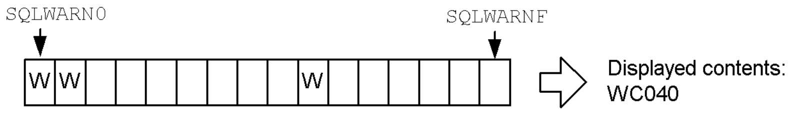
vvvvv

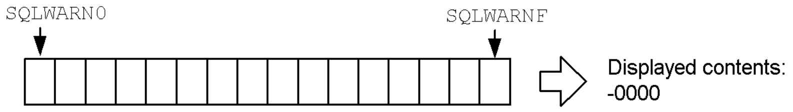
Displays the `SQLCODE` of the SQL execution result. (The maximum size of the displayed characters is 5 bytes.) The displayed `SQLCODE` is right-justified and padded with leading single-byte space characters.

wwwww

Displays warning information in hexadecimal format. (The maximum size of the displayed characters is 5 bytes.) In the warning information, one bit is assigned to each of the items `SQLWARN0` to `SQLWARNF`, starting from the left. If a warning flag is set to one of these items, the corresponding bit is set to `1`. If a warning flag is not set, the bit is set to `0`. All of these bits combined are output as a 4-digit hexadecimal value. If at least one warning flag is set, the 4-digit hexadecimal value is preceded by `W`. If no warning flag is set, the value is preceded by `-`. Examples are shown below.

Example:





xxxx

Displays information that the system uses. (The maximum size of the displayed characters is 4 bytes.)

If the first byte is `M`, it indicates that the inter-process memory communication facility is being used. The other three bytes represent maintenance information. However, if the client version is 07-01 or earlier, `****` is displayed.

yyyyy

Displays the error log number. (The maximum size of the displayed characters is 5 bytes.)

The output error log number is right-justified and padded with leading single-byte space characters. However, If the client version is 07-01 or earlier, `*****` is displayed.

zzzzzzzzzzzzzzz

Displays the time that the SQL execution request was received from the client. The time is displayed in the format shown below. (The maximum size of the displayed characters is 15 bytes.)

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

AAAAAAAAAAAAAAA

Displays the time that processing of the client request ended. The time is displayed in the format shown below. (The maximum size of the displayed characters is 15 bytes.)

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

BB...BB

Displays the processing time of the client request in the format shown below. (The maximum size of the displayed characters is 17 bytes.) The displayed seconds value is right-justified and padded with leading single-byte space characters.

ssssssssss.*uuuuuu*

ssssssssss: Seconds

uuuuuu: Microseconds

CC...CC

Displays the message that was output during SQL execution. (The maximum size of the displayed characters is 254 bytes.)

DD...DD

Displays information that the system uses. (The maximum size of the displayed characters is 21 bytes.)

EE...EE

Displays the SQL statement. (The maximum size of the displayed characters is 2,000,000 bytes.)

If comments or SQL optimization specifications are described in the SQL statement, those are also displayed. If an error occurred during execution of a control SQL statement, `*` is displayed. For details about comments and SQL optimization specifications, see the manual *HiRDB Version 9 SQL Reference*.

FFFFF

Displays the affected element number if an error occurs in an SQL statement that uses an array. (The maximum size of the displayed characters is 5 bytes.)

GGGGG

Displays `INPRM` for input parameter information or `OUTRM` for output parameter information. For input/output parameter information, this variable displays `INPRM` for input information and `OUTRM` for output information. (The maximum size of the displayed characters is 5 bytes.)

HHHHH

Displays the parameter number. (The maximum size of the displayed characters is 5 bytes.)

III

Displays the data-type code. (The maximum size of the displayed characters is 3 bytes.)

JJJJJ

Displays the number of array elements. (The maximum size of the displayed characters is 5 bytes.)

KKKKK

Displays the area length of one element, including gaps. (The maximum size of the displayed characters is 5 bytes.)

LLLLLLLLLLL

Displays the data length. (The maximum size of the displayed characters is 11 bytes.)

MMMMMMMMMMM

Displays the indicator variable value. (The maximum size of the displayed characters is 11 bytes.)

NNNNN

Displays the number of elements in the repetition column if the SQL statement contains a repetition column. (The maximum size of the displayed characters is 5 bytes.) If the SQL statement does not contain a repetition column, this information is not displayed.

OO...OO

Displays parameter information. (The size of the displayed characters is the value specified for the `pd_uap_exerror_log_param_size` operand.) The types of parameter information are input parameter information, output parameter information, and input/output parameter information. The rules pertaining to parameter information are as follows:

- If an input parameter is a `BLOB`-type or `BINARY`-type locator, the value of the `BLOB`-type or `BINARY`-type locator is displayed.
- If the indicator variable is a negative value, only the portion up to `DATA=` is displayed.
- If there is information for several parameters, the parameter information is displayed in the sequence that the parameters were specified.
- If similar data extends beyond one line, `--- SAME` *x* `LINES ---` (*x* is the number of lines) is displayed.
- The size of the displayed parameter information is the sum of the actual data length and the data area length.
- For a repetition column, the number of elements in the repetition column is displayed in `ARRAY NUM`.
- For a repetition column, `DATA` is preceded by an indicator variable for each repetition element.
- For a repetition column, `DATA` is followed by the repetition column element number enclosed in parentheses.

(b) Output format of the client error log file

The output format of the client error log file when the facility for output of extended SQL error information is used is shown below.

Output format

```
>  8355     0  8393     9 2005/08/12 14:06:30 KFPZ03000-I Error
information, type=CONNECT STATUS,
 inf=CLT=07-02(Aug  4 2005):WS SVR=07-02   US:WS LIBTYPE=UNIX_32
>  8355     0  8393    10 2005/08/12 14:06:30 KFPZ03000-I Error
information, type=SQL STREAM,
 inf=insert into t1 values ( ? , ? ,? )
>> 8355     0  8393    11 2005/08/12 14:06:30 SQLCODE:-404
47(140630218) sqaexp0.c   :2348 AUX
 KFPA11404-E Input data too long for column or assignment target
in variable 3
 UAP userprog1,hiuser01 [1]
 SVR host03,1146,sds,hp [2]
 SQLINF
1034,1,2,7,17,-0000,0000,14:06:30.216463,14:06:30.217765,0.001
302 [3]
 SQL INSERT INTO T1 VALUES(?,?,?) [4]
 PRM [5]
 INPRM 1,f1,1,0,4,0
       DATA=00 00 ff ff                            *....
*
 INPRM 2,c1,10,258,255,9
          ARRAY NUM=    9
     0 DATA(   0)=00 01 61                          *..a
*
     0 DATA(   1)=00 02 61 62                       *..ab
*
      0 DATA(    2)=00 03 61 62 63
*..abc          *
      0 DATA(    3)=00 04 61 62 63 64
*..abcd         *
      0 DATA(    4)=00 05 61 62 63 64 65
*..abcde        *
      0 DATA(    5)=00 06 61 62 63 64 65 66
*..abcdef       *
      0 DATA(    6)=00 07 61 62 63 64 65 66 67
*..abcdefg      *
      0 DATA(    7)=00 08 61 62 63 64 65 66 67 68
*..abcdefgh     *
      -1 DATA(    8)=
 INPRM 3,93,1,0,32002,0
        DATA=00 00 00 00 00 00 7d 02 41 41 41 41 41 41 41 41
*......}.AAAAAAAA*
             41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
*AAAAAAAAAAAAAAAA*
              --- SAME 14 LINES ---
```

Explanation

1. UAP information

UAP name

The name of the UAP that was specified in the PDCLTAPNAME client environment definition is displayed.

Authorization identifier

The authorization identifier of the connected user is displayed.

2. Server information

Host name

The name of the host in which the server process is operating is displayed.

Port number

The communication port number of the server process is displayed.

Server name

The server name of the single server or front-end server is displayed.

Platform

The platform supported by the client library is displayed.

For details about the displayed information, see the UAP operation platform in *11.1.1(2) Examining SQL trace information*. If the client version is 07-01 or earlier, * is displayed.

3. SQL information

SQL optimization option

The value of the SQL optimization option is displayed in decimal format.

SQL extension optimizing option

The value of the SQL extension optimizing option is displayed in decimal format.

Data guarantee level

The value of the data guarantee level is displayed.

Connection sequence number

The connection sequence number, which is incremented sequentially each time the server accepts CONNECT, is displayed.

Section number

The section number of the SQL statement is displayed.

SQLWARN

Warning information is displayed in hexadecimal format. In the warning information, one bit is assigned to each of the items SQLWARN0 to SQLWARNF, starting from the left. If a warning flag is set to one of these items, the corresponding bit is set to 1. If a warning flag is not set, the bit is set to 0. All of these bits combined are output as a 4-digit hexadecimal value. If at least one warning flag is set, the 4-digit hexadecimal value is preceded by W. If no warning flag is set, the value is preceded by -. (For examples, see the explanation for *wwwww* in *(a) Output format of the SQL error report file*.)

System information

Information used by the system is displayed.

If the first byte is M, it indicates that the inter-process memory communication facility is being used. The other three bytes represent maintenance information. However, if the client version is 07-01 or earlier, **** is displayed.

SQL start time

The time when the SQL execution request from the client was received is displayed in the following format:

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

SQL end time

The time when the process requested by the client ended is displayed in the following format:

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

SQL runtime

The processing time of the client request is output in the following format. The displayed seconds value is right-justified and padded with leading single-byte space characters.

ssssssssss.*uuuuuu*

ssssssssss: Second

uuuuuu: Microsecond

4. SQL statement

SQL statement

The SQL statement is displayed.

If comments or SQL optimization specifications are described in the SQL statement, those are also displayed. The size of the displayed SQL statement is the value specified for `PDSQLTEXTSIZE` in the client environment definitions.

If an error occurs during execution of a control SQL statement, that SQL statement cannot be obtained and `*` is displayed instead.

For details about comments and SQL optimization specifications, see the manual *HiRDB Version 9 SQL Reference*.

5. Parameter information

ELM NO

If an error occurs in an SQL statement that uses an error, the number of that element is displayed.

Parameter information type

`INPRM` is displayed for input parameter information and `OUTRM` for output parameter information. For input/output parameter information, `INPRM` is displayed for input information and `OUTRM` for output information.

NO

The parameter number is displayed.

COD

The data type code is displayed.

XDIM

The number of array elements is displayed.

SYS

The area length of one element, including gaps, is displayed.

LEN

The data length is displayed.

IND

The value of the indicator variable is displayed.

ARRAY NUM

If the SQL statement contains a repetition column, the number of elements in the repetition column is displayed. If the SQL statement does not contain a repetition column, this information is not displayed.

DATA

Parameter information is displayed.

The types of parameter information are input parameter information, output parameter information, and input/output parameter information.

The rules pertaining to parameter information are as follows:

- If an input parameter is a `BLOB`-type or `BINARY`-type locator, the value of the `BLOB`-type or `BINARY`-type locator is displayed.
- If the indicator variable is a negative value, only the portion up to `DATA=` is displayed.
- If there is information for several parameters, the parameter information is displayed in the sequence that the parameters were specified.
- If similar data extends beyond one line, `--- SAME` *x* `LINES ---` (*x* is the number of lines) is displayed.
- The size of the displayed parameter information is sum of the actual data length and the data area length.
- For a repetition column, the number of elements in the repetition column is displayed in `ARRAY NUM`.
- For a repetition column, `DATA` is preceded by an indicator variable for each repetition element.
- For a repetition column, `DATA` is followed by the repetition column element number enclosed in parentheses.

(5) Rules for SQL error report files

The rules pertaining to the SQL error report files are described below. To view an SQL error report file, use a text editor or similar software.

1. HiRDB executes SQL statements, and each time it detects an error, it opens the SQL error report file, writes SQL error information, and then closes the file. Since

the SQL error information is appended to the final position of the SQL error report file, SQL error information accumulates in the file in chronological order.

2. Two SQL error report files are created (`pduaperrlog1` and `pduaperrlog2`). If the size of the file to which data is currently being written exceeds the specified value of the `pd_uap_exerror_log_size` operand in the system definition, the output destination is switched to the other file. The system uses the two files alternately by performing this switching process on the other file as well. (After switching takes place, the contents of the previous file are deleted.) After HiRDB is started, the file that was most recently updated becomes the output destination.
3. When SQL processing ends, the SQL error report files are closed. Therefore, when an SQL statement is not being executed, you can use an OS command to back up or view the files. Even while an SQL statement is being executed, you can back up or view the file that is not the output destination.
4. To determine the SQL error report file that is being used currently, use the OS's `dir` command (`ls -l` command in UNIX) to check the most recent update dates of the files. The SQL error report file that was updated most recently is the current file.

(6) Notes

1. When the facility for output of extended SQL error information is used, time is required for executing a system call that retrieves the SQL start time and runtime, even if SQL error information is not output.
2. If the OS detects an error (such as a file system failure or an invalid file write privilege) while information is being output to a client error log file or to an SQL error report file, SQL error information is not output to the client error log file or SQL error report file.
3. When the facility for output of extended SQL error information is used, memory space becomes necessary because parameter information is output.

11.1.4 UAP statistical report facility

The UAP statistical report facility outputs UAP statistical information during UAP execution to a UAP statistical report file.

(1) How to obtain the UAP statistical report

To obtain UAP statistical reports, specify values to `PDCLTPATH`, `PDSQLTRACE`, and `PDUAPREPLVL` in the client environment definitions.

This subsection describes the output destination and names of the UAP statistical report files.

- Output destination

  Two UAP statistical report files are output to the directory specified in

PDCLTPATH. If you specify PDREPPATH, the UAP statistical report files are output to a different directory than the one specified in PDCLTPATH.

- File names

pd*HHMMSSmmm*_*xxxxxxxxxx*_1.trc or
pd*HHMMSSmmm*_*xxxxxxxxxx*_2.trc

Explanation:

HHMMSSmmm: Time (hour, minute, second, millisecond) at which the request to connect to HiRDB was issued

xxxxxxxxxx: Connection number (up to 10 bytes)

You can open and close SQL trace files in CONNECT and DISCONNECT units by specifying CNCT in PDSQLTRCOPENMODE. For details about the individual client environment definitions, see *6.6 Client environment definitions (setting environment variables)*.

You can specify the information to be obtained using PDUAPREPLVL in the client environment definitions. The following table shows the relationship between the value of PDUAPREPLVL and the information to be obtained.

Table 11-4: Relationship between the value of PDUAPREPLVL and information to be obtained

Value of PDUAPREPLVL	Information to be obtained			
	By SQL	By UAP	Access path information	SQL runtime interim results
s#	Y	N	N	N
u	N	Y	N	N
p	N	N	Y	N
r	N	N	N	Y
su#	Y	Y	N	N
sp#	Y	N	Y	N
sr#	Y	N	N	Y
up	N	Y	Y	N
ur	N	N	Y	Y
pr	N	N	Y	Y

Value of PDUAPREPLVL	Information to be obtained			
	By SQL	By UAP	Access path information	SQL runtime interim results
sur#	Y	Y	N	Y
spr#	Y	N	Y	Y
upr	N	Y	Y	Y
a or sup#	Y	Y	Y	Y

Y: Information is obtained.

N: Information is not obtained.

#: If s is specified, SQL trace information is also obtained.

■ Notes

1. When access path information or SQL runtime interim results are obtained, the server's workload may increase because the system re-creates an SQL object even if the SQL object is found in the buffer.
2. Information is not output for a UAP in the following cases:
   - The program uses an X/Open-compliant API under OLTP.
   - The UAP terminated without issuing DISCONNECT.
3. The facility does not display access path information or SQL runtime interim results if the size of the information exceeds one gigabyte.
4. For time information (such as SQL execution time, lock-release wait time, and CPU time), 0 is displayed for any value that is too small to be obtained by using system calls of the OS.
5. For a HiRDB/Parallel Server, the information by the UAP does not include the privilege checking executed at the connected dictionary server.
6. If you specify output of access path information or SQL runtime interim results, and you are using the inter-process memory communication facility (PDIPC=MEMORY specified in the client environment definitions), the PDIPC specification is ignored and PDIPC=DEFAULT is assumed.

■ Size of an SQL trace file

You can use the formula shown below to determine the size of an SQL trace file. Note that you can change the size of SQL statements by using the PDSQLTEXTSIZE environment variable.

Size of SQL trace file = 3208 + *A* + 80 x number of operations + total length of SQL statements (maximum of 4096) (bytes)

A: Total length of character strings specified in `PDHOST`, `PDFESHOST`, `PDSQLOPTLVL`, `PDADDITIONALOPTLVL`, `PDREPPATH`, and `PDTRCPATH` in the client environment definitions

To output information by SQL, information by UAP, access path information, and SQL runtime interim results, add the following sizes (bytes):

Information by SQL: 83# x *number of SQL statements*

Information by UAP: 2740# x *number of DISCONNECTs*

Access path information: See *(2)(b) Access path information*.

SQL runtime interim results: See *(2)(c) SQL runtime interim results*.

#: This is the maximum value. The value changes according to the number of digits to be displayed.

(2) Interpreting a UAP statistical report

The following shows a sample UAP statistical report, followed by explanations (a) through (d):

Output example

```
 CNCT  CLPID  CLTID  NO  OP    SEC  SQL   SQL    START-TIME    END-TIME      OP
 NO                      CODE  NO   CODE  WARN                               TION
 ----  -----  -----  --  ----  ---  ----  -----  ------------  ------------  ----
    1   9155      0   1  CNCT    0     0  WC040  16:03:55.720  16:03:58.080  0001
    1   9155      0   2  AUI2    1     0  -0000  16:03:58.630  16:03:59.400  0000

*SQL*  INSERT INTO T1(C1,C2,C3,C4,C5,C6) VALUES(?,?,?,?,?,?)
00:00:00.770    00:00:00.430000    340    1    0    0    0    0    0 ........(a)
    [1]              [2]           [3]    [4]  [5]  [6]  [7]  [8]  [9]

    1   9155      0   3  SET     2     0  -0000  16:04:00.820  16:04:01.540  0000
```

```
*SQL*  SELECT * from T1, T2, T3 where ((T1.C1='a' and T1.C2='A')
 or (T1.C1='a' and T1.C2='B')) and T1.C1=T2.C1 and T1.C2=T2.C2 and T2.C3>=1995
 and T1.C1=T3.C1 and T1.C2=T3.C2 order by T1.C1

00:00:00.720    00:00:00.240000    480    1     0    0    0

Result of SQL Optimizer :   .........................................(b)
 Connect No      :   1
 --------------------------------------------------------------------------------
 Section No      : 2
 UAP Source      :XXXXXXXX.ec
 Optimize Mode   : COST_BASE_2
 SQL Opt Level   : 0x00000420(1056) = "PRIOR_NEST_JOIN"(32),"RAPID_GROUPING"(1024)
 Add Opt Level   : 0x00000003(3) = "COST_BASE_2"(1),"APPLY_HASH_JOIN"(2)
 Work Table      : 0
 Table Cost      : 12672.66944
 ----- QUERY EXPRESSION BODY ID : 1 -----
              :
 ----- QUERY ID : 1 -----
              :
 JOIN
   :
 SCAN
   :

 --------------------------------------------------------------------------------

    1   9155      0   4  OPEN    2     0  -0000  16:04:02.090  16:04:02.800  0000


Result of SQL Execution :   ........................................(c)
--------------------------------------------------------------------------------
Connect No      :    1
UAP Source      : XXXXXXXX.ec
Section No      : 2
----- QUERY EXPRESSION BODY ID : 1 -----
   :
----- QUERY ID : 1 -----
   :
JOIN
   :
SCAN
   :
--------------------------------------------------------------------------------


    1   9155      0   9  DISC    0     0  -0000  16:05:55.110  16:05:56.660  0004

UAP INFORMATION:   ..............................................(d)
 [1]UAPNAME()
 [2]SVHOST(dcm3500)  [3]SVPORT(4439)   [4]SVNAME(fes1)    [5]CNCTNO(1)
 [6]SVPID(8945)      [7]CLPID(9155)      [8]CLTTID(0)
 [9]WAITT(0)        [10]CTIME(0)
 [11]ROREQ(0)       [12]ROHITS(0)
 [13]SOREQ(10)      [14]SOHITS(3)    [15]SOCRT(0)        [16]SOMAX(0)
 [17]COMT(0)        [18]ROLB(0)      [19]FROW(0)     [20]DROW(0)     [21]IROW(3)
 [22]UROW(0)        [23]SET(1)      [24]OPEN(2)     [25]FETC(1)     [26]CLOS(0)
 [27]DESC(0)        [28]SEL(1)      [29]INS(3)      [30]UPD(0)      [31]DEL(0)
 [32]LOCK(0)       [33]CRTT(0)      [34]DRPT(0)     [35]ALTT(0)     [36]CRTI(0)
 [37]DRPI(0)      [38]CMTT(0)       [39]CMTC(0)     [40]CRTS(0)     [41]DRPS(0)
 [42]GRTR(0)      [43]GRTS(0)       [44]GRTA(0)     [45]GRTC(0)     [46]GRTD(0)
 [47]RVKR(0)      [48]RVKS(0)       [49]RVKA(0)     [50]RVKC(0)     [51]RVKD(0)
 [52]CRTV(0)      [53]DRPV(0)       [54]PRGT(0)     [55]CRTP(0)     [56]DRPP(0)
 [57]ALTP(0)      [58]CALL(0)       [59]DESI(0)     [60]MISC(0)
 [61]MAXIO(0)     [62]MAXIOM(0)     [63]MINIO(0)    [64]MINIOM(0)
 [65]IOTIM(0)     [66]IOTIMM(0)

 [67]DIDRC(0)     [68]DIDUC(0)      [69]DIDHC(0)    [70]DIDRD(0)    [71]DIDWT(0)
 [72]LBRFC(0)     [73]LBUPC(0)      [74]LBRHC(0)    [75]LBUHC(0)    [76]LBRDC(0)
 [77]LBWTC(0)     [78]BFSHC(2320)   [79]BRDWC(0)    [80]BWTWC(50)
 [81]BLKWC(2)     [82]MWFN(0)       [83]MWFEC(0)    [84]MWFVL(0)
 [85]WFRDC(0)     [86]WFWTC(0)      [87]WBFOC(0)
 [88]MWHTS(0)     [89]MBSL1(0)      [90]MBSL2(0)    [91]MBSL3(0)
 [92]SCHSKD(0)    [93]SCHCHG(0)
 [94]CINSM(0)     [95]CAFLS(0)      [96]CAFWR(0)    [97]CFMAX(0)    [98]CFAVG(0)
 [99]LDIRC(0)    [100]LDIUC(0)     [101]LDIHC(0)   [102]LDIRD(0)
[103]LDIWT(0)    [104]LBFSHC(0)
[105]ARREQ(0)    [106]ARWC(0)      [107]ARWT(0)    [108]ARWTM(0)
[109]ARWTA(0)    [110]ARWTMA(0)    [111]ARSTA(0)   [112]ARSTMA(0)
[113]HJMAX(0)    [114]HJCMC(0)     [115]HJHTC(0)
```

(a) Information by the SQL

1. SQL execution time (milliseconds)

   Displays the SQL execution time in the format *HH*:*MM*:*SS*.*mmm*. If YES is specified in the PDSQLEXECTIME client environment definition, the unit becomes microseconds.

2. SQL execution time at server (microseconds)

   Displays the SQL execution time at the server in the format *HH*:*MM*:*SS*.*mmmmmm*.

3. Difference between 1 and 2 (milliseconds)

   Provides a guideline for communication time. If YES is specified in the PDSQLEXECTIME client environment definition, the unit becomes microseconds.

4. Number of processed rows

   Displays the number of rows processed by the SQL statements that were issued during the session.

5. Work table creations count

   Displays the number of times a work table was created during internal processing for the SQL statements that were issued during the session.

6. Work table deletions count

   Displays the number of times a work table was deleted during internal processing for the SQL statements that were issued during the session.

7. SQL object size (bytes)

   Displays the size of the SQL object created by the SQL statements that were issued during the session.

8. Total comparison count during hash table search processing by hash join, subquery hash execution

   Displays the total number of comparisons that the SQL statements issued during this connection perform on the data having the same hash value during the hash table search.

9. Total hash join search count during hash join, subquery hash execution

   Displays the number of times that the hash table was searched by the SQL statements issued during this connection.

(b) Access path information

A UAP statistical report displays access path information. `Connect No` displays the connect number. By executing an upward search based on the connect number, you can identify the SQL statements displayed in the SQL trace information. You can also use the connect number to find out the execution request start and end times of the SQL statements displayed in the SQL trace information. For dynamic SQL, execute a downward search based on the connect number, while for static SQL, execute an upward search. If you specify information acquisition in SQL units, the SQL execution times are also displayed. If you find an SQL statement that has a long SQL execution time, tune the UAP.

The UAP statistical report facility does not include the following information in the access path information: HiRDB version, number of back-end servers, UAP name, authorization identifier, SQL optimization processing time, and SQL statements. However, if the routine contains data manipulation SQL statements, the facility displays them as the SQL statements.

If the access path is `SELECT-APSL` for a HiRDB/Single Server (access path is to be selected from multiple candidates by the boundary value during execution), the facility displays the boundary value at the beginning, followed by multiple candidates separated by `Section No`.

For details about the access path information, see the access path display utility in the manual *HiRDB Version 9 Command Reference*.

- **Notes**

  1. The facility does not display access path information for an external Java

stored routine.

2. For an SQL routine, the facility displays the access path information if the SQL object's index information becomes invalid due to an index addition or deletion made to the table used within the routine.

3. The access path information increases the size of the SQL trace file. You can use the formula shown below to determine this increase in size. This is just a guideline; the actual size of the access path information depends on the table definitions, index definitions, and SQL statement used.

$$\text{Size of access path information} = 1 + 0.1 \times \text{Number of set operations} + 4 \times \sum_{i=1}^{n} (S_i) \quad \text{(KB)}$$

n: Number of query specifications in SQL statement

Si: Number of tables in query specification i

Note

For a query in the routine, add the length of the SQL statement to the result of this formula.

(c) SQL runtime interim results

A UAP statistical report displays SQL runtime interim results.

When SQL runtime interim results are displayed, the information listed below can be checked. (The number of rows displayed in the results is the number of rows that HiRDB actually processed at the stage that the interim results are displayed.)

- Number of rows fetched from the table
- Number of rows narrowed by the index
- Number of rows in the results for each join
- Number of input/output rows for any duplicate exclusion, `GROUP BY`, `ORDER BY`, or `LIMIT` specified in the query and number of rows in the query results
- Number of rows in the results for each set operation

Use the SQL runtime interim results and the access path information to carry out SQL tuning. For details about using access path information for SQL tuning, see the description of the access path display utility in the manual *HiRDB Version 9 Command Reference*.

Output format

```
 -------------------------------------------------------
Connect No     : aa...a
UAP Source     : bb...b
Section No     : cc...c
 ----- QUERY EXPRESSION BODY ID : ... -----   ............1
  :
 ----- QUERY ID : ... -----   ...........................2
  :
 JOIN   ..................................................3
  :
 SCAN   ..................................................4
  :
```

Explanation

1. Set operation process information

   For details about set operation process information, see *Set operation process information*.

2. Query process information

   For details about query process information, see *Query process information*.

3. Join process information

   For details about join process information, see *Join process information*.

4. Base table search process information

   For details about base table search process information, see *Base table search process information*.

aa...a

Displays the connection sequence number.

bb...b

Displays the UAP source file name.

cc...c

Displays the section number (number for checking the SQL correspondence).

The information after `Connect No` is repeated for each SQL statement. By conducting a search using a connection sequence number and a section number, you can identify correspondences with the SQL statements displayed in SQL trace information and the access path information.

- Set operation process information

```
----- QUERY EXPRESSION BODY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
Order by       : ee...e ROWS
SetOpe Process : ff..f = gg...g ROWS <-- hh...h  ii...i  hh...h
                         :
```

Explanation

aa...a

Displays the query express body ID.

An ID number is assigned to each query expression body that includes a set operation. If the SQL statement consists of multiple query expression bodies, this line is used to separate the information displayed for each query expression body.

When *(b) Access path information* is being displayed, this value corresponds to the query expression body ID displayed in the access path information.

bb...b

Displays the number of rows in the results of the query expression.

cc...c `ROWS <--` *dd...d* `ROWS`

Displays the final number of rows for the process (`LIMIT` process) that gets search results for the maximum number of rows to return.

If `LIMIT` clause is not specified, this line is not displayed.

cc...c

Displays the number of output rows in the `LIMIT` process.

dd...d

Displays the number of input rows in the `LIMIT` process.

ee...e

Displays the number of rows of the sort process (`ORDER BY` process).

This line is not displayed if any one of the following conditions applies:

- An `ORDER BY` clause is not specified.
- The sort processing specified in the `ORDER BY` clause is omitted.
- A `LIMIT` clause is specified.

ff..f `=` *gg...g* `ROWS <--` *hh...h ii...i hh...h*

Displays the number of rows in the results of the set operation.

If multiple set operations are specified, the information is displayed over several lines.

If the facility that executes partitioned scanning of `UNION ALL` is applied (this facility returns the search results of each query in succession without creating a work table), this line is not displayed.

ff...f

Displays the set operation number of the set operation results in the format `LID(`*set-operation-number*`)`.

If access path information is being displayed, this corresponds to the set operation number displayed in the access path information.

gg...g

Displays the number of rows in the set operation results.

hh...h

If the query expression body to be operated is a query specification, this information is displayed in the format `QID(`*query-ID*`)`. If the query expression body to be operated is the joined result of multiple query specifications, `LID(`*set-operation-number*`)` is displayed.

ii...i

Displays the set operation type (`UNION`, `UNION ALL`, `EXCEPT`, or `EXCEPT ALL`). The *hh...h* values before and after this value form the query expression body.

- Query process information

```
----- QUERY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
Order by       : ee...e ROWS
Distinct       : ff...f ROWS <-- gg...g ROWS
Having         : hh...h ROWS
Group by       : ii...i ROWS <-- jj...j ROWS
```

Explanation

aa...a

Displays the query ID.

A number is assigned to each query specification. If the SQL statement consists of multiple query specifications, this line is used to separate the information displayed for each specification.

If access path information is being displayed, this value corresponds to the query ID displayed in the access path information.

bb...b

Displays the number of rows in the query results.

cc...c `ROWS <--` *dd...d* `ROWS`

Displays the final number of rows for the process (`LIMIT` process) that gets the search results for the maximum number of rows to return.

If `LIMIT` is not specified, this line is not displayed.

cc...c

Displays the number of output rows in the `LIMIT` process.

dd...d

Displays the number of input rows in the `LIMIT` process.

ee...e

The number of rows in sort processing (`ORDER BY` processing) is displayed. Note that `ORDER BY` processing may be executed implicitly even if an `ORDER BY` clause is not specified.

This line is not displayed if any one of the following conditions applies:

- An `ORDER BY` clause is not specified.
- The sort processing specified in the `ORDER BY` clause is omitted.
- `ORDER BY` processing is not executed implicitly.
- A `LIMIT` clause is specified.

ff...f `ROWS <--` *gg...g* `ROWS`

Displays the number of rows processed by duplicate exclusion. Note that duplicate exclusion may be executed implicitly even if duplicate exclusion is not specified.

This line is not displayed if any one of the following conditions applies:

- Duplicate exclusion is not specified.
- Duplicate exclusion is not executed implicitly.
- A `LIMIT` clause is specified.

ff...f

The number of output rows in duplicate exclusion processing is displayed.

gg...g

The number of input rows in duplicate exclusion processing is displayed.

hh...h

Displays the number of rows after the `HAVING` clause is evaluated.

If a `HAVING` clause is not specified, this line is not displayed.

ii...i `ROWS <--` *jj...j* `ROWS`

Displays the number of rows processed by grouping (including implicit grouping).

If grouping is not executed, this line is not displayed.

ii...i

Displays the number of output rows in grouping.

jj...j

Displays the number of input rows in grouping.

- Join process information

```
JOIN
 # Join ID      : aa...a
   Row Count    : bb...b ROWS
   Left         : cc...c ROWS
   Right        : dd...d ROWS
   Join Type    : ee...e (ff...f)
```

Explanation

aa...a

Displays the join process ID.

An ID number is assigned to each join process unit, and if there are multiple join processes, the processes are separated with this line.

If access path information is being displayed, this value corresponds to the join process ID displayed in the access path information.

bb...b

Displays the number of rows in the join process results.

cc...c

Displays the number of rows that were fetched from the join partner on the left side.

dd...d

Displays the number of rows that were fetched from the join partner on the right side.

ee...e

- For HiRDB/Single Server and for HiRDB/Parallel Server when the join method is not determined dynamically during SQL execution

  Displays the join process type (`MERGE JOIN`, `NESTED LOOPS JOIN`, `CROSS JOIN`, or `HASH JOIN`).

- For HiRDB/Parallel Server when the join method is determined dynamically during SQL execution

  Displays `SELECT-APSL` as the join process type.

ff...f

Displays the execution type of the join process (`INNER`, `LEFT OUTER`, `EXIST`, `NOT EXIST`, `ALL`, or `VALUE`).

■ Base table search process information

- When no index or only one index is used in the search process

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
   Index Name  : dd...d 0xeeeeeeee(ee...e)
                        Search    : ff...f gg...g
                        Key       : hh...h gg...g
```

Explanation

aa...a(*aa...a*)

Displays the name of the table to be searched and the correlation name (in parentheses). If a correlation name is not being used, the correlation name (in parentheses) is not displayed. If there are several search processes, this line is used to separate the information displayed for each search.

0x*bbbbbbbb*(*bb...b*)

Displays the ID of the table to be searched in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows fetched from the base table.

dd...d

Displays the index name to be used in the search.

This line is not displayed in the following cases:

- The search is performed without the use of an index.
- HiRDB/Parallel Server dynamically determines the search method during SQL execution.

`0x`*eeeeeeee*`(`*ee...e*`)`

Displays the ID of the index used in the search. The ID is displayed in hexadecimal and decimal (in parentheses) formats.

ff...f

Displays the number of rows in the results narrowed by the search condition.

When an index is used in the search, the number of rows that make up the index is displayed, even if there is no search condition.

This line is not displayed when the surrogate facility for plug-in indexes is used to determine the results of a set function.

gg...g

Displays `ELEMENTS` for an index that contains a repetition column and `ROWS` for all other cases.

hh...h

Displays the number of rows in the results narrowed by the key condition.

If there is no key condition, this line is not displayed.

- When multiple indexes are used in the search process

```
SCAN
 # Table Name   : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count    : cc...c ROWS
   Index Name   : dd...d = ee...e 0xffffffff(ff...f)
                        Search     : gg...g hh...h
                        Key        : ii...i hh...h
                        Row Count  : jj...j ROWS
                  dd...d = ee...e 0xffffffff(ff...f)
                        Search     : gg...g hh...h
                        Key        : ii...i hh...h
                        Row Count  : jj...j ROWS
                  dd...d = kk...k ROWS <-- ll...l mm...m ll...l
```

Explanation

aa...a(*aa...a*)

Displays the name of the table to be searched and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed. If there are multiple search processes, this line is used to separate the information displayed for each process.

`0x`*bbbbbbbb*(*bb...b*)

Displays the ID of the table to be searched in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows fetched from the base table.

dd...d

Displays the number of the work table created when `AND PLURAL INDEXES SCAN`# is executed. The work table number is displayed in the `LID(`*work-table-number*`)` format.

If access path information is being displayed, this value corresponds to the work table number displayed in the access path information.

ee...e

Displays the name of the index used to create the work table when `AND PLURAL INDEXES SCAN`# or `OR PLURAL INDEXES SCAN`# is executed. The index name is displayed in multiple lines. However, if a work table is created without the use of an index, `(NO USE)` is displayed as the index name.

`0x`*ffffffff*(*ff...f*)

Displays the index IDs used in the search. The IDs are displayed in hexadecimal and decimal (in parentheses) formats.

gg...g

Displays the number of rows in the results narrowed by the search condition.

Even if there is no search condition, the number of rows that make up the index is displayed when a search using an index is executed.

hh...h

Displays `ELEMENTS` for an index that contains a repetition column and `ROWS` for all other cases.

ii...i

Displays the number of rows in the results narrowed by the key condition.

If there is no key condition, this line is not displayed.

jj..j

Displays the number of rows fetched from the base table.

dd...d = *kk...k* ROWS <-- *ll...l* *mm...m* *ll...l*

Displays the creation sequence of the work tables created when AND PLURAL INDEXES SCAN# is executed. When three or more indexes are used in the search process, this information is displayed in multiple lines.

kk...k

Displays the number of rows in the operation results.

ll...l

Displays the work table that becomes the input for the operation. The work table is displayed in the LID(*work-table-number*) format.

mm...m

Displays the operation type (AND, OR, or ANDNOT) performed on the work tables.

#: For details about AND PLURAL INDEXES SCAN and OR PLURAL INDEXES SCAN, see the description of the access path display utility in the manual *HiRDB Version 9 Command Reference*.

- When a work table is created for retrieving the results of a view table

```
SCAN
 # Table Name   : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count    : cc...c ROWS
```

Explanation

aa...a(*aa...a*)

Displays the view name and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed.

0x*bbbbbbbb*(*bb...b*)

Displays the view ID in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows that were fetched from the table.

- When a work table is created for the `WITH` clause

```
SCAN
 # Table Name   : aa...a(aa...a)
   Row Count    : bb...b ROWS
```

Explanation

aa...a(*aa...a*)

Displays the `WITH` clause query name and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed.

bb...b

Displays the number of rows that were fetched from the table.

- When a work table is created for the derived table specified in the `FROM` clause

```
SCAN
 # Table Name   : aa...a(aa...a)
   Row Count    : bb...b ROWS
```

Explanation

aa...a(*aa...a*)

Displays `(NO NAME)` or `(NO NAME)(`*correlation-name*`)`.

bb...b

Displays the number of rows that were fetched from the table.

- When a work table that HiRDB creates internally is searched

```
SCAN
 # Table Name   : aa...a
   Row Count    : bb...b ROWS
```

Explanation

aa...a

Displays the name of the work table that HiRDB created internally.

The name of the work table that HiRDB created internally is displayed in `(DUMMY` *work-table-number*`)` format.

The work table number is a three-digit integer.

bb...b

Displays the number of rows fetched from the work table that HiRDB created internally.

Notes

1. SQL runtime interim results are displayed when of one of the following SQL statements is executed:

   - Definition SQL#1
   - `ASSIGN LIST` statement#5
   - `CLOSE` statement
   - `DELETE` statement
   - `EXECUTE` statement#1
   - `EXECUTE IMMEDIATE` statement#2
   - `INSERT` statement#3
   - `PREPARE` statement#4
   - `PURGE TABLE` statement#1
   - Single-row `SELECT` statement
   - `UPDATE` statement
   - `COMMIT` statement#1
   - `DISCONNECT` statement#1
   - `ROLLBACK` statement#1
   - If an error that has implicit rollback occurs#1

   #1: SQL runtime interim results are displayed if there is a cursor that has not been closed.

   #2: SQL runtime interim results are displayed for the following SQL statements:

- `ASSIGN LIST` statement
- `DELETE` statement
- `INSERT` statement
- `UPDATE` statement

#3: SQL runtime interim results are displayed when a scalar subquery or a query specification is specified in the `VALUES` clause.

#4: If `YES` is specified in the `PDPRPCRCLS` client environment definition and an SQL identifier being used by an open cursor is reused by a `PREPARE` statement, the SQL runtime interim results of the open cursor are displayed.

#5: SQL runtime interim results are not displayed when `FOR ALTER LIST` is specified.

2. SQL runtime interim results are not displayed for an SQL statement described in a stored procedure, even if the `CALL` statement is executed.
3. SQL runtime interim results are not displayed for a trigger SQL statement described in a trigger, even if the trigger is executed.
4. When HiRDB/Parallel Server is used, the total number of rows of all servers is displayed.
5. The displayed number of rows may not be an accurate value.
6. When SQL runtime interim results are displayed, the size of the SQL trace file increases by the size shown in the expression below. Note this increase when estimating the size of the SQL trace file. However, the size of the interim results varies significantly depending on the table definitions, the index definitions, and the SQL statements. The value estimated with the following expression should be used only as a rough guideline.

Size of SQL runtime interim results

$$= 0.8 + 0.1 \text{ x set-operation-count} + 0.9 \text{ x } \sum_{i=1}^{n} (Si) \text{ (kilobytes)}$$

n: Number of queries specified in SQL statement

Si: Number of tables in query specification *i*

(d) Information by the UAP

1. UAP name

   This is the name of the UAP for which statistical information was edited.

2. Host name

   This is the name of the host at the connected server.

3. Port number

   This is the port number at the connected server.

4. Connected server name

   This is the name of the front-end server or single server that was connected.

5. Connection sequence number

   This is the sequence number assigned by the server each time `CONNECT` is accepted.

6. Server process number

   This is the connected server's process number.

7. Client process number

   This is the UAP's process number. If connection is established from a Type4 JDBC driver, `0` is displayed.

8. Client's thread number

   This is the thread number of the UAP that is running in multi-thread. If connection is established from a Type4 JDBC driver, `0` is displayed.

9. Lock release wait time (milliseconds)[#1]

   This is the length of time during which a lock acquisition request in the server was placed on lock release wait status because another user locked the requested resource.

10. CPU time (milliseconds)[#1]

    This is the CPU time at the server that was used by transaction during UAP execution.

11. Stored procedure's SQL object acquisition requests count

    This is the number of times a stored procedure's SQL object acquisition request was issued for the SQL object buffer at the single server or front-end server.

12. Stored procedure object buffer hits count

    This is the number of times requested information was found in the SQL object buffer at the single server or front-end server.

13. SQL object acquisition requests count

    This is the number of times an SQL object acquisition request was issued for the

SQL statements issued during the session.

14. SQL object buffer hits count

    This is the number of times requested information was found in the SQL object buffer for the SQL statements issued during the session.

15. SQL object creations count

    This is the number of times an SQL object was created for the SQL statements issued during the session.

16. Maximum size of SQL object created (bytes)

    This is the maximum size of the SQL object created with the SQL statements issued during the session.

17. `COMMIT` statement executions count during the session.
18. `ROLLBACK` statement executions count during the session.
19. Number of retrieval rows passed to UAP by the `FETCH` and `SELECT` statements during the session.
20. Number of rows deleted by the `DELETE` statements during the session.
21. Number of rows inserted by the `INSERT` statements during the session.
22. Number of rows updated by the `UPDATE` statements during the session.
23. Preprocessing time during the session.
24. `OPEN` statement executions count during the session.
25. `FETCH` statement executions count during the session.
26. `CLOSE` statement executions count during the session.
27. `DESCRIBE` statement executions count during the session.
28. `SELECT` statement executions count during the session.
29. `INSERT` statement executions count during the session.
30. `UPDATE` statement executions count during the session.
31. `DELETE` statement executions count during the session.
32. `LOCK` statement executions count during the session.
33. `CREATE TABLE` executions count during the session.
34. `DROP TABLE` executions count during the session.
35. `ALTER TABLE` executions count during the session.
36. `CREATE INDEX` executions count during the session.

37. DROP INDEX executions count during the session.
38. COMMENT (TABLE) executions count during the session.
39. COMMENT (COLUMN) executions count during the session.
40. CREATE SCHEMA executions count during the session.
41. DROP SCHEMA executions count during the session.
42. GRANT RDAREA executions count during the session.
43. GRANT SCHEMA executions count during the session.
44. GRANT access privilege executions count during the session.
45. GRANT CONNECT executions count during the session.
46. GRANT DBA executions count during the session.
47. REVOKE RDAREA executions count during the session.
48. REVOKE SCHEMA executions count during the session.
49. REVOKE access privilege executions count during the session.
50. REVOKE CONNECT executions count during the session.
51. REVOKE DBA executions count during the session.
52. CREATE VIEW executions count during the session.
53. DROP VIEW executions count during the session.
54. PURGE TABLE statement executions count during the session.
55. CREATE PROCEDURE executions count during the session.
56. DROP PROCEDURE executions count during the session.
57. ALTER PROCEDURE executions count during the session.
58. CALL statement executions count during the session.
59. DESCRIBE statement (INPUT) executions count during the session.
60. Other SQL executions count during the session.
61. Maximum input/output time (seconds).
62. Maximum input/output time (microseconds). (A value in seconds is not included.)
63. Maximum input/output time (seconds).
64. Maximum input/output time (microseconds) (A value in seconds is not included.)

    Check whether the input and output times are appropriate. If input/output processing takes longer than necessary, obtain and check the hardware log for any hardware errors.

If you used the asynchronous READ facility, the input and output times for batch look-ahead reading by the asynchronous READ process are not included.

65. Cumulative input/output time for database (seconds).

66. Cumulative input/output time for database (microseconds) (A value in seconds is not included.)

    Use this information to determine whether the cause is input/output or CPU.

    If you used the asynchronous READ facility, the input and output times for batch look-ahead reading by the asynchronous READ process are not included.

67. Data, index, and directory page references count

    This is the number of times a data, index, or directory page was referenced from this UAP.

68. Data, index, and directory page updates count

    This is the number of times a data, index, or directory page was updated from this UAP.

69. Data, index, and directory page buffer hits count

    This is the number of times a requested data, index, or directory page was found in the buffer. If the hit rate ((item 69 ÷ item 67) x 100) is low, obtain the global buffer statistical information and tune the global buffer with a low hit rate. In this case, all global buffers are subject to tuning except for the LOB global buffer.

70. Data, index, and directory page real READs count

    This is the number of times a data, index, or directory page was actually read by this UAP.

    If you are using the prefetch facility, the number of look-ahead READs by the prefetch facility is included. If you used the asynchronous READ facility, the number of look-ahead READs by the asynchronous READs process is also included.

    If the buffer hit rate is low, the READs count becomes high.

71. Data, index, and directory page real WRITEs count

    This is the number of times a data, index, or directory page was actually written by this UAP. If the commit output facility is used, this count includes the number of outputs to the database during commit processing.

72. LOB page references count

    This is the number of times a LOB page was referenced by this UAP. This count includes the LOB data and plug-in retrieval operations.

73. LOB page updates count

    This is the number of times a LOB page was updated by this UAP. This count includes the LOB data and plug-in update operations.

74. LOB page reference buffer hits count

    This is the reference buffer hits count. This information is applicable if the LOB global buffer is used. If the hit rate ((item 74 ÷ item 72) x 100) is low, obtain the global buffer statistical information and tune the global buffer with a low hit rate. In this case, the LOB global buffer is subject to tuning. If the LOB global buffer is not used, the hit rate is 0.

75. LOB page update buffer hits count

    This is the update buffer hits count. This information is applicable if the LOB global buffer is used. If the hit rate (item 75 ÷ item 73 x 100) is low for LOB data updating or plug-in index updating, obtain the global buffer statistical information and tune the global buffer with a low hit rate. In this case, the LOB global buffer is subject to tuning. If the LOB global buffer is not used, the hit rate is 0. Update buffer hits are not applicable to addition of new LOB data.

76. LOB page real READs count

    This is the number of times a LOB page was actually read by this UAP. If the LOB global buffer is used and the READ buffer hit rate is low, the READs count becomes high.

77. LOB page real WRITEs count

    This is the number of times a LOB page was actually written by this UAP. When updating a plug-in index, you can reduce the real WRITEs count by using the LOB global buffer.

78. Global buffer flushes count

    This is the number of times the buffer was flushed to create space for a new page. This indicates the number of times a page was swept out of memory because the buffer was full.#2

79. Global buffer READ waits count

    This is the number of times the UAP was placed on wait status because a page in the global buffer was being read from a HiRDB file by another user. This indicates the number of times the UAP was placed on wait status until a READ operation was completed because the page to be referenced was under READ operation by another user.#2

80. Global buffer WRITE waits count

This is the number of times the UAP was placed on wait status because a page in the global buffer was being output to a HiRDB file by another user. This indicates the number of times the UAP was placed on wait status until a WRITE operation was completed because the page to be updated was under WRITE operation by another user.[#2]

81. Global buffer lock release waits count

    This is the number of times the UAP was placed on wait status because a page in the global buffer was in use by another user. This indicates the number of times the UAP was placed in wait status until update processing was completed because the page to be referenced or updated was under update processing by another UAP.[#2]

82. Maximum work table files count

    This is the maximum number of work table files used by this UAP.[#3] You can determine the validity of the `-l` option value (maximum number of files) specified in the `pdfmkfs` command. The value of the `-l` option must satisfy the following condition:[#4]

    Value of `-l` option ≥ total number of work table files for all UAPs that are executed concurrently + 20

83. Maximum work table file extensions count

    This is the maximum number of work table file extensions for this UAP. You can determine the validity of the `-e` option value (maximum number of extensions) specified in the `pdfmkfs` command. The value of the `-e` option must satisfy the following condition:[#4]

    Value of `-e` option ≥ total number of work table file extensions for all UAPs that are executed concurrently

84. Maximum size of work table file (MB)

    This is the maximum size of a work table file for this UAP. You can determine the validity of the `-n` option value (maximum number of extensions) specified in the `pdfmkfs` command. The value of the `-n` option must satisfy the following condition:[#4]

    Value of `-n` option ≥ total size of work table files for all UAPs that are executed concurrently + management area size for HiRDB file system area

85. Work table file READs count

    This is the number of times work table data was input from file to buffer.[#1]

86. Work table file WRITEs count

This is the number of times work table data was output from buffer to file.#1

87. Forced outputs count for the work table buffer

    This is the number of times buffer contents in use were forcibly output to a file due to a shortage of the work table buffer.#1 If this value is not 0, increase the value of the `pd_work_buff_size` operand (size of the work table buffer) in the system definitions.

88. Estimated value for expanding hash table in batch mode (KB)

    This is the estimated size of the hash table required to expand the processed hash data in batch mode during hash join or subquery hash execution.#3

    If the size of the hash table is greater than this value, batch hash join is assumed, which does not involve any packet division.#5 If this value exceeds the specified range of the hash table size, batch hash join is not possible. If this value is 0, hash join or subquery hash execution has not taken place.

89. Maximum packet size at level 1 (KB)

    This is the maximum packet size after level 1 packet division during hash join or subquery hash execution.#3

    If the size of the hash table is at least this value, packet division was completed at level 1. If the packet division level is 2 or more, you can complete the packet division at level 1 by specifying this value as the hash table size.#6 For a batch hash join that does not involve any packet division, this value is 0.

90. Maximum packet size at level 2 (KB)

    This is the maximum packet size after level 2 packet division during hash join or subquery hash execution.#3

    If the size of the hash table is at least this value, packet division was completed at level 2. If the packet division level is 3 or more, you can complete the packet division at level 2 by specifying this value as the hash table size.#6 If level 2 packet division did not take place, this value is 0.

91. Maximum packet size at level 3 (KB)

    This is the maximum packet size after level 3 packet division during hash join or subquery hash execution.#3

    If the size of the hash table is at least this value, data was processed in packets with a maximum level of 3. If the hash table size is not greater than this value, a packet was partially expanded in the hash table, thereby adversely affecting the processing efficiency. In this case, specify at least this value as the hash table size.#6 Alternatively, performance may improve by avoiding the hash join or subquery hash execution. If level 3 packet division did not take place, this value is 0.

92. Unsuccessful page searches count during free space reusage execution

    This is the number of times that the mode was returned to new page allocation mode because the free space reusage facility was unable to find reusable free space when the mode was switched from new page allocation mode to free page reuse mode. If this value is a value other than 0, an inefficient page search process may have occurred during an update or insertion process executed by the UAP. For details about the free space reusage facility, see the *HiRDB Version 9 Installation and Design Guide*.

93. Mode switches count from new page allocation mode to free page reuse mode

    This is the number of times that the mode was switched from new page allocation mode to free page reuse mode when the free area reusage facility was executed. If this value is close to the number of update and insertion processes executed by the UAP, an inefficient page search process may have occurred.

94. Cache buffer shortage occurrences count

    This is internal information used by the system.

95. Cache buffer allocation flushes count

    This is internal information used by the system.

96. WRITEs count during cache buffer area allocation flushing

    This is internal information used by the system.

97. Maximum cache buffer allocation flushes count

    This is internal information used by the system.

98. Average cache buffer allocation flushes count

    This is internal information used by the system.

99. Data and index page references count when local buffer used

    This is the number of times a data or index page was referenced from this UAP.

100. Data and index page updates count when local buffer used

    This is the number of times a data or index page was updated from this UAP.

101. Data and index page buffer hits count in local buffer

    This is the buffer hits count for data pages and index pages.

    If the buffer hit rate (101 ÷ 99 x 100) is low for a UAP that performs random access, tune the buffer.

102. Data and index page real READs count when local buffer is used

This is the number of times a data or index page was actually read by this UAP.

If the prefetch facility is being used, the number of look-ahead READs by the prefetch facility is also included. If the buffer hit rate is low, the READs count becomes high.

103. Data and index page real WRITEs count when local buffer is used

This is the number of times a data or index page was actually written by this UAP.

104. Local buffer flush count

This is the number of times the buffer was flushed to create space for a new page. This indicates the number of times a page was swept out of memory because the buffer was full.

105. Asynchronous READ request count

This is the number of times the asynchronous READ process requested a batch look-ahead read processing when the asynchronous READ facility was used.

106. Synchronization wait count during asynchronous READ

This is the number of times a synchronization wait occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

107. Cumulative synchronization wait time during asynchronous READ (seconds)

This is the cumulative wait time of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

108. Cumulative synchronization wait time during asynchronous READ (microseconds) (A value in seconds is not included.)

This is the cumulative wait time of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

109. Average synchronization wait time during asynchronous READ (seconds)

This is the average wait time of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

110. Average synchronization wait time during asynchronous READ (microseconds) (A value in seconds is not included.)

This is the average wait time of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

111. Average synchronous input/output time during asynchronous READ (seconds)

This is the average synchronous READ time for initial batch reads of the first page when the asynchronous READ facility was used.

112. Average synchronous input/output time during asynchronous READ (microseconds) (A value in seconds is not included.)

This is the average synchronous READ time for initial batch reads of the first page when the asynchronous READ facility was used.

113. Maximum comparison count[#3] during hash table search processing in hash join, subquery hash execution

This is the maximum number of comparisons for data items that have the same hash value in one hash table search.

114. Total comparison count[#1] during hash table search processing in hash join, subquery hash execution

This is the total number of comparisons for data items that have the same hash value during hash table search processing.

115. Total hash table search count[#1] in hash join, subquery hash execution

This is the number of times the hash table is searched.

#1: For HiRDB/Parallel Server, this is the total of all servers.

#2: This is the sum of all global buffers.

#3: For HiRDB/Parallel Server, this is the maximum value of each back-end server.

#4: More resources than the value obtained from the formula may be required due to temporary fragmentation. Therefore, specify a sufficient value.

#5: If the hash table size increases, the number of packet divisions may increase; therefore, a bigger hash table may be required than when the tuning information was obtained. If you have increased the hash table size on the basis of this tuning information, obtain the tuning information again. If an expected result is not obtained, you need to increase the hash table size again on the basis of the obtained tuning information.

#6: If the hash table size increases, the number of packet divisions may increase; therefore, a smaller hash table may be enough to complete packet division at an intended level than when the tuning information was obtained. On the other hand, if you reduce the hash table size, the number of packet divisions may decrease; therefore, packet division may not be completed at the same level as when the tuning information was obtained. Therefore, use the tuning information for the purpose of increasing the hash table size.

11.1.5 Command trace facility

The command trace facility outputs a client's trace information to the command trace file when a command is executed by a UAP (during the execution of the COMMAND EXECUTE SQL statement).

When the command trace file becomes full, the facility overwrites the oldest information.

(1) How to obtain command trace information

You can obtain command trace information by specifying appropriate values in PDCLTPATH and PDCMDTRACE in the client environment definitions. For details about each client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

Two command trace files named pdccmd1.trc and pdccmd2.trc are output to the specified directory.

(2) Interpreting command trace information

Command trace information is output when a command is executed by a UAP. The following shows sample command trace information and explains each item:

Output example

```
** COMMAND TRACE (CLT:06-00:Jan 11 2001) HP32 **   [1]

  USER APPLICATION PROGRAM FILE NAME : TESTAP   [2]
  COMMAND START TIME : 2001/01/11 10:55:27   [3]
  COMMAND EXECUTE ENVIRONMENT & STATUS :   [4]
    PDASTHOST(dcm3500)
    PDASTPORT(20266)
    PDSYSTEMID("HRD1")
    PDUSER("hirdb")
    PDASTUSER("hirdb ")
    PDCMDWAITTIME(0)
    ENVGROUP("")
    CLTPID(9155) CLTTID(0)
  [5]  [6]    [7]         [8]     [9]
  9155  0  2001/01/11 10:55:27   0  pdhold -r RDDATA01
  9155  0  2001/01/11 10:55:27   1  KFPZ02444-E Communication
error,
                                    func=connect, errno=2
```

Explanation

1. Command trace header

   The header contains the following information:

- Version of the linked library
- Library creation date (in the format *Mmm dd yyyy*)
- Platform in use (For details about the character strings that are displayed for the platforms, see the *Explanation* section in *11.1.1(2) Examining SQL trace information*.)

2. UAP name

   This is the value of `PDCLTAPNAME` specified in the client environment definition.

3. Command start date and time

   This is the date and time the command execution began.

4. Command execution environment and status

   This is the value of the client environment definition and status during command execution.

5. UAP process number

   This is the UAP process number. If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

6. UAP thread number

   If the UAP is running with multi-thread, this indicates the UAP thread number; otherwise, a value of `0` is displayed. Note that the facility may display an invalid numeric value if it is unable to obtain the correct thread number.

7. Command trace acquisition date and time

   This is the date and time the command trace information was acquired.

8. Command trace counter

   This is the count that was incremented each time a command trace was accepted. The value range is from `0` to `65535`.

9. Trace data

   This is the trace data.

(3) Backing up the command trace file

If the command trace file becomes full while writing command trace information, HiRDB continues output using another command trace file. In this case, existing contents of the command trace file are overwritten, beginning with the oldest information. Therefore, you should make a backup copy of a command trace file when the UAP is terminated.

To determine the command trace file that is being used currently, check the most recent update dates/times of the files. The command trace file that was updated most recently is the current file.

For a Windows edition HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX edition HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

11.1.6 SQL trace dynamic acquisition facility

The SQL trace dynamic acquisition facility lets you dynamically obtain SQL trace information using a command during UAP execution. Acquisition of SQL trace information begins at the next `CONNECT`.

(1) Output destination and names of SQL trace files

This subsection describes the output destination and names of the SQL trace files.

- Output destination

  Specify the SQL trace file storage directory in `PDTRCPATH` beforehand. Two SQL trace files are output to the specified directory.

- File names

  `pd`*HHMMSSmmm_xxxxxxxxxx*`_1.trc` or
  `pd`*HHMMSSmmm_xxxxxxxxxx*`_2.trc`

  Explanation:

  HHMMSSmmm: Time (hour, minute, second, millisecond) at which the request to connect to HiRDB was issued

  xxxxxxxxxx: Connection number (up to 10 bytes)

When you use the `pdclttrc` command to obtain SQL traces, the destination and names of the SQL trace files are as follows:

- Output destination

  The SQL trace files are output to the directory specified in `PDCLTPATH`. If `PDCLTPATH` is omitted, the SQL trace files are output to the current directory.

- File names

  `pdsql`*xxxxxxxxyyyyyyyyyy*`-1.trc` and `pdsql`*xxxxxxxxyyyyyyyyyy*`-2.trc`

  Explanation:

  xxxxxxxx: Server name (up to 8 bytes)

  yyyyyyyyyy: Server process ID (up to 10 bytes)

For details about the `pdclttrc` command, see the manual *HiRDB Version 9 Command Reference*.

(2) Trace acquisition command (pdtrcmgr)

If the directory specified with the `-d` option is the same as the directory specified in the `PDTRCPATH` client environment definition variable during UAP execution, the `pdtrcmgr` command issues the trace acquisition start and end requests.

(a) Format

```
pdtrcmgr  -d directory-name-specified-in-PDTRCPATH
            [{-b| -e}]
            [-k{[s] [u] [p] [r]| a}]
            [-n PDCLTAPNAME]
            [-s SQL-trace-file-size]
            [-o]
```

(b) Options

- **-d *directory-name-specified-in-*PDTRCPATH**

~ <path name>

Specifies the absolute path name of the value (directory name) specified in the `PDTRCPATH` client environment definition variable to start or stop the acquisition of trace information for the UAP.

The facility issues a trace acquisition start or stop request for all UAPs for which the specified directory matches the directory in `PDTRCPATH`.

- **{-b| -e}**

Specifies whether to start or stop the acquisition of the SQL trace:

`-b`: Starts the acquisition of SQL trace.

`-e`: Stops the acquisition of SQL trace.

- **-k{[s] [u] [p] [r] | a}**

Specifies the information to be output. When this option is omitted, the facility outputs only the SQL trace information.

`s`: Outputs information by the SQL.

`u`: Outputs information by the UAP.

`p`: Outputs access path information.

`r`: Outputs SQL runtime interim results.

`a`: Outputs all information.

`s`, `u`, `p`, and `r` can be specified in different combinations (such as `su`, `spr`, or `spr`). Specifying `sup` is the same as specifying `a`. If `u`, `p`, `r`, `ur`, `pr`, or `upr` is specified, SQL trace information is not output.

When the -e option is specified, the specification of the -k option becomes invalid.

For details about the UAP statistical report, see *11.1.4 UAP statistical report facility*.

- **-n PDCLTAPNAME**

Specifies that only the UAP specified in the PDCLTAPNAME client environment definition variable is to be subject to acquisition of an SQL trace. The facility ignores this option if the -e option is specified.

- **-s *SQL-trace-file-size***

~ <unsigned integer> ((0 or 32,768 to 2,000,000,000)) <<32,768>>

Specifies the size of the SQL trace file in bytes.

If 0 is specified, the maximum file size is assumed. If a value in the range from 32,768 to 2,000,000,000 is specified, the specified size of file is used.

The facility ignores this option if the -e option is specified.

- **-o**

Specifies that SQL trace files are to opened and closed in CONNECT and DISCONNECT units. The facility ignores this option when the -e option is specified.

When SQL trace files are opened and closed in CONNECT and DISCONNECT units instead of operation units (SQL units), the SQL trace output time can be shortened because the overhead is reduced.

If you omit this option, the SQL trace dynamic acquisition facility opens and closes SQL trace files in operation units.

This facility continues to write information as long as the SQL trace file is open. Therefore, if you specify this option, some SQL trace information may be discarded if DISCONNECT cannot be executed properly.

11.1.7 Reconnect trace facility

When the automatic reconnect facility executes reconnection, reconnect trace information, which consists of the connection handle value managed internally by HiRDB, the connection information before and after reconnect, and the reconnect time, is output to the reconnect trace file. This information is used for tracking connection information in the trace output by the PRF trace facility of Cosminexus.

(1) How to obtain the reconnect trace information

Reconnect trace information can be obtained by setting a value in the PDRCTRACE client environment definition.

HiRDB creates two reconnect trace files in the directory specified in the PDCLTPATH client environment definition. The names of the created files are pdrcnct1.trc and pdrcnct2.trc.

(2) Interpreting reconnect trace information

Reconnect trace information is output when the automatic reconnect facility establishes a connection automatically.

An output example of a reconnect trace is shown below.

```
   [1]  [2]                          [3]                                [4]
40004250 S 2004/04/12 11:10:36.766 - 2004/04/12 11:10:41.846 sds:9:23763 =>
sds:10:23750
40004250 S 2004/04/12 11:11:07.491 - 2004/04/12 11:11:12.547 sds:10:23750 =>
sds:11:23765
40004850 F 2004/04/12 11:17:58.285 - 2004/04/12 11:18:23.395 sds:14:23751 =>
40005050 S 2004/04/12 11:27:35.098 - 2004/04/12 11:27:40.152 sds:1:24414 =>
sds:2:24418
```

Explanation

1. Connection handle value

   The connection handle value that HiRDB manages internally is output in hexadecimal format.

   The value is 8 digits if the client is operating in 32-bit mode or 16 digits if the client is operating in 64-bit mode. The UAP views traces that have the same connection handle value as the same connection.

   In the output example above, `40004250` is output twice as the connection handle value. When viewed from the UAP that uses this connection handle, this information indicates that reconnect processing was executed twice.

2. Reconnect result

   The reconnection result is displayed.

   `S`: Success

   `F`: Failure

3. Reconnect start and end dates and times

   After a disconnection is detected, the dates and times when the reconnection was started and when it ended normally are displayed in milliseconds. If reconnect processing fails, the date and time immediately before control is returned to the UAP is displayed.

4. Connection information before and after reconnect

   Connection information for both before reconnect and after reconnect is displayed. The connection information displays the connection server name, the connection sequence number, and the process ID of the connection server, with the items separated by colons.

If reconnect processing fails, the connection information for after the reconnect is not displayed (becomes blank).

(3) Matching trace information with PRF trace information of Cosminexus

The connection information shown under 4 of the output example is output to the PRF trace information of Cosminexus. If the automatic reconnect facility subsequently executes reconnect processing, match the trace information with PRF trace information as follows.

To match trace information with the PRF trace information of Cosminexus:

1. Get the HiRDB connection information in the PRF trace information.
2. In 4 of the reconnect trace file, search for the connection information obtained in Step 1, and get the corresponding connection handle value.
3. From 1 of the reconnect trace file, track the trace information that has the same connection handle value obtained in Step 2. If the same connection handle value is found, and the connection information before reconnect is the same as the connection information after reconnect for the previous instance of the same connection handle, the connection handle can be used for tracking. If the connection information is different, the connection handle cannot be used for tracking because a new connection (`DISCONNECT`-`CONNECT`) was established with the connection handle.

(4) Backing up reconnect trace information

If the reconnect log file becomes full while reconnect trace information is being output, the reconnect log is output to the other reconnect trace file. In this case, the old reconnect trace information stored in the takeover reconnect trace file is erased and overwritten by new reconnect trace information. Therefore, if the system is being operated for a long period of time, copy the contents of the reconnect trace file and back up the information, as necessary.

To determine the reconnect trace file that is being used currently, check the most recent update dates/times of the files. The reconnect trace file that was updated most recently is the current file.

For a Windows edition HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX edition HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

11.2 UAP error recovery

When an error occurs in a UAP, measures must be taken to prevent the entire HiRDB system from halting. This section explains the following three methods of recovering from UAP errors:

- UAP transaction rollback by HiRDB
- Transaction rollback by UAP instruction
- Memory capacity re-evaluation

The following table shows the UAP error types and recovery methods.

Table 11-5: UAP error types and recovery methods

Error type	Detection method	System action	Recovery method
UAP abnormal termination	UAP processing time Monitoring	Disconnects the UAP	UAP transaction rollback
UAP endless loop			
Transaction incomplete			
UAP processing error	Various error detection at the servers#1	Sends error response to UAP	Transaction rollback by UAP instruction
Error detection and rollback request by UAP	Error detection by UAP	Follows an instruction from UAP	
Deadlock	HiRDB deadlock detection	Sends error response to UAP (implicit rollback)	Termination of UAP transaction
Memory shortage	Error during memory allocation	Disables UAP activation	Reevaluate shared memory and process-specific memory#2

#1: Front-end or back-end server.

#2: Request that the HiRDB system administrator re-evaluate shared memory and process-specific memory.

(1) Monitoring UAP processing time

When a UAP is executed, HiRDB's UAP monitors the processing time to prevent a UAP error from halting the HiRDB processing for an extended period of time.

For time monitoring, a monitoring time must be specified in the `PDSWAITTIME` environment variable during client environment definition; if omitted, the UAP monitors by HiRDB's default monitoring time.

For details about client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

(2) Detecting errors at servers

A HiRDB/Parallel Server returns an error status to the UAP when an error, such as a database processing error, is detected at the front-end server or back-end server while executing SQL statements; steps such as process disconnection must be taken. When a UAP issues a rollback request in response to an error status, HiRDB performs a recovery process.

(3) Detecting UAP errors

When an error is detected in a UAP, a recovery process is started when a `rollback` request is issued. If the UAP was processed normally, the process is disconnected based on a disconnection instruction from the UAP.

(4) Re-evaluating memory capacity

When a shortage occurs in the shared memory or process-specific memory, a message is output indicating the memory or disk space shortage. When such a message is output, enough memory to activate the UAP must be allocated and the UAP must be re-executed.

For details about how to check and, if necessary, revise shared memory and process-specific memory, see the *HiRDB Version 9 Installation and Design Guide* or contact the HiRDB system administrator.

Chapter
12. Command Execution from UAPs

This chapter explains how to execute commands from UAPs.

This chapter contains the following sections:

12.1 Overview
12.2 Preparations for executing commands from COMMAND EXECUTE
12.3 Command executability

12.1 Overview

You execute commands by specifying them in a UAP. The specified commands are executed at the HiRDB server.

The following SQL statements are used to execute commands from a UAP:

- CALL COMMAND statement

  This statement is used to execute HiRDB operation commands and utilities. The CALL COMMAND statement does not require any preparations for command execution.

- COMMAND EXECUTE

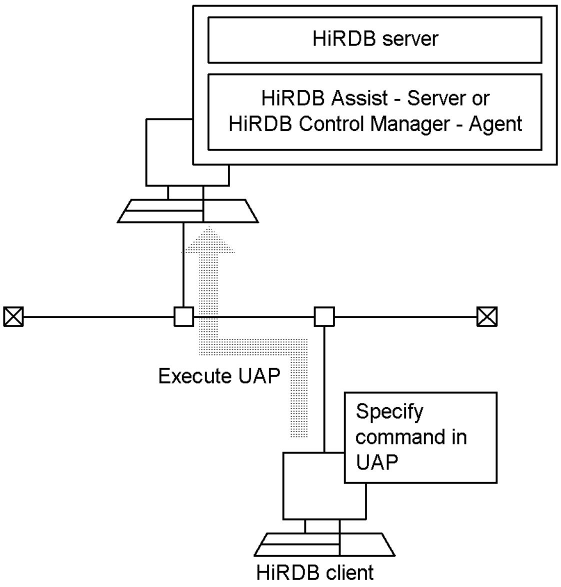
  This statement is used to execute HiRDB operation commands, utilities, and OS commands.

  Because execution of commands from COMMAND EXECUTE is implemented by collaboration between the HiRDB client and HiRDB Control Manager - Agent, HiRDB Control Manager - Agent must be installed on the HiRDB server. For details about HiRDB Control Manager - Agent, see the respective Release Notes.

  Command execution from COMMAND EXECUTE can be used only when the UAP is written in C.

  The following figure provides an overview of command execution from COMMAND EXECUTE.

Figure 12-1: Overview of command execution from COMMAND EXECUTE

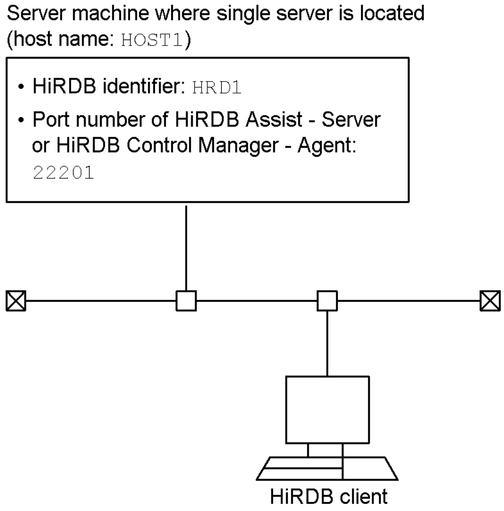


12.2 Preparations for executing commands from COMMAND EXECUTE

(1) HiRDB/Single Server

This section uses a sample UAP that executes data loading (database load utility). The following figure shows a sample server-client configuration for a HiRDB/Single Server.

Figure 12-2: Sample server-client configuration for a HiRDB/Single Server


Server machine where single server is located
(host name: HOST1)
• HiRDB identifier: HRD1
• Port number of HiRDB Assist - Server or HiRDB Control Manager - Agent: 22201
HiRDB client

To execute a data-loading UAP with the server-client configuration shown in *Figure 12-2*, you need to define the following information beforehand:

1. Specify the following client environment definitions:

   `PDSYSTEMID`

   Specifies the HiRDB server's HiRDB identifier (`HRD1`).

   `PDASTHOST`

   Specifies the HiRDB Control Manager - Agent's host name (`HOST1`).

   `PDASTPORT`

   Specifies the HiRDB Control Manager - Agent's port number (`22201`).

2. Prepare the control information file and input data file needed for loading data at the HiRDB server.

3. Suppose that the HiRDB administrator's user name is `USERA` (password: `USERA`) and the owner of the table subject to data loading is `USERB` (password: `USERB`). In this case, specify the following client environment definitions:
   ```
   PDASTUSER=USERA/USERA
   PDUSER=USERB/USERB
   ```

You can now execute the data-loading UAP. For details about each client environment definition, see *6.6.4 Environment definition information*.

The following shows a sample UAP for loading data:

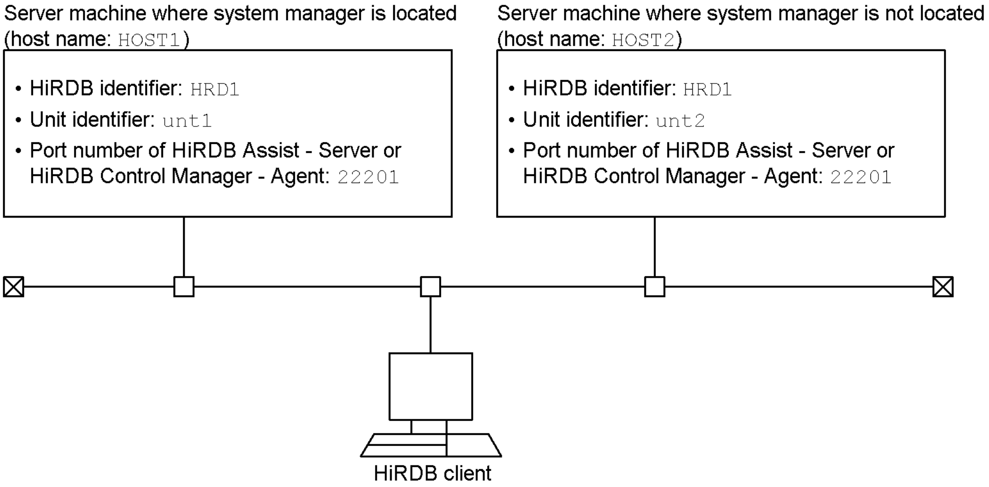
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                   /* CmdLine variable */
long ReturnCode;                   /* variable receiving return
                                          code */
long OutBufLen;                   /* size of area for receiving
                                          execution result */
long CmdRetCode;                      /* variable for receiving
                                      executed command's return
                                          code */
long OutDataLen;                  /* variable for receiving the
                                  length of execution result */
PDOUTBUF OutBuf;                       /* area for receiving
execution
                                          result */
char EnvGroup[256];               /* environment variable group
                                          name variable */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10");  /* specifying execution
command
                                      line command line (RDAREA
                                          shutdown) */
OutBuf = malloc(30000);             /* allocating the execution
                                       result receiving area */
if (OutBuf == NULL){              /* memory allocation error */
printf("Memory allocation error\n");
return ;
}
OutBufLen = 30000 ;                   /* specifying the size of
                                          execution result
                                          receiving area */
EnvGroup[0] = '\0' ;               /* specifying no environment
```

```
                                          variable group */

/* Command execution */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                       /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                 /* if command execution is
                                          normal */

/* Specifying execution command line (to execute dataloading) */
strcpy(CmdLine,"pdload  -i c  -be STOCK c:\HiRDB_S\conf\LOAD");
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                /* if COMMAND EXECUTE terminates
                                          normally */
if (CmdRetCode==0) {                 /* if command execution is
                                          normal */
printf("pdload command successfully\n");
printf("%s\n", OutBuf);
} else {                          /* execution command error */
printf("pdload command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                            /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                          /* execution command error */
printf("pdhold command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
strcpy(CmdLine,"pdrels -r RDDATA10");
                                /* specifying execution command
                                          line (RDAREA shutdown
                                          release) */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                       /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode!=0) {               /* execution command error */
printf("pdrels command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                            /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                                /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
return ;
}
```

(2) HiRDB/Parallel Server

This section uses a sample UAP that executes data loading (database load utility). The following figure shows a sample server-client configuration for a HiRDB/Parallel Server.

Figure 12-3: Sample server-client configuration for a HiRDB/Parallel Server


Server machine where system manager is located (host name: HOST1)
• HiRDB identifier: HRD1
• Unit identifier: unt1
• Port number of HiRDB Assist - Server or HiRDB Control Manager - Agent: 22201
Server machine where system manager is not located (host name: HOST2)
• HiRDB identifier: HRD1
• Unit identifier: unt2
• Port number of HiRDB Assist - Server or HiRDB Control Manager - Agent: 22201
HiRDB client

To execute a data-loading UAP with the server-client configuration shown in *Figure 12-3*, you need to define the following information beforehand:

1. Specify the following client environment definitions:

   PDSYSTEMID

   Specifies the HiRDB server's HiRDB identifier (HRD1).

   PDASTHOST

   Specifies the HiRDB Control Manager - Agent's host name (HOST1). For a HiRDB/Parallel Server, specify the host name of the server machine where the system manager is located.

PDASTPORT

Specifies the HiRDB Control Manager - Agent's port number (22201).

2. Prepare the control information file and input data file needed for loading data at the HiRDB server.
3. Suppose that the HiRDB administrator's user name is USERA (password: USERA) and the owner of the table subject to data loading is USERB (password: USERB). In this case, specify the following client environment definitions:
   PDASTUSER=USERA/USERA
   PDUSER=USERB/USERB

You can now execute the data-loading UAP. For details about each client environment definition, see *6.6.4 Environment definition information*.

The following shows a sample UAP for loading data:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                       /* CmdLine variable */
long ReturnCode;                       /* variable receiving return
                                               code */
long OutBufLen;                       /* size of area for receiving
                                                execution result */
long CmdRetCode;                          /* variable for receiving
                                          executed command's return
                                               code */
long OutDataLen;                      /* variable for receiving the
                                      length of execution result */
PDOUTBUF OutBuf;                          /* area for receiving
execution
                                                result */
char EnvGroup[256];                   /* environment variable group
                                               name variable */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10");  /* specifying execution
command
                                         line (RDAREA shutdown) */
OutBuf = malloc(30000);                /* allocating the execution
                                          result receiving area */
if (OutBuf == NULL){                  /* memory allocation error */
printf("Memory allocation error\n");
return ;
```

```
}
OutBufLen = 30000 ;                   /* specifying the size of
                                     execution result receiving
                                          area */
EnvGroup[0] = '\0' ;                /* specifying no environment
                                          variable group */

/* Command execution */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {     /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                 /* if command execution is
                                          normal */

/* Specifying execution command line (to execute dataloading) */
strcpy(CmdLine,"pdload  -i c  -be STOCK c:\HiRDB_P\conf\LOAD");
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {     /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                 /* if command execution is
                                          normal */
printf("pdload command successfully\n");
printf("%s\n", OutBuf);
} else {                           /* execution command error */
printf("pdload command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                             /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                           /* execution command error */
printf("pdhold command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
strcpy(CmdLine,"pdrels -r RDDATA10");  /* specifying execution
command
                                          line (RDAREA shutdown
                                          release) */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {     /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode!=0) {               /* execution command error */
printf("pdrels command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else{                              /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                             /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
return ;
}
```

12.3 Command executability

Some HiRDB commands can be executed from UAPs and some cannot. The following table shows whether each command is executable from UAPs.

Table 12-1: Command executability from UAPs

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
System operation	`pdadmvr`	Gets the HiRDB version information.	E	E
	`pdcat`	Displays file contents.	E	E
	`pdchgconf`	System reconfiguration command	--	E
	`pdclibsync`	Manipulates C library files.	E	E
	`pdconfchk`	Checks system definitions.	--	E
	`pdcspool`	Deletes troubleshooting information.	E	E
	`pddivinfgt`	Acquires and outputs table partitioning conditions.	--	--
	`pdgeter`	Acquires error information.	E	E
	`pdinfoget`	Acquires error information and estimates its volume.	--	E
	`pditvtrc`	Periodically gets the HiRDB status.	E	E
	`pditvstop`	Stops periodic acquisition of the HiRDB status.	E	E
	`pdjarsync`	Manipulates JAR files.	E	E
	`pdlistls`	Displays list definition information.	E	E
	`pdlodsv`	Reduces the size of the installation directory.	--	--
	`pdls`	Displays HiRDB system status.	E	E
	`pdmemsv`	Saves memory space.	--	--
	`pdntenv`	Sets the HiRDB operation environment.	--	--

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
	`pdobjconv`	Migrates SQL objects into 64-bit-mode HiRDB.	E	--
	`pdopsetup`	Installs an additional HiRDB program product.	--	--
	`pdsetup`	Registers or deletes a HiRDB system in the OS.	--	--
	`pdsvhostname`	Displays the server host name.	--	E
	`pdvrup`	Upgrades HiRDB.	--	E
HiRDB file system	`pdfbkup`	Backs up the HiRDB file system.	E	E
	`pdfls`	Displays HiRDB file system information.	E	E
	`pdfmkfs`	Initializes a HiRDB file system area.	E	E
	`pdfrm`	Deletes a HiRDB file.	E	E
	`pdfrstr`	Restores the HiRDB file system.	E	E
	`pdfstatfs`	Displays the status of a HiRDB file system area.	E	E
	`pdffsck`	Verifies and restores HiRDB file system area integrity.	E	E

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
Log files	`pdlogadpf`	Allocates a log file.	E	E
	`pdlogatul`	Controls the automatic log unloading facility.	--	E
	`pdlogchg`	Changes the status of a log file.	E	E
	`pdlogcls`	Closes a log file.	E	E
	`pdloginit`	Initializes a log file.	E	E
	`pdlogls`	Displays log file information.	E	E
	`pdlogopen`	Opens a log file.	E	E
	`pdlogrm`	Deletes a log file.	E	E
	`pdlogswap`	Swaps log files.	E	E
	`pdlogsync`	Collects a synchronization point dump.	E	E
	`pdlogucat`	Displays unload log file information.	E	E
	`pdlogunld`	Unloads a log file.	E	E
Status files	`pdstscls`	Closes an open status file.	E	E
	`pdstsinit`	Initializes a status file.	E	E
	`pdstsopen`	Opens a status file.	E	E
	`pdstsrm`	Deletes a status file.	E	E
	`pdstsswap`	Swaps status files.	E	E
HiRDB startup and termination	`pdstart`	Starts a HiRDB system, unit, or server.	E	E
	`pdstop`	Terminates a HiRDB system, unit, or server.	E	E

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
Statistics log	`pdstbegin`	Starts output of statistical information.	E	E
	`pdstend`	Stops output of statistical information.	E	E
	`pdstjswap`	Swaps statistics log files.	E	E
	`pdstjsync`	Copies the contents of the statistics log buffer to the statistics log file.	E	E
RDAREAs	`pdclose`	Closes RDAREAs.	E	E
	`pddbls`	Displays the status of RDAREAs.	E	E
	`pdhold`	Shuts down RDAREAs.	E	E
	`pdopen`	Opens RDAREAs.	E	E
	`pdrels`	Releases RDAREAs from shutdown status.	E	E
	`pddbfrz`	Executes frozen update of full HiRDB files in the user LOB RDAREA.	E	E
	`pdrdrefls`	Displays related RD area information.	E	E
Global Buffer	`pdbufls`	Displays global buffer information.	E	E
	`pdbufmod`	Dynamically changes the global buffer.	E	E
Transaction control	`pdcmt`	Commits a transaction.	E	E
	`pdfgt`	Forcibly terminates a transaction.	E	E
	`pdrbk`	Rolls back a transaction.	E	E
	`pdtrndec`	Forcibly completes uncompleted transactions automatically.	--	E

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
Process control	pdcancel	Forcibly terminates UAP and utility processing.	E	E
	pdchprc	Changes the number of server process activations.	E	E
	pdkill	Stops a process forcibly.	--	E
	pdpfresh	Refreshes a server process.	E	E
	pdrpause	Restarts a process server process.	--	E
Update to HiRDB update version	pdprgcopy	Copies the HiRDB update version.	--	E
	pdprgrenew	Updates HiRDB to the update version.	--	E
HiRDB Datareplicator linkage	pdrplstart	Starts HiRDB Datareplicator linkage.	--	E
	pdrplstop	Stops HiRDB Datareplicator linkage.	--	E
Inner replica facility	pddbchg	Switches the replica status of the replica RDAREA.	E	E
Updatable online reorganization	pdorbegin	Commits the database for online reorganization.	E	E
	pdorcheck	Checks the application conditions for online reorganization.	E	E
	pdorchg	Switches the current RDAREA for online reorganization.	E	E
	pdorcreate	Creates a reflection environment for online reorganization.	E	E
	pdorend	Executes reflection of online reorganization.	E	E

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
Security audit	`pdaudbegin`	Starts audit trail acquisition.	E	E
	`pdaudend`	Stops audit trail acquisition.	E	E
	`pdaudrm`	Deletes audit trail files that are shut down.	E	E
	`pdaudswap`	Swaps the current audit trail file.	E	E
	`pdaudatld`	Controls the facility for automatically loading audit trail table data.	E	E
	`pdaudput`	Outputs audit logs for JP1/NETM/Audit.	E	--
Connection security facility	`pdacunlck`	Unlocks the consecutive certification failure account lock state.	--	E
Real Time SAN Replication	`pdrisechk`	Checks the configuration of Real Time SAN Replication.	--	E
	`pdrisedbto`	Inherits the Real Time SAN Replication database.	--	E
	`pdriseset`	Sets the site status of Real Time SAN Replication.	--	E
In-memory data processing	`pdmemdb`	Performs in-memory data processing.	E	E
SQL trace acquisition	`pdclttrc`	Dynamically acquires an SQL trace.	E	E
SQL object information display	`pdobils`	Displays statistical information for an SQL object.	--	E
SQL compilation	`pdcbl`	COBOL preprocessor	--	--
	`pdcpp`	C preprocessor	--	--
	`pdocb`	OOCOBOL preprocessor	--	--
	`pdocc`	C++ preprocessor	--	--

Type	Command	Description	Executability from COMMAND EXECUTE	Executability from CALL COMMAND
Database creation	`pdinit`	Database initialization utility	--	E
	`pddef`	Database definition utility	--	E
	`pdload`	Database load utility	E	E
	`pdparaload`	Parallel loading facility	--	--
	`pdsql`#	Interactive SQL execution utility	--	--
	`pddefrev`	Generates a definition SQL statement.	--	E
Database operations	`pdmod`	Database structure modification utility	E	E
	`pdrorg`	Database reorganization utility	E	E
	`pdexp`	Dictionary import/export utility	--	E
	`pdrbal`	Rebalancing utility	--	E
	`pdreclaim`	Free page release utility	E	E
	`pdpgbfon`	Global buffer residence utility	E	E
	`pdconstck`	Integrity check utility	--	E
Tuning	`pdstedit`	Statistics analysis utility	--	E
	`pddbst`	Database condition analysis utility	E	E
	`pdgetcst`	Optimizing information collection utility	--	E
	`pdvwopt`	Access path display utility	--	E
Database error handling	`pdcopy`	Database copy utility	E	E
	`pdbkupls`	Displays backup file information.	E	E
	`pdrstr`	Database recovery utility	E	E
Plug-in-related	`pdplgrgst`	Registers a plug-in.	--	--
	`pdplgset`	Sets up a plug-in.	--	--
	`pdreginit`	Registry facility initialization utility	E	--

E: Can be executed from UAPs.

--: Cannot be executed from UAPs.

Notes

1. The following commands cannot be used in the UNIX edition:

   `pdkill`, `pdntenv`

2. The following commands cannot be used in the Windows edition:

   `pdgeter`, `pditvtrc`, `pditvstop`, `pdlodsv`, `pdmemsv`, `pdopsetup`, `pdplgset`, `pdrisechk`, `pdrisedbto`, `pdriseset`, `pdrpause`, `pdsetup`

#: This command does not exist in the Windows edition; instead, the HiRDB SQL Executer is used.

Chapter

13. Connection from an XDS Client

This chapter describes the procedure for developing UAPs by connecting from an XDS client.

13.1 Overview of format of connection from an XDS client

13.1 Overview of format of connection from an XDS client

Not supported.

Chapter

14. HiRDB Access from ODBC Application Programs

This chapter explains the OBDC driver installation procedure, ODBC functions, and tuning and troubleshooting procedures that are necessary when ODBC application programs access HiRDB.

The ODBC2.0 driver has become obsolete. It is currently supported in order to maintain compatibility with applications, but this support will be discontinued in the future. Please use the ODBC 3.5 driver.

14.1 ODBC application programs
14.2 Installing the ODBC2.0 driver
14.3 Installing the ODBC 3.5 driver and setting the environment variables
14.4 ODBC functions provided by HiRDB
14.5 ODBC function data types and HiRDB data types
14.6 Specifiability of attributes in the ODBC functions
14.7 Asynchronous execution of ODBC functions
14.8 Setting cursor libraries
14.9 File DSNs
14.10 Executing a UAP in Unicode
14.11 Tuning and troubleshooting
14.12 Facilities that cannot be used when HiRDB is accessed with ODBC
14.13 Notes about using the Linux edition of the HiRDB ODBC 3.5 driver
14.14 Automatic SQL statement generation by using .NET Framework Data Provider for ODBC

14.1 ODBC application programs

Examples of ODBC application programs are Microsoft Access and Microsoft Excel. The ODBC driver must be installed before these application programs can access HiRDB. For information on ODBC driver installation, see *14.2 Installing the ODBC2.0 driver*. You can also access HiRDB via the ODBC driver from a UAP that uses the ODBC functions provided by HiRDB. For information on the ODBC functions provided by HiRDB, see *14.4 ODBC functions provided by HiRDB*.

When the ODBC 3.5 driver is used, you can access HiRDB from a UAP that uses the ODBC 3.*x* interface.

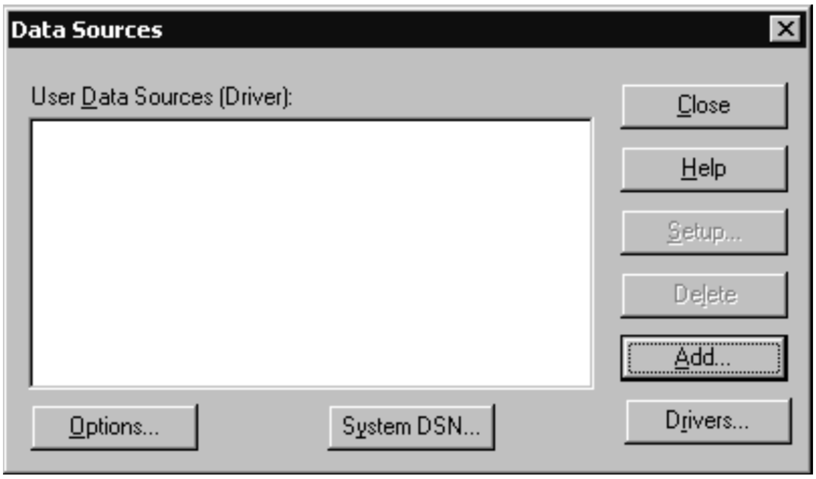
14.2 Installing the ODBC2.0 driver

To run an ODBC application program or a UAP that uses ODBC functions, you need to install the ODBC driver in the HiRDB client beforehand. To execute a UAP via ODBC on the HiRDB server, you also need to install the ODBC driver in the HiRDB server.

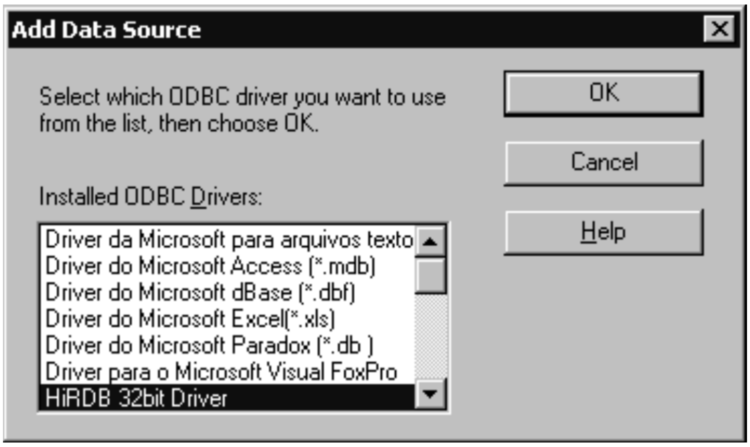
This section presents the ODBC driver installation procedure. Be sure to exit all Windows applications before starting the installation.

To install the ODBC driver:

1. Execute the `ODBC20INST.exe` file that is expanded under the *HiRDB-client-installation-directory*`\utl` directory after the HiRDB client has been installed. The Hitachi self-expanding program starts.
2. In the Hitachi Integrated Installer dialog box, click the **Install** button to start the installer for the ODBC driver.
3. Select the displayed HiRDB driver and choose the **OK** button. Installation does not take place if you choose **OK** without selecting anything.
4. Existing data sources are displayed. If no data source has been defined, nothing is displayed. Choose the **Add** button.



5. Select the HiRDB driver as being subject to data source addition.

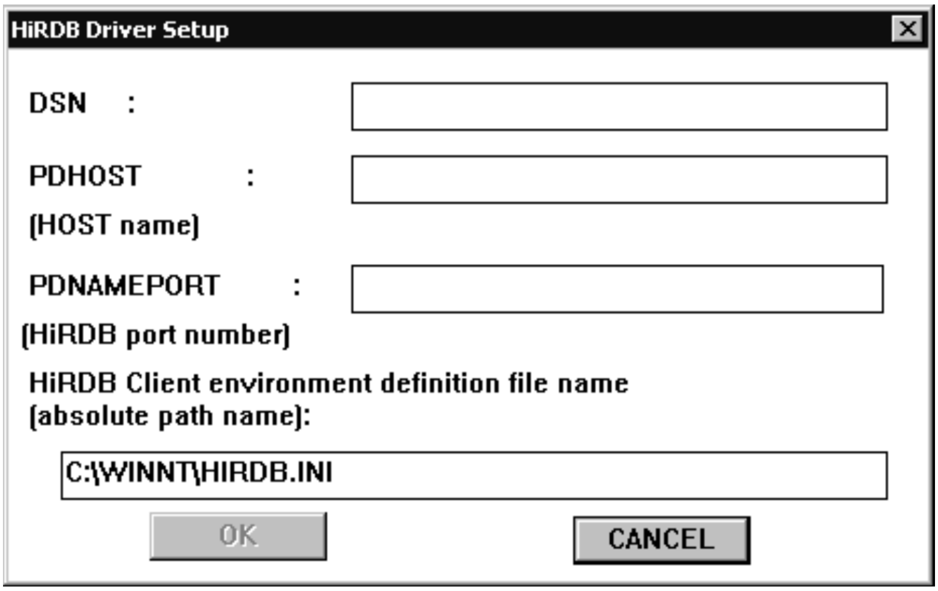

Add Data Source

Select which ODBC driver you want to use from the list, then choose OK.

Installed ODBC Drivers:

- Driver da Microsoft para arquivos texto
- Driver do Microsoft Access (*.mdb)
- Driver do Microsoft dBase (*.dbf)
- Driver do Microsoft Excel(*.xls)
- Driver do Microsoft Paradox (*.db)
- Driver para o Microsoft Visual FoxPro
- HiRDB 32bit Driver

6. A dialog box for setting up the data source is displayed.


HiRDB Driver Setup

DSN :

PDHOST :
(HOST name)

PDNAMEPORT :
(HiRDB port number)

HiRDB Client environment definition file name
(absolute path name):

C:\WINNT\HIRDB.INI

Explanation

Data source name

Specify a name identifying the data source. The name can have up to 32 single-byte characters or 16 double-byte characters. Single-byte and double-byte characters can also be mixed.

PDHOST (host name)

Specify the host name of the server machine. This is the name specified in the client environment definition. For details about PDHOST, see *6.6.4 Environment definition information*. If this information is omitted, the system assumes the value in the client environment definition.

PDNAMEPORT (HiRDB port number)

Specify the port number of the server machine. This is the port number specified in the client environment definition. For details about PDNAMEPORT, see *6.6.4 Environment definition information*. If this information is omitted, the system assumes the value in the client environment definition.

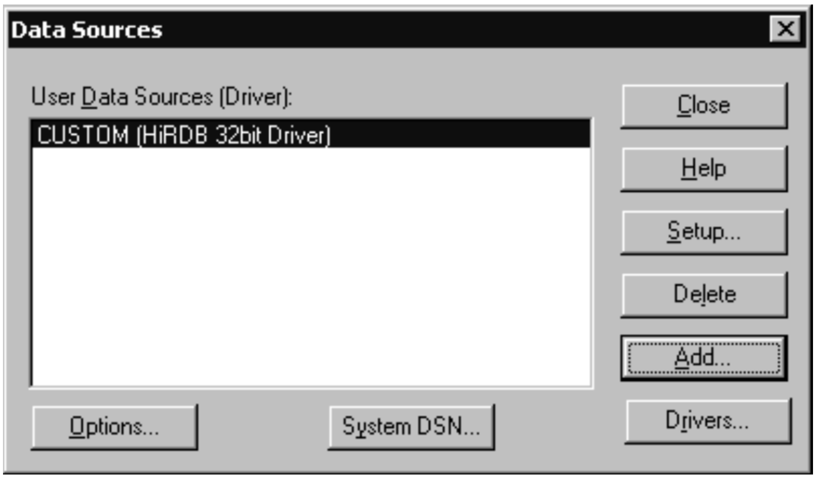
HiRDB client environment definition file name (absolute path name)#

Specify the absolute path name of the HiRDB client environment definition file. Use this item to change the specification values for the HiRDB client environment variables for a particular data source. For example, if you are using the high-speed connection facility (PDSERVICEPORT) to connect to multiple HiRDB systems, you can use this item to specify the file name of the HiRDB client environment definition file and change the connection destination for each data source.

If this information is omitted, the system assumes HIRDB.INI. For all client environment variables, except PDHOST and PDNAMEPORT, the system uses the settings in the HiRDB client environment definition file specified here.

If the specified file is not HIRDB.INI, the system ignores the specifications in HIRDB.INI.

7. After specifying all items, choose the **OK** button. The specified data source is displayed. To change the settings, choose the **Set** button to display the previous dialog box.



#: When you install a HiRDB client, a HiRDB client environment definition file is automatically created under the filename `HiRDB.INI` in the system directory. To install the ODBC driver before installing the HiRDB client, you need to create the `HIRDB.INI` file, because this file has not been created. To create a client environment definition file, copy the `HIRDB.INI` file found in the `odb32\Disk1\Sampleap` directory of the installation CD-ROM to an appropriate directory, and then edit the file. For details about each client environment variable, see *6.6.4 Environment definition information*.

14.3 Installing the ODBC 3.5 driver and setting the environment variables

14.3.1 Installation

(1) Installation directory

The following table shows the ODBC 3.5 driver installation directory.

Table 14-1: ODBC 3.5 driver (32-bit mode) installation directory

Platform	Installation directory
Windows 2000	*Windows-directory*\System32 or *Windows-directory*\SysWOW64
Windows Server 2003	
Windows Server 2008	
Windows XP	
Windows Vista	
Windows 7	
Linux	/*<u>HiRDB</u>*/client/lib/

Note 1

The default Windows directory is C:\WINNT in Windows 2000 and C:\WINDOWS in other platforms.

Note 2

- Underline indicates the HiRDB client installation directory.

Table 14-2: ODBC 3.5 driver (64-bit mode) installation directory

Platform	Installation directory
Windows Server 2003	*Windows-directory*
Windows Server 2008	
Windows XP	
Windows Vista	
Linux	/*<u>HiRDB</u>*/client/lib/

Note 1

The default Windows directory is `C:\WINDOWS`.

Note 2

Underline indicates the HiRDB client installation directory.

(2) Installation flow

The ODBC 3.5 driver installation flow is described as follows.

1. Installing the ODBC 3.5 driver

   Insert the provided medium and follow the installation procedure.

2. Installing the ODBC driver manager

   In the Windows edition, if the ODBC driver manager version is old, install the new ODBC driver manager.

   In the Linux edition, HiRDB does not provide an ODBC driver manager. Separately install an ODBC driver manager that supports ODBC 3.5 API and that can be run on Linux.

3. Setting data sources

   Set data sources.

(3) Installation procedure (in the Windows edition)

(a) Installing the ODBC 3.5 driver

To install the ODBC 3.0 driver:

1. Execute `hcd_inst.exe` found on the integrated CD-ROM to start Hitachi Integrated Installer.

2. At the Hitachi Integrated Installer screen, select one of the following, and then click the **Execute Installation** button to start the HiRDB setup program:

   For Windows edition client products:

   - For HiRDB/Run Time: **HiRDB/Run Time**
   - For HiRDB/Developer's Kit: **HiRDB/Developer's Kit**

   For Windows edition server products:

   - For a HiRDB/Single Server: **HiRDB/Single Server**
   - For a HiRDB/Parallel Server: **HiRDB/Parallel Server**

3. Perform the following operation; the setup program for the selected program process starts:

   For Windows edition client products:

   From the Select Program Process window of the HiRDB setup program, select one of the following, and then click the **Next** button:

- For HiRDB/Run Time: **HiRDB/Run Time**
- For HiRDB/Developer's Kit: **HiRDB/Developer's Kit**

For Windows edition server products:

From the Select Program Process window of the HiRDB setup program, select **HiRDB/Run Time**, and then click the **Next** button.

4. When the Select Installation Destination dialog box appears, change the installation destination as needed and click the **Next** button.
5. In the Select Setup Type dialog box, select **Typical** or **Custom,** and then click the **Next** button.
6. If you have selected **Custom** in step 5, in the Select Component dialog box, select **ODBC3.5 driver**, and then click the **Next** button.
7. The ODBC 3.5 driver is copied under *Windows-directory*`\System32`.
8. The installation procedure is now complete.

(b) Installing the ODBC driver manager (which is included in MDAC2.6RTM)

If the version of the installed ODBC driver manager is old, you must download the most recent MDAC from the Microsoft home page and install it. To determine the version of the ODBC driver manager, start the ODBC Administrator and double-click the **About the ODBC driver manager** tab. If the driver manager version is 3.520.6526.0 or earlier, it is old.

(c) Setting data sources

To set data sources:

1. Start the ODBC Data Source Administrator.
2. Make sure that the tab item is **User DSN** and click the **Add** button.
3. When the Add Data Source dialog box appears, select **HiRDB ODBC3.5 Driver** and click the **Finish** button.
4. When the HiRDB ODBC3.5 Driver Setup dialog box appears, specify the necessary items.

Data source name

Specify a name identifying the data source. The name can have up to 32 single-byte characters or 16 double-byte characters. Single-byte and double-byte characters can also be mixed.

PDHOST (host name)

For a HiRDB/Single Server, specify the host name of the server machine on which the single server is located. For a HiRDB/Parallel Server, specify the host name of the server machine on which the system manager is located.

If this item is omitted, the value specified for PDHOST in the client environment definition is assumed. For details about PDHOST, see *6.6.4 Environment definition information*.

PDNAMEPORT (HiRDB port number)

Specify the port number (the value specified for the pd_name_port operand of the system definition) of the HiRDB server to be accessed.

If this item is omitted, the value specified for PDNAMEPORT in the client environment definition is assumed. For details about PDNAMEPORT, see *6.6.4 Environment definition information*.

HiRDB client environment definition file name

Specify the absolute path name of the HiRDB client environment definition file. Use this item to change the specification values for the HiRDB client environment variables for a particular data source. For example, if you are using the high-speed connection facility (PDSERVICEPORT) to connect to multiple HiRDB systems, you can use this item to specify the file name of the HiRDB client environment definition file and change the connection destination for each data source.

If this item is omitted, HIRDB.INI is assumed.

5. Choosing the **OK** button returns the window to the **User DSN** tab, and the registered data sources are displayed.

- Stopping data source setup

To stop data source setup, from the HiRDB ODBC3.5 Driver Setup dialog box, click the **Cancel** button. When the **Cancel** button is clicked, no data source is registered.

- Deleting a data source

To delete a data source:

1. From the Data Source dialog box, select the name of the data source to be deleted.
2. Click the **Delete** button to delete the data source.

(4) Installation procedure (in the Linux edition)

(a) Installing the ODBC 3.5 driver

Start the Hitachi Program Product Installer and install the ODBC 3.5 driver.

(b) Installing the ODBC driver manager

The Linux edition of HiRDB does not provide an ODBC driver manager. You must install it separately.

(c) Registering the ODBC driver information

You must edit the `odbcinst.ini` file in order to register the ODBC driver information.

This subsection describes an example that uses unixODBC as the driver manager.

The `odbcinst.ini` file is located in `/usr/local/etc`. The following example edits the `odbcinst.ini` file for using the HiRDB ODBC 3.5 driver for the 32-bit-mode edition of Linux.

Example

```
[HiRDBOdbcDriver]  ............................................1
Driver = /HiRDB/client/lib/libodbcdrv.so  ......................2
```

1. Driver name

   Square brackets ([ ]) enclose the driver name that corresponds to the data source. You can specify any name.

2. Driver

   Specify the absolute path of the HiRDB ODBC 3.5 driver for the Linux edition. In 64-bit mode, it is `libodbcdrv64.so`.

(d) Setting data sources

Edit the `odbc.ini` file.

This subsection describes an example that uses unixODBC as the driver manager.

The `odbc.ini` file is located in `/usr/local/etc`. It also exists as a hidden file named `.odbc.ini` in the home directory. These two files correspond to the system DSN and the user DSN in Windows. If you edit the file as user DSN, place the `.odbc.ini` file in the home directory of the user who executes applications.

The following example edits the `odbc.ini` file.

Example

```
[HiRDB_LIN30] ............................................1
Driver = HiRDBOdbcDriver ..................................2
PDHOST = 10.209.34.223 ....................................3
PDNAMEPORT = 22200 ........................................4
INIFLNAME = /usr/local/etc/HiRDB.ini ......................5
```

1. Data source name

   Specify a name identifying the data source. This name must observe the rules for the ODBC driver manager being used.

2. Driver

   Specify the driver name specified in the `odbcinst.ini` file when you registered the driver.

3. PDHOST

   For a HiRDB/Single Server, specify the host name of the server machine on which the single server is located. For a HiRDB/Parallel Server, specify the host name of the server machine on which the system manager is located.

   If this item is omitted, the value specified for `PDHOST` in the client environment definition is assumed. For details about `PDHOST`, see *6.6.4 Environment definition information*.

4. PDNAMEPORT

   Specify the port number (value specified in the `pd_name_port` operand in the system definition) of the HiRDB server to be accessed.

   If this item is omitted, the value specified for `PDNAMEPORT` in the client environment definition is assumed. For details about `PDNAMEPORT`, see *6.6.4 Environment definition information*.

5. INIFLNAME

   Specify the absolute path name of the HiRDB client environment definition file. You use this item to change specification values for HiRDB client environment variables for a particular data source. For example, if you are using the high-speed connection facility (`PDSERVICEPORT`) to connect to multiple HiRDB systems, you can use this item to specify the file name of the HiRDB client environment definition file and change the connection destination for each data source.

   If this item is omitted, the values specified in the environment definition are assumed.

   In the UNIX edition, the HiRDB client environment definition file (`HiRDB.ini`) is not created. Therefore, you must provide the file and then specify it in this item.

14.3.2 Setting the environment variables (in the Windows edition)

Set up the following environment variables:

`PATH=`*Windows-directory*`;`*Windows-directory*`\System32`

Note 1

The default Windows directories are as follows:

- Windows 2000: `C:\WINNT`
- Windows Server, Windows XP, Windows Vista, and Windows 7: `C:\WINDOWS`

Note 2

Set as system environment variables.

14.3.3 Determining the version number of the ODBC 3.5 driver

To determine the version number of the ODBC driver, start the ODBC Data Source Administrator and select the **Driver** tab. For the storage location of **ODBC Data Source Administrator**, see *14.3.1(1) Installation directory*.

Note:

In the ODBC data source administrator, the ODBC 3.5 driver is registered under the name `HiRDB ODBC3.0 Driver`, but its functions comply with ODBC 3.5.

14.4 ODBC functions provided by HiRDB

HiRDB provides ODBC functions, and you can access HiRDB on a server from a UAP that utilizes these ODBC functions. The following table shows the ODBC functions provided by HiRDB.

Table 14-3: ODBC functions provided by HiRDB

Classification	ODBC functions	ODBC2.0 driver		ODBC 3.5 driver	
		Provided?	Expansion level	Provided?	Expansion level
Connection to data source	`SQLAllocEnv`	Y	Core	--	--
	`SQLAllocHandle`	--	--	Y	Core
	`SQLAllocConnect`	Y	Core	--	--
	`SQLConnect`	Y	Core	Y	Core
	`SQLDriverConnect`	Y	1	Y	Core
	`SQLBrowseConnect`	Y	2	Y	1
Driver and data source information acquisition	`SQLDataSources`	Y[#1]	2	Y[#1]	Core
	`SQLDrivers`	--	--	Y[#1]	Core
	`SQLGetInfo`	Y	1	Y	Core
	`SQLGetFunctions`	--	--	Y	Core
	`SQLGetTypeInfo`	Y	1	Y	Core
Driver option setting and acquisition	`SQLSetConnectOption`	Y	1	--	--
	`SQLGetConnectOption`	Y	1	--	--
	`SQLSetStmtOption`	Y	1	--	--
	`SQLGetStmtOption`	Y	1	--	--
	`SQLSetConnectAttr`	--	--	Y	Core
	`SQLGetConnectAttr`	--	--	Y	Core
	`SQLSetEnvAttr`	--	--	Y	Core
	`SQLGetEnvAttr`	--	--	Y	Core
	`SQLSetStmtAttr`	--	1	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC 3.5 driver	
		Provided?	Expansion level	Provided?	Expansion level
	`SQLGetStmtAttr`	--	1	Y	Core
Descriptor value setup	`SQLGetDescField`	--	--	Y	Core
	`SQLGetDescRec`	--	--	Y	Core
	`SQLSetDescField`	--	--	Y	Core
	`SQLSetDescRec`	--	--	Y	Core
	`SQLCopyDesc`	--	--	Y	Core
SQL request creation	`SQLAllocStmt`	Y	Core	--	--
	`SQLPrepare`	Y	Core	Y	Core
	`SQLBindParameter`	Y	11	Y	Core
	`SQLSetParam`#2	Y	1	--	--
	`SQLGetCursorName`	Y	Core	Y	Core
	`SQLSetCursorName`	Y	Core	Y	Core
	`SQLDescribeParam`	Y	2	--	--
	`SQLNumParam`	Y	2	--	--
	`SQLDescribeParams`	--	--	Y	2
	`SQLNumParams`	--	--	Y	Core
	`SQLParamOptions`	N	2	--	--
	`SQLSetScrollOptions`	N#3	2	N	2
SQL execution	`SQLExecute`	Y	Core	Y	Core
	`SQLExecDirect`	Y	Core	Y	Core
	`SQLNativeSql`	Y	2	Y	Core
	`SQLParamData`	Y	1	Y	Core
	`SQLPutData`	Y	1	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC 3.5 driver	
		Provided?	Expansion level	Provided?	Expansion level
Execution result and execution result information acquisition	`SQLRowCount`	Y	Core	Y	Core
	`SQLNumResultCols`	Y	Core	Y	Core
	`SQLDescribeCol`	Y	Core	Y	Core
	`SQLColAttributes`	Y	Core	--	--
	`SQLColAttribute`	--	--	Y	Core
	`SQLBindCol`	Y	Core	Y	Core
	`SQLFetch`	Y	Core	Y	Core
	`SQLFetchScroll`	--	--	Y#4	Core
	`SQLExtendedFetch`	N#3	2	Y	Core
	`SQLGetData`	Y	1	Y	Core
	`SQLSetPos`	N#3	2	Y#4	1
	`SQLBulkOperations`	--	--	N	1
	`SQLMoreResults`#5	Y	2	Y	1
	`SQLError`	Y	Core	--	--
	`SQLGetDiagField`	--	--	Y	Core
	`SQLGetDiagRec`	--	--	Y	Core
Data source system information acquisition	`SQLColumnPrivileges`	Y	2	Y	2
	`SQLColumns`	Y	1	Y	Core
	`SQLForeignKeys`	Y#6	2	Y#6	2
	`SQLPrimaryKeys`	Y#6	2	Y#6	1
	`SQLProcedureColumns`	Y	2	Y	1
	`SQLProcedure`	Y	2	Y	1
	`SQLSpecialColumns`	Y#6	1	Y#6	Core
	`SQLStatistics`	Y	1	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC 3.5 driver	
		Provided?	Expansion level	Provided?	Expansion level
	`SQLTablePrivileges`	Y	2	Y	2
	`SQLTables`	Y	1	Y	Core
SQL execution termination	`SQLFreestmt`	Y	Core	Y	Core
	`SQLCloseCursor`	--	--	Y	Core
	`SQLCancel`	Y	Core	Y	Core
	`SQLTransact`	Y	Core	Y	Core
	`SQLEndTran`	--	--	Y	Core
Disconnection	`SQLDisconnect`	Y	Core	Y	Core
	`SQLFreeConnect`	Y	Core	--	--
	`SQLFreeEnv`	Y	Core	--	--
	`SQLFreeHandle`	--	--	Y	Core

Legend:

Y: The applicable ODBC function is provided.

N: The applicable ODBC function is not provided.

--: Not applicable

1: Level 1

2: Level 2

Core: Core level

#1

This function is provided by the drive manager.

#2

Although the `SQLSetParam` function was included in `SQLBindParameter` beginning with ODBC 2.0, this function is provided to maintain compatibility with applications that do not support ODBC 2.0.

#3

Because this function is installed in the ODBC2.0 cursor library, the range of functions specified by the cursor library can be used. To use

SQLExtendedFetch, set up a cursor library. For details on setting up a cursor library, see *14.8 Setting cursor libraries*.

#4

To use these ODBC functions, you must use the cursor library provided by Microsoft.

#5

The statement handle used to execute an SQL statement is used to call SQLMoreResults. If the executed SQL statement uses HiRDB's results-set return facility and a result set is available, the function returns SQL_SUCCESS and the next result set becomes available. If the next result set is not available, the function returns SQL_NO_DATA.

For details about the results-set return facility, see the manual *HiRDB Version 9 SQL Reference*.

#6

Only the call is supported. The result set created by this function always contains no rows.

14.5 ODBC function data types and HiRDB data types

The table below shows the correspondence between ODBC function data types and server HiRDB data types.

ODBC function data type refers to an SQL data type that is specified in an argument of an ODBC function.

Table 14-4: ODBC function data types and HiRDB data types

Classification	ODBC data type	HiRDB data type	Description	Availability
Character data	`SQL_CHAR`	`CHAR(`*n*`)`	Fixed-length character string	U
	`SQL_VARCHAR`	`VARCHAR(`*n*`)`	Variable-length character string	U
	`SQL_LONGVARCHAR`	`VARCHAR(`*n*`)`	Variable-length character string	U
	`SQL_CHAR`	`NCHAR(`*n*`)`	Fixed-length national character string `NATIONAL CHARACTER(`*n*`)`	U
	`SQL_VARCHAR`	`NVARCHAR(`*n*`)`	Variable-length national character string	U
	`SQL_CHAR`	`MCHAR(`*n*`)`	Fixed-length mixed character string	U
	`SQL_VARCHAR`	`MVARCHAR(`*n*`)`	Variable-length mixed character string	U

Classification	ODBC data type	HiRDB data type	Description	Availability
Numeric data	SQL_DECIMAL	DEC[IMAL] (*p*,*s*)	Fixed-point number Precision (total number of digits) = *p*, Scale (number of digits below the decimal point) = *s* $1 \leq p \leq 15, 0 \leq s \leq p$	U
	SQL_NUMERIC	--		NU
	SQL_SMALLINT	SMALLINT	Integer from -32,768 to 32,767	U
	SQL_INTEGER	INTEGER	Integer from -2,147,483,648 to 2,147,483,647	U
	SQL_TINYINT	--	Integer from -256 to 255	NU
	SQL_BIGINT	--	1-digit sign and 19-digit integer	NU
	SQL_REAL	SMALLFLT,REAL	Single-precision floating-point number	U
	SQL_FLOAT	FLOAT, DOUBLE PRECISION	Double-precision floating-point number	U
	SQL_DOUBLE	FLOAT, DOUBLE PRECISION	Double-precision floating-point number	U
	SQL_BIT	--	Bit	NU
	SQL_BINARY	--	Fixed-length binary data	NU
	SQL_LONGVARBINARY	BINARY(n)	Variable-length binary data	U
	SQL_LONGVARBINARY	BLOB	Variable-length binary data	U
Date and time data	SQL_TYPE_DATE	DATE	Date	U
	SQL_TYPE_TIMESTAMP	TIMESTAMP	Date/time	U
	SQL_TYPE_TIME	TIME	Time	U
	--#	INTERVAL YEAR TO DAY	Date interval	NU
	SQL_INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR TO SECOND	Time interval	U

Classification	ODBC data type	HiRDB data type	Description	Availability
User-defined type	--	Abstract data type	Abstract data type	NU

--: Data type not available in ODBC.

U: Can be used.

NU: Cannot be used.

#: Database data types in the server are reported without change.

Note

For details about the maximum character string lengths and value ranges for the various data types, see the manual *HiRDB Version 9 SQL Reference*.

(1) Facilities available to ODBC functions

When a UAP uses ODBC functions to access the HiRDB system in the server, not all HiRDB facilities are available to the UAP. The following table lists the facilities that can be used by such a UAP.

Table 14-5: Available facilities

Facility	Availability
Obtaining special column information	--
Obtaining index information	U
Using date and time data types	U#1
Using repetition columns	NU#3
Using array columns	--
Obtaining table and column headers	--
Asynchronous processing	NU
Using the escape character for the `LIKE` predicate	U
Obtaining an updated row count	U
Setting the timeout value for logging in	NU
Using Japanese data types	U#2
Executing definition SQL statements	U

U: Can be used.

NU: Cannot be used.

--: Not a DBMS function

#1: The INTERVAL YEAR TO DAY data type cannot be used.

#2: The database data types are reported without change.

#3: A repetition column can be accessed if it has a simple structure without repeated ? parameters.

Example

Column C1 of table T1 is a repetition column.

```
SELECT C1[1],C1[2] FROM T1             A
SELECT C1 FROM T1                      --
INSERT INTO T1 VALUES(ARRAY[?,?])      A
INSERT INTO T1 VALUES(?)               --
```

A: Can be accessed

--: Cannot be accessed

(2) Setting update and deletion operations that use cursors

The SQLGetCursorName function obtains the user cursor name that was set with the SQLSetCursorName function. If no cursor name has been set, the SQLGetCursorName function cannot obtain a system-defined cursor. Therefore, set an appropriate user cursor name to update or delete an item with a cursor.

(3) Setting driver options

The options that can be set with the SQLSetConnectOption or SQLGetConnectOption function are limited. The following table shows the options that can be set.

Table 14-6: Options that can be set with the SQLSetConnectOption and SQLGetConnectOption functions

Option	Setting
SQL_ACCESS_MODE	SQL_MODE_READ_WRITE
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF or SQL_AUTOCOMMIT_ON
SQL_LOGIN_TIMEOUT	--
SQL_TRANSLATE_DLL	--
SQL_TRANSLATE_OPTION	--
SQL_TXN_ISOLATION	--

--: Cannot be set

14.6 Specifiability of attributes in the ODBC functions

(1) SQLSetConnectAttr

The following table lists the ODBC connection attributes that can be specified in `SQLSetConnectAttr`.

Table 14-7: ODBC connection attributes that can be specified in SQLSetConnectAttr

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_ACCESS_MODE	Y	Core	--
SQL_ATTR_ASYNC_ENABLE	Y	Level 1	--
SQL_ATTR_AUTO_IPD	N	Level 2	--
SQL_ATTR_AUTO_COMMIT	Y	Level 1	--
SQL_ATTR_CONNECTION_DEAD	N	Level 1	--
SQL_ATTR_CONNECTION_TIMEOUT	Y	Level 2	Only `0` can be specified. If any other value is specified, an error results.
SQL_ATTR_CURRENT_CATALOG	N	Level 2	--
SQL_ATTR_LOGIN_TIMEOUT	Y	Level 2	Only `0` can be specified. If any other value is specified, `0` is set.
SQL_ATTR_METADATA_ID	N	Core	--
SQL_ATTR_ODBC_CURSORS	N	Core	--
SQL_ATTR_PACKET_SIZE	N	Level 2	--
SQL_ATTR_QUIET_MODE	N	Core	--
SQL_ATTR_TRACE	Y	Core	--
SQL_ATTR_TRACEFILE	Y	Core	--
SQL_ATTR_TRANSLATE_LIB	N	Core	--
SQL_ATTR_TRANSLATE_OPTION	N	Core	--
SQL_ATTR_ANSI_APP	Y	Undefined	--
SQL_ATTR_TXN_ISOLATION	Y	Level 1	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_ENLIST_IN_DTC	Y	Undefined	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(2) SQLGetConnectAttr

The following table lists the ODBC connection attributes that can be specified in `SQLGetConnectAttr`.

Table 14-8: ODBC connection attributes that can be specified in SQLGetConnectAttr

Attribute	Whether specifiabl e	Compliance level	Remarks
SQL_ATTR_ACCESS_MODE	Y	Core	--
SQL_ATTR_ASYNC_ENABLE	Y	Level 1	--
SQL_ATTR_AUTO_IPD	Y	Level 2	--
SQL_ATTR_AUTO_COMMIT	Y	Level 1	--
SQL_ATTR_CONNECTION_DEAD	Y	Level 1	--
SQL_ATTR_CONNECTION_TIMEOUT	N	Level 2	--
SQL_ATTR_CURRENT_CATALOG	N	Level 2	--
SQL_ATTR_LOGIN_TIMEOUT	N	Level 2	--
SQL_ATTR_METADATA_ID	Y	Core	--
SQL_ATTR_ODBC_CURSORS	N	Core	--
SQL_ATTR_PACKET_SIZE	N	Level 2	--
SQL_ATTR_QUIET_MODE	N	Core	--
SQL_ATTR_TRACE	Y	Core	Returned by the driver manager
SQL_ATTR_TRACEFILE	Y	Core	Returned by the driver manager

Attribute	Whether specifiabl e	Compliance level	Remarks
SQL_ATTR_TRANSLATE_LIB	N	Core	--
SQL_ATTR_TRANSLATE_OPTION	N	Core	--
SQL_ATTR_ANSI_APP	Y	Undefined	--
SQL_ATTR_TXN_ISOLATION	Y	Level 1	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(3) SQLSetDescField

The following table lists the ODBC descriptor attributes that can be specified in `SQLSetDescField`.

Table 14-9: ODBC descriptor attributes that can be specified in SQLSetDescField

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_ALLOC_TYPE	N	Core	--
SQL_DESC_ARRAY_SIZE	Y	Core	--
SQL_DESC_ARRAY_STATUS_PTR	Y	Core	--
SQL_DESC_BIND_OFFSET_PTR	Y	Core	--
SQL_DESC_DESC_BIND_TYPE	Y	Core	--
SQL_DESC_COUNT	Y	Core	--
SQL_DESC_ROWS_PROCESSED_PTR	Y	Core	--
SQL_DESC_AUTO_UNIQUE_VALUE	N	Level 2	--
SQL_DESC_BASE_COLUMN_NAME	N	Core	--
SQL_DESC_BASE_TABLE_NAME	N	Level 1	--
SQL_DESC_CASE_SENSITIVE	N	Core	--
SQL_DESC_CATALOG_NAME	N	Level 2	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_CONCISE_TYPE	Y	Core	--
SQL_DESC_DATA_PTR	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_CODE	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_PRECISION	Y	Core	--
SQL_DESC_SQL_DESC_DISPLAY_SIZE	N	Core	--
SQL_DESC_FIXED_PREC_SCALE	N	Core	--
SQL_DESC_INDICATOR_PTR	Y	Core	--
SQL_DESC_LABEL	N	Level 2	--
SQL_DESC_LENGTH	Y	Core	--
SQL_DESC_LITERAL_PREFIX	N	Core	--
SQL_DESC_LITERAL_SUFFIX	N	Core	--
SQL_DESC_LOCAL_TYPE_NAME	N	Core	--
SQL_DESC_NAME	Y	Core	--
SQL_DESC_NULLABLE	N	Core	--
SQL_DESC_NUM_PREC_RADIX	Y	Undefined	--
SQL_DESC_OCTET_LENGTH	Y	Core	--
SQL_DESC_OCTET_LENGTH_PTR	Y	Core	--
SQL_DESC_PARAMETER_TYPE	Y	Core	--
SQL_DESC_PRECISION	Y	Core	--
SQL_DESC_ROWVER	N	Level 1	--
SQL_DESC_SCALE	Y	Core	--
SQL_DESC_SCHEMA_NAME	N	Level 1	--
SQL_DESC_SEARCHABLE	N	Core	--
SQL_DESC_TABLE_NAME	N	Level 1	--
SQL_DESC_TYPE	Y	Core	--
SQL_DESC_TYPE_NAME	N	Core	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_UNNAMED	Y	Core	--
SQL_DESC_UNSIGNED	N	Core	--
SQL_DESC_UPDATABLE	N	Core	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(4) SQLGetDescField

The following table lists the ODBC descriptor attributes that can be specified in `SQLGetDescField`.

Table 14-10: ODBC descriptor attributes that can be specified in SQLGetDescField

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_ALLOC_TYPE	Y	Core	--
SQL_DESC_ARRAY_SIZE	Y	Core	--
SQL_DESC_ARRAY_STATUS_PTR	Y	Core	--
SQL_DESC_BIND_OFFSET_PTR	Y	Core	--
SQL_DESC_DESC_BIND_TYPE	Y	Core	--
SQL_DESC_COUNT	Y	Core	--
SQL_DESC_ROWS_PROCESSED_PTR	Y	Core	--
SQL_DESC_AUTO_UNIQUE_VALUE	Y	Level 2	--
SQL_DESC_BASE_COLUMN_NAME	Y	Core	--
SQL_DESC_BASE_TABLE_NAME	Y	Level 1	--
SQL_DESC_CASE_SENSITIVE	Y	Core	--
SQL_DESC_CATALOG_NAME	Y	Level 2	--
SQL_DESC_CONCISE_TYPE	Y	Core	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_DATA_PTR	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_CODE	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_PRECISION	Y	Core	--
SQL_DESC_SQL_DESC_DISPLAY_SIZE	Y	Core	--
SQL_DESC_FIXED_PREC_SCALE	Y	Core	--
SQL_DESC_INDICATOR_PTR	Y	Core	--
SQL_DESC_LABEL	Y	Level 2	--
SQL_DESC_LENGTH	Y	Core	--
SQL_DESC_LITERAL_PREFIX	Y	Core	--
SQL_DESC_LITERAL_SUFFIX	Y	Core	--
SQL_DESC_LOCAL_TYPE_NAME	Y	Core	--
SQL_DESC_NAME	Y	Core	--
SQL_DESC_NULLABLE	Y	Core	--
SQL_DESC_NUM_PREC_RADIX	Y	Undefined	--
SQL_DESC_OCTET_LENGTH	Y	Core	--
SQL_DESC_OCTET_LENGTH_PTR	Y	Core	--
SQL_DESC_PARAMETER_TYPE	Y	Core	--
SQL_DESC_PRECISION	Y	Core	--
SQL_DESC_ROWVER	N	Level 1	--
SQL_DESC_SCALE	Y	Core	--
SQL_DESC_SCHEMA_NAME	Y	Level 1	--
SQL_DESC_SEARCHABLE	Y	Core	--
SQL_DESC_TABLE_NAME	Y	Level 1	--
SQL_DESC_TYPE	Y	Core	--
SQL_DESC_TYPE_NAME	Y	Core	--
SQL_DESC_UNNAMED	Y	Core	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_UNSIGNED	Y	Core	--
SQL_DESC_UPDATABLE	Y	Core	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(5) SQLSetEnvAttr

The following table lists the ODBC environment attributes that can be specified in `SQLSetEnvAttr`.

Table 14-11: ODBC environment attributes that can be specified in SQLSetEnvAttr

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_CONNECTION_POOLING	Y	Undefined	Not included in the conformance level
SQL_ATTR_CP_MATCH	Y	Undefined	Not included in the conformance level
SQL_ATTR_ODBC_VERSION	Y	Core	--
SQL_ATTR_OUTPUT_NTS	Y	Undefined	Not included in the conformance level

Legend:

Y: Can be specified

--: Not applicable

(6) SQLGetEnvAttr

The following table lists the ODBC environment attributes that can be specified in `SQLGetEnvAttr`.

Table 14-12: ODBC environment attributes that can be specified in SQLGetEnvAttr

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_CONNECTION_POOLING	Y	Undefined	Not included in the conformance level
SQL_ATTR_CP_MATCH	Y	Undefined	Not included in the conformance level
SQL_ATTR_ODBC_VERSION	Y	Core	--
SQL_ATTR_OUTPUT_NTS	Y	Undefined	Not included in the conformance level

Legend:

Y: Can be specified

--: Not applicable

(7) SQLSetStmtAttr

The following table lists the ODBC statement attributes that can be specified in `SQLSetStmtAttr`.

Table 14-13: ODBC statement attributes that can be specified in SQLSetStmtAttr

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_APP_PARAM_DESC	Y	Core	--
SQL_ATTR_APP_ROW_DESC	Y	Core	--
SQL_ATTR_ASYNC_ENABLE	Y	Level 1	--
SQL_ATTR_CONCURRENCY	Y	Level 2	--
SQL_ATTR_CURSOR_SCROLLABLE	Y	Level 1	--
SQL_ATTR_CURSOR_SENSITIVITY	Y	Level 2	--
SQL_ATTR_CURSOR_TYPE	Y	Level 2	Only `SQL_CURSOR_FORWARD_ONLY` can be specified. If any other value is specified, `SQL_CURSOR_FORWARD_ONLY` is set.

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_ENABLE_AUTO_IPD	Y	Level 2	--
SQL_ATTR_FETCH_BOOKMARK_PTR	N	Level 2	--
SQL_ATTR_IMP_PARAM_DESC	N	Core	--
SQL_ATTR_IMP_ROW_DESC	N	Core	--
SQL_ATTR_KEYSET_SIZE	N	Level 2	--
SQL_ATTR_MAX_LENGTH	Y	Level 1	--
SQL_ATTR_MAX_ROWS	Y	Level 1	--
SQL_ATTR_METADATA_ID	Y	Core	--
SQL_ATTR_NOSCAN	Y	Core	--
SQL_ATTR_PARAM_BIND_OFFSET_PTR	N	Core	--
SQL_ATTR_PARAM_BIND_TYPE	Y	Core	--
SQL_ATTR_PARAM_OPERATION_PTR	N	Core	--
SQL_ATTR_PARAM_STATUS_PTR	Y	Core	--
SQL_ATTR_PARAMS_PROCESSED_PTR	Y	Core	--
SQL_ATTR_PARAMSET_SIZE	Y	Core	--
SQL_ATTR_QUERY_TIMEOUT	N	Level 2	--
SQL_ATTR_RETRIEVE_DATA	Y	Level 1	--
SQL_ATTR_ROW_ARRAY_SIZE	Y	Core	--
SQL_ATTR_ROW_BIND_OFFSET_PTR	Y	Core	--
SQL_ATTR_ROW_BIND_TYPE	Y	Core	--
SQL_ATTR_ROW_NUMBER	N	Level 1	--
SQL_ATTR_ROW_OPERATION_PTR	Y	Level 1	--
SQL_ATTR_ROW_STATUS_PTR	Y	Core	--
SQL_ATTR_ROWS_FETCHED_PTR	Y	Core	--
SQL_ATTR_SIMULATE_CURSOR	N	Level 2	--
SQL_ATTR_USE_BOOKMARKS	N	Level 2	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(8) SQLGetStmtAttr

The following table lists the ODBC statement attributes that can be specified in `SQLGetStmtAttr`.

Table 14-14: ODBC statement attributes that can be specified in SQLGetStmtAttr

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_APP_PARAM_DESC	Y	Core	--
SQL_ATTR_APP_ROW_DESC	Y	Core	--
SQL_ATTR_ASYNC_ENABLE	Y	Level 1	--
SQL_ATTR_CONCURRENCY	Y	Level 2	--
SQL_ATTR_CURSOR_SCROLLABLE	Y	Level 1	--
SQL_ATTR_CURSOR_SENSITIVITY	Y	Level 2	--
SQL_ATTR_CURSOR_TYPE	Y	Level 2	--
SQL_ATTR_ENABLE_AUTO_IPD	Y	Level 2	--
SQL_ATTR_FETCH_BOOKMARK_PTR	N	Level 2	--
SQL_ATTR_IMP_PARAM_DESC	Y	Core	--
SQL_ATTR_IMP_ROW_DESC	Y	Core	--
SQL_ATTR_KEYSET_SIZE	N	Level 2	--
SQL_ATTR_MAX_LENGTH	Y	Level 1	--
SQL_ATTR_MAX_ROWS	Y	Level 1	--
SQL_ATTR_METADATA_ID	Y	Core	--
SQL_ATTR_NOSCAN	Y	Core	--
SQL_ATTR_PARAM_BIND_OFFSET_PTR	N	Core	--
SQL_ATTR_PARAM_BIND_TYPE	N	Core	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_ATTR_PARAM_OPERATION_PTR	Y	Core	--
SQL_ATTR_PARAM_STATUS_PTR	Y	Core	--
SQL_ATTR_PARAMS_PROCESSED_PTR	Y	Core	--
SQL_ATTR_PARAMSET_SIZE	Y	Core	--
SQL_ATTR_QUERY_TIMEOUT	N	Level 2	--
SQL_ATTR_RETRIEVE_DATA	Y	Level 1	--
SQL_ATTR_ROW_ARRAY_SIZE	Y	Core	--
SQL_ATTR_ROW_BIND_OFFSET_PTR	N	Core	--
SQL_ATTR_ROW_BIND_TYPE	N	Core	--
SQL_ATTR_ROW_NUMBER	N	Level 1	--
SQL_ATTR_ROW_OPERATION_PTR	Y	Level 1	--
SQL_ATTR_ROW_STATUS_PTR	Y	Core	--
SQL_ATTR_ROWS_FETCHED_PTR	Y	Core	--
SQL_ATTR_SIMULATE_CURSOR	N	Level 2	--
SQL_ATTR_USE_BOOKMARKS	N	Level 2	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(9) SQLGetInfo

This subsection shows the information types that can be specified in `SQLGetInfo`.

(a) Driver information

The `SQLGetInfo` function returns information about the ODBC driver, such as the number of active statements, data source name, and interface standard conformance level. The following table lists the driver information items that can be specified in `SQLGetInfo`.

Table 14-15: Driver information items that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_ACTIVE_ENVIRONMENTS	Y	--
SQL_ASYNC_MODE	Y	--
SQL_BATCH_ROW_COUNT	Y	--
SQL_BATCH_SUPPORT	Y	--
SQL_DATA_SOURCE_NAME	Y	--
SQL_DRIVER_HDBC	Y	--
SQL_DRIVER_HDESC	Y	Returned by the driver manager
SQL_DRIVER_HENV	Y	--
SQL_DRIVER_HLIB	Y	Returned by the driver manager
SQL_DRIVER_HSTMT	Y	--
SQL_DRIVER_NAME	Y	--
SQL_DRIVER_ODBC_VER	Y	--
SQL_DRIVER_VER	Y	--
SQL_DYNAMIC_CURSOR_ATTRIBUTES1	Y	--
SQL_DYNAMIC_CURSOR_ATTRIBUTES2	Y	--
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	Y	--
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	Y	--
SQL_FILE_USAGE	Y	--
SQL_GETDATA_EXTENSIONS	Y	--
SQL_INFO_SCHEMA_VIEWS	Y	--
SQL_KEYSET_CURSOR_ATTRIBUTES1	Y	--
SQL_KEYSET_CURSOR_ATTRIBUTES2	Y	--
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	Y	--
SQL_MAX_CONCURRENT_ACTIVITIES	Y	--

Option value	Whether specifiable	Remarks
SQL_MAX_DRIVER_CONNECTIONS	Y	--
SQL_ODBC_INTERFACE_CONFORMANCE	Y	--
SQL_ODBC_VER	Y	--
SQL_PARAM_ARRAY_ROW_COUNTS	Y	--
SQL_PARAM_ARRAY_SELECTS	Y	--
SQL_ROW_UPDATES	Y	--
SQL_SEARCH_PATTERN_ESCAPE	Y	--
SQL_SERVER_NAME	Y	--
SQL_STATIC_CURSOR_ATTRIBUTES1	Y	--
SQL_STATIC_CURSOR_ATTRIBUTES2	Y	--

Legend:

Y: Can be specified

--: Not applicable

(b) Information about DBMS product

The SQLGetInfo function returns information about the DBMS product, such as the DBMS product name and version. The following table lists the DBMS product information items that can be specified in SQLGetInfo.

Table 14-16: DBMS product information items that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_DATABASE_NAME	Y	--
SQL_DBMS_NAME	Y	--
SQL_DBMS_VER	Y	--

Legend:

Y: Can be specified

--: Not applicable

(c) Data source information

The SQLGetInfo function returns information about the data source, such as the cursor attributes and transaction functions. The following table lists the data source information items that can be specified in SQLGetInfo.

Table 14-17: Data source information items that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_ACCESSIBLE_PROCEDURES	Y	--
SQL_ACCESSIBLE_TABLES	Y	--
SQL_BOOKMARK_PERSISTENCE	Y	--
SQL_CATALOG_TERM	Y	--
SQL_COLLATION_SEQ	Y	--
SQL_CONCAT_NULL_BEHAVIOR	Y	--
SQL_CURSOR_COMMIT_BEHAVIOR	Y	--
SQL_CURSOR_ROLLBACK_BEHAVIOR	Y	--
SQL_CURSOR_SENSITIVITY	Y	--
SQL_DATA_SOURCE_READ_ONLY	Y	--
SQL_DEFAULT_TXN_ISOLATION	Y	--
SQL_DESCRIBE_PARAMETER	Y	--
SQL_MULT_RESULT_SETS	Y	--
SQL_MULTIPLE_ACTIVE_TXN	Y	--
SQL_NEED_LONG_DATA_LEN	Y	--
SQL_NULL_COLLATION	Y	--
SQL_PROCEDURE_TERM	Y	--
SQL_SCHEMA_TERM	Y	--
SQL_SCROLL_OPTIONS	Y	--
SQL_TABLE_TERM	Y	--
SQL_TXN_CAPABLE	Y	--
SQL_TXN_ISOLATION_OPTION	Y	--
SQL_USER_NAME	Y	--

Legend:

Y: Can be specified

--: Not applicable

(d) SQL statement information

The SQLGetInfo function returns information about the SQL statements that are supported by the data source. The following table lists the SQL statement information items that can be specified in SQLGetInfo.

Table 14-18: SQL statement information items that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_AGGREGATE_FUNCTIONS	Y	--
SQL_ALTER_DOMAIN	Y	--
SQL_ALTER_SCHEMA	N	--
SQL_ALTER_TABLE	Y	--
SQL_ANSI_SQL_DATETIME_LITERALS	Y	--
SQL_CATALOG_LOCATION	Y	--
SQL_CATALOG_NAME	Y	--
SQL_CATALOG_NAME_SEPARATOR	Y	--
SQL_CATALOG_USAGE	Y	--
SQL_COLUMN_ALIAS	Y	--
SQL_CORRELATION_NAME	Y	--
SQL_CREATE_ASSERTION	Y	--
SQL_CREATE_CHARACTER_SET	Y	--
SQL_CREATE_COLLATION	Y	--
SQL_CREATE_DOMAIN	Y	--
SQL_CREATE_SCHEMA	Y	--
SQL_CREATE_TABLE	Y	--
SQL_CREATE_TRANSLATION	Y	--
SQL_DDL_INDEX	Y	--

Option value	Whether specifiable	Remarks
SQL_DROP_ASSERTION	Y	--
SQL_DROP_CHARACTER_SET	Y	--
SQL_DROP_COLLATION	Y	--
SQL_DROP_DOMAIN	Y	--
SQL_DROP_SCHEMA	Y	--
SQL_DROP_TABLE	Y	--
SQL_DROP_TRANSLATION	Y	--
SQL_DROP_VIEW	Y	--
SQL_EXPRESSIONS_IN_ORDERBY	Y	--
SQL_GROUP_BY	Y	--
SQL_IDENTIFIER_CASE	Y	--
SQL_IDENTIFIER_QUOTE_CHAR	Y	--
SQL_INDEX_KEYWORDS	Y	--
SQL_INSERT_STATEMENT	Y	--
SQL_INTEGRITY	Y	--
SQL_KEYWORDS	Y	--
SQL_LIKE_ESCAPE_CLAUSE	Y	--
SQL_NON_NULLABLE_COLUMNS	Y	--
SQL_SQL_CONFORMANCE	Y	--
SQL_OJ_CAPABILITIES	Y	--
SQL_ORDER_BY_COLUMNS_IN_SELECT	Y	--
SQL_OUTER_JOINS	Y	--
SQL_PROCEDURES	Y	--
SQL_QUOTED_IDENTIFIER_CASE	Y	--
SQL_SCHEMA_USAGE	Y	--
SQL_SPECIAL_CHARACTERS	Y	--

Option value	Whether specifiable	Remarks
SQL_SUBQUERIES	Y	--
SQL_UNION	Y	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(e) Information about limitations to SQL statements

The SQLGetInfo function returns information about the limitations to SQL statement identifiers and clauses, such as the maximum length of an identifier and the maximum number of columns in a list. The following table lists information items related to SQL statement limitations that can be specified in SQLGetInfo.

Table 14-19: Information items related to SQL statement limitations that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_MAX_BINARY_LITERAL_LEN	Y	--
SQL_MAX_CATALOG_NAME_LEN	Y	--
SQL_MAX_CHAR_LITERAL_LEN	Y	--
SQL_MAX_COLUMN_NAME_LEN	Y	--
SQL_MAX_COLUMNS_IN_GROUP_BY	Y	--
SQL_MAX_COLUMNS_IN_INDEX	Y	--
SQL_MAX_COLUMNS_IN_ORDER_BY	Y	--
SQL_MAX_COLUMNS_IN_SELECT	Y	--
SQL_MAX_COLUMNS_IN_TABLE	Y	--
SQL_MAX_CURSOR_NAME_LEN	Y	--
SQL_MAX_IDENTIFIER_LEN	Y	--
SQL_MAX_INDEX_SIZE	Y	--
SQL_MAX_PROCEDURE_NAME_LEN	Y	--

Option value	Whether specifiable	Remarks
SQL_MAX_ROW_SIZE	Y	--
SQL_MAX_ROW_SIZE_INCLUDES_LONG	Y	--
SQL_MAX_SCHEMA_NAME_LEN	Y	--
SQL_MAX_STATEMENT_LEN	Y	--
SQL_MAX_TABLE_NAME_LEN	Y	--
SQL_MAX_TABLES_IN_SELECT	Y	--
SQL_MAX_USER_NAME_LEN	Y	--

Legend:

Y: Can be specified

--: Not applicable

(f) Scalar function information

The SQLGetInfo function returns information about the scalar functions supported by the data source or driver. The following table lists the scalar function information items that can be specified in SQLGetInfo.

Table 14-20: Scalar function information items that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_CONVERT_FUNCTIONS	Y	--
SQL_NUMERIC_FUNCTIONS	Y	--
SQL_STRING_FUNCTIONS	Y	--
SQL_SYSTEM_FUNCTIONS	Y	--
SQL_TIMEDATE_ADD_INTERVALS	Y	--
SQL_TIMEDATE_DIFF_INTERVALS	Y	--
SQL_TIMEDATE_FUNCTIONS	Y	--

Legend:

Y: Can be specified

--: Not applicable

(g) Information about the target SQL data type

The SQLGetInfo function returns information about the target SQL data type when an SQL data type for which a data source is specified is converted by the CONVERT scalar function. The following table lists information items related to the target SQL data type that can be specified in SQLGetInfo.

Table 14-21: Information items related to the target SQL data type that can be specified in SQLGetInfo

Option value	Whether specifiable	Remarks
SQL_CONVERT_BIGINT	Y	--
SQL_CONVERT_BINARY	Y	--
SQL_CONVERT_BIT	Y	--
SQL_CONVERT_CHAR	Y	--
SQL_CONVERT_DATE	Y	--
SQL_CONVERT_DECIMAL	Y	--
SQL_CONVERT_DOUBLE	Y	--
SQL_CONVERT_FLOAT	Y	--
SQL_CONVERT_INTEGER	Y	--
SQL_CONVERT_INTERVAL_YEAR_MONTH	Y	--
SQL_CONVERT_INTERVAL_DAY_TIME	Y	--
SQL_CONVERT_LONGVARBINARY	Y	--
SQL_CONVERT_LONGVARCHAR	Y	--
SQL_CONVERT_NUMERIC	Y	--
SQL_CONVERT_REAL	Y	--
SQL_CONVERT_SMALLINT	Y	--
SQL_CONVERT_TIME	Y	--
SQL_CONVERT_TIMESTAMP	Y	--
SQL_CONVERT_TINYINT	Y	--
SQL_CONVERT_VARBINARY	Y	--
SQL_CONVERT_VARCHAR	Y	--

Legend:

Y: Can be specified

--: Not applicable

(h) Information types added for ODBC 3.0 or later

The following table lists the information types that have been added for ODBC 3.0 or later.

Table 14-22: Information types added for ODBC 3.0 or later

Option value	Whether specifiable	Remarks
SQL_ACTIVE_ENVIRONMENTS	Y	--
SQL_AGGREGATE_FUNCTIONS	Y	--
SQL_ALTER_DOMAIN	Y	--
SQL_ALTER_SCHEMA	N	--
SQL_ANSI_SQL_DATETIME_LITERALS	Y	--
SQL_ASYNC_MODE	Y	--
SQL_BATCH_ROW_COUNT	Y	--
SQL_BATCH_SUPPORT	Y	--
SQL_CATALOG_NAME	Y	--
SQL_COLLATION_SEQ	Y	--
SQL_CONVERT_INTERVAL_YEAR_MONTH	Y	--
SQL_CONVERT_INTERVAL_DAY_TIME	Y	--
SQL_CREATE_ASSERTION	Y	--
SQL_CREATE_CHARACTER_SET	Y	--
SQL_CREATE_COLLATION	Y	--
SQL_CREATE_DOMAIN	Y	--
SQL_CREATE_SCHEMA	Y	--
SQL_CREATE_TABLE	Y	--
SQL_CREATE_TRANSLATION	Y	--
SQL_CURSOR_SENSITIVITY	Y	--

Option value	Whether specifiable	Remarks
SQL_DDL_INDEX	Y	--
SQL_DESCRIBE_PARAMETER	Y	--
SQL_DM_VER	Y	Returned by the driver manager
SQL_DRIVER_HDESC	Y	--
SQL_DROP_ASSERTION	Y	--
SQL_DROP_CHARACTER_SET	Y	--
SQL_DROP_COLLATION	Y	--
SQL_DROP_DOMAIN	Y	--
SQL_DROP_SCHEMA	Y	--
SQL_DROP_TABLE	Y	--
SQL_DROP_TRANSLATION	Y	--
SQL_DROP_VIEW	Y	--
SQL_DYNAMIC_CURSOR_ATTRIBUTES1	Y	--
SQL_DYNAMIC_CURSOR_ATTRIBUTES2	Y	--
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	Y	--
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	Y	--
SQL_INFO_SCHEMA_VIEWS	Y	--
SQL_INSERT_STATEMENT	Y	--
SQL_KEYSET_CURSOR_ATTRIBUTES1	Y	--
SQL_KEYSET_CURSOR_ATTRIBUTES2	Y	--
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	Y	--
SQL_MAX_IDENTIFIER_LEN	Y	--
SQL_PARAM_ARRAY_ROW_COUNTS	Y	--
SQL_PARAM_ARRAY_SELECTS	Y	--
SQL_STATIC_CURSOR_ATTRIBUTES1	Y	--
SQL_STATIC_CURSOR_ATTRIBUTES2	Y	--

Option value	Whether specifiable	Remarks
SQL_SQL92_DATETIME_FUNCTIONS	Y	--
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	Y	--
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	Y	--
SQL_SQL92_GRANT	Y	--
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	Y	--
SQL_SQL92_PREDICATES	Y	--
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	Y	--
SQL_SQL92_REVOKE	Y	--
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	Y	--
SQL_SQL92_STRING_FUNCTIONS	Y	--
SQL_SQL92_VALUE_EXPRESSIONS	Y	--
SQL_STANDARD_CLI_CONFORMANCE	Y	--
SQL_XOPEN_CLI_YEAR	Y	--

Legend:

Y: Can be specified

N: Cannot be specified

--: Not applicable

(10) SQLColAttribute

The following table lists the ODBC descriptor attributes that can be specified in `SQLColAttribute`.

Table 14-23: ODBC descriptor attributes that can be specified in SQLColAttribute

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_AUTO_UNIQUE_VALUE	Y	Level 2	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_BASE_COLUMN_NAME	Y	Core	The column name that can be acquired by the DESCRIBE SQL statement, not the base column name, is returned. Therefore, the same column name as in SQL_DESC_NAME is returned.
SQL_DESC_BASE_TABLE_NAME	Y	Level 1	--
SQL_DESC_CASE_SENSITIVE	Y	Core	--
SQL_DESC_CATALOG_NAME	Y	Level 2	--
SQL_DESC_CONCISE_TYPE	Y	Core	--
SQL_DESC_DATA_PTR	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_CODE	Y	Core	--
SQL_DESC_DATETIME_INTERVAL_PRECISION	Y	Core	--
SQL_DESC_SQL_DESC_DISPLAY_SIZE	Y	Core	--
SQL_DESC_FIXED_PREC_SCALE	Y	Core	--
SQL_DESC_INDICATOR_PTR	Y	Core	--
SQL_DESC_LABEL	Y	Level 2	--
SQL_DESC_LENGTH	Y	Core	--
SQL_DESC_LITERAL_PREFIX	Y	Core	--
SQL_DESC_LITERAL_SUFFIX	Y	Core	--
SQL_DESC_LOCAL_TYPE_NAME	Y	Core	--
SQL_DESC_NAME	Y	Core	--
SQL_DESC_NULLABLE	Y	Core	--
SQL_DESC_NUM_PREC_RADIX	Y	--	--
SQL_DESC_OCTET_LENGTH	Y	Core	--
SQL_DESC_OCTET_LENGTH_PTR	Y	Core	--

Attribute	Whether specifiable	Compliance level	Remarks
SQL_DESC_PARAMETER_TYPE	Y	Core	--
SQL_DESC_PRECISION	Y	Core	--
SQL_DESC_SCALE	Y	Core	--
SQL_DESC_SCHEMA_NAME	Y	Level 1	--
SQL_DESC_SEARCHABLE	Y	Core	--
SQL_DESC_TABLE_NAME	Y	Level 1	--
SQL_DESC_TYPE	Y	Core	--
SQL_DESC_TYPE_NAME	Y	Core	--
SQL_DESC_UNNAMED	Y	Core	--
SQL_DESC_UNSIGNED	Y	Core	--
SQL_DESC_UPDATABLE	Y	Core	--

Legend:

Y: Can be specified

--: Not applicable

14.7 Asynchronous execution of ODBC functions

(1) About asynchronous execution of ODBC functions

When an ODBC application program accesses HiRDB, the program can execute the ODBC functions asynchronously.

When ODBC functions are executed simultaneously, the ODBC driver does not return control to the application until function calling ends. However, when ODBC functions are executed asynchronously, the ODBC driver can return control to the application program at any time. The application program can therefore execute other processes when the ODBC functions are being executed asynchronously.

The following ODBC functions can be executed asynchronously:

- SQLColumnPrivileges
- SQLColumns
- SQLExecute
- SQLExecDirect
- SQLParamData
- SQLProcedureColumns
- SQLFetch
- SQLStatistics
- SQLTablePrivileges
- SQLTables
- SQLProcedures

(2) Procedure for asynchronous execution of ODBC functions

To execute asynchronous ODBC functions:

1. To enable asynchronous execution in a specific hstmt (statement handle) only, use the SQL_ASYNC_ENABLE option to call SQLSetStmtOption.#1 To enable asynchronous execution in all hstmt handles related to hdbc (connection handle), use the SQL_ASYNC_ENABLE option to call SQLSetConnectOption.#2
2. When an ODBC function that can be executed asynchronously#1 is called with an hstmt for which asynchronous execution has been enabled, the ODBC driver starts asynchronous execution of that function and returns SQL_STILL_EXECUTING. (If asynchronous execution is not set or if an error occurs, the ODBC driver returns a synchronous execution code, such as

SQL_SUCCESS or SQL_ERROR.)

3. The application program can execute another process while an ODBC function is being executed asynchronously. An application program can call only the SQLAllocStmt, SQLCancel, and SQLGetFunctions with the hstmt that is executing the function asynchronously. If any other function is called (except the function being executed asynchronously), the driver manager returns a sequence error.

4. The application program calls the ODBC function that was being executed asynchronously to check whether execution of that function terminated. If the function is still executing, SQL_STILL_EXECUTING is returned. If the process has terminated, a return code such as SQL_SUCCESS or SQL_ERROR is returned.

   When an application program calls a function to check the execution status, all specified arguments, except hstmt, are ignored. (However, the specified argument values must be effective; otherwise, an error can occur if an incorrect address or value is specified.) For example, if SQLExecDirect is executed asynchronously with the INSERT statement function, and SQLExecDirect is called again, the execution status of the INSERT statement is returned, even if the UPDATE statement is specified.

Note

To disable asynchronous execution in a specific hstmt only, use the SQL_ASYNC_ENABLE option to call SQLSetStmtOption. To disable asynchronous execution in all hstmt handles related to hdbc, use the SQL_ASYNC_ENABLE option to call SQLSetConnectOption.

#1: The settings for SQLSetStmtOption are shown as follows.

Option	Setting
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF or SQL_ASYNC_ENABLE_ON
SQL_BIND_TYPE	Cannot be set.
SQL_MAX_LENGTH	Limit specified by server or value specified by user
SQL_NOSCAN (Default=FALSE)	SQL_NOSCAN_OFF or SQL_NOSCAN_ON
SQL_QUERRY_TIMEOUT	Cannot be set.
SQL_MAX_ROWS	Limit specified by server or value specified by user

#2: The settings for SQLSetConnectOption are shown as follows.

Option	Setting
SQL_ACCESS_MODE	Fixed to SQL_MODE_READ_WRITE

Option	Setting
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF or SQL_AUTOCOMMIT_ON
SQL_LOGON_TIMEOUT	Cannot be set.
SQL_OPT_TRACE	Fixed to 0 (Off). This option is returned from the ODBC driver manager.
SQL_OPT_TRACEFILE	Fixed to NULL. This option is returned from the ODBC driver manager.
SQL_TRANSLATE_DLL	Cannot be set.
SQL_TRANSLATE_OPTION	Cannot be set.
SQL_TXN_ISOLATION	SQL_TXN_READ_UNCOMMITED
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF or SQL_ASYNC_ENABLE_ON

(3) Cancelling asynchronous execution for an ODBC function

(a) Cancelling asynchronous execution of an ODBC function

To cancel an ODBC function during asynchronous execution, call SQLCancel.

SQLCancel issues a process cancellation request to the server as soon as it confirms that the specified hstmt is currently undergoing asynchronous execution.

The return value for SQLCancel only reports whether the cancel request was completed. To find out whether asynchronous execution of the function was actually cancelled, call the function that was being processed asynchronously and check the return value. If the function is still executing, SQL_STILL_EXECUTING is returned. If cancel processing was completed, SQL_ERROR and SQLSTATE S1008 (process cancellation) are returned. If the function has already terminated normally, or if an error occurred, a code such as SQL_SUCCESS or SQL_ERROR is returned.

(b) Cancelling asynchronous execution in a multi-thread application program

A multi-thread application program can cancel an ODBC function that is being executed asynchronously with hstmt. To cancel the function, the application program calls SQLCancel from a different thread and uses the same hstmt as that used by the function being cancelled.

The return value of SQLCancel indicates whether the driver received the request correctly. The return values of the original function are SQL_SUCCESS, or SQL_ERROR and SQLSTATE S1008 (process cancellation).

Note

The HiRDB cancel process is executed for an individual connection, and the connection with the server is forcibly disconnected. (The server outputs `KFPS00993-I: Server process termination REQUEST=clt_attention`). Consequently, all statements of the `hstmt` handlers related to the specified `hstmt` are cancelled (the transaction is rolled back). Carefully consider any data being updated before cancelling an ODBC function that is being executed asynchronously.

(4) Coding example

The following is an example of coding for asynchronous execution:

```
SQLSetStmtOption(hstmst, SQL_ASYNC_ENABLE,
SQL_ASYNC_ENABLE_ON);
         ...
 Retrieval processing with  SQLFetch
rc=SQLFetch(hstmt);
while(rc==SQL_STILL_EXECUTING)
{
         ...
 Continue processing of UAP being executed asynchronously
         ...
  if(process cancel request was issued)
  {
        rc=SQL_Cancel(hstmt);
        if(rc==SQL_ERROR){  To error processing for cancel
                            request failure  }
  }
 rc=SQLFetch(hstmt);
}
if(rc == SQL_ERROR){  To error processing  }
 To retrieval data manipulation processing
       ...
```

14.8 Setting cursor libraries

A cursor library must be set before SQLExtendedFetch can be used in an ODBC UAP. A cursor library can be set in two ways:

When the SetConnectOption ODBC function is used:

Use the SetConnectOption ODBC function and specify SQL_ODBC_CURSORS in the fOption argument and SQL_CUR_USE_ODBC in the vParam argument.

When RDO of Visual Basic is used:

Specify rdUseOdbc in the CursorDriver property of the rdoEnvironment object. The following is an example of coding when RDO of Visual Basic is used:

```
Dim mrdoEnv as rdoEnvironment
Set mrdoEnv = rdoEngine.rdoCreateEnvironment("","","")
    mrdoEnv.CursorDriver = rdUseOdbc
    Src = "DSN=host1; UID=USER_A;PWD=USER_A"
    Set mrdoConn = mrdoEnv.OpenConnection
                   ("", rdDriverComplete, False, Src)
        ...
        ...
```

See the simple sample UAPs found in the Sampleap directory of the installation floppy disk for the ODBC driver.

14.9 File DSNs

When an application program uses a file DSN, it can connect to a data source without obtaining information from `ODBC.INI` or the registry because the DSN file stores information for connecting to the data source.

By sharing the file, multiple users can connect to the HiRDB system without having to register the data source (formerly the machine data source) to each machine. A file DSN can be used when the ODBC component version is 3.0 or higher.

File DSNs can be created by the ODBC data source administrator.

Creating file DSNs

To create a file DSN, select a file DSN, add the file DSN, select a driver (HiRDB 32-bit driver), and then specify the storage file name. A connection request is then issued to the HiRDB system, and the driver manager creates the file based on the complete connection character string returned by `SQLDriverConnect`. However, in this case, the password is not stored in the file DSN. If the password is to be shared, add the line `PWD=`*password* to the created file.

14.10 Executing a UAP in Unicode

This section explains the ODBC functions that can be used by a UAP in Unicode.

(1) ODBC functions that can be used by a UAP in Unicode

The following table shows the ODBC functions that can be used by a UAP in Unicode.

Table 14-24: ODBC functions that can be used by a UAP in Unicode

Classification	Function name	Function
Connection with data source	`SQLConnectW`	Connects to a specific driver based on the data source name, authorization identifier, and password.
	`SQLDriverConnectW`	Connects to a specific driver based on the connection character string. Also, requests to the driver manager and driver that a connection dialog box be displayed for the user.
	`SQLBrowseConnectW`	Returns the continuous level connection attributes and valid attribute values. If a value is specified for each connection attribute, connects to the data source.
	`SQLDriversW`	Returns the installed driver and a list of its attributes.
Driver and data source information	`SQLDataSources`	Returns a list of data sources that can be used.
	`SQLGetInfoW`	Returns a specific driver and data source information.
Setting and acquisition of driver options	`SQLSetConnectAttrW`	Sets the connection attributes.
	`SQLGetConnectAttrW`	Returns the connection attribute values.
	`SQLSetStmtAttrW`	Sets the statement attribute.
	`SQLGetStmtAttrW`	Returns the statement attribute value.
Descriptor setting and acquisition	`SQLSetDescFieldW`	Sets one descriptor field.
	`SQLGetDescFieldW`	Returns one descriptor field value.
	`SQLSetDescRecW`	Sets multiple descriptor fields.

Classification	Function name	Function
	SQLGetDescRecW	Returns multiple descriptor field values.
	SQLPrepareW	Prepares an SQL statement to be executed later.
SQL request creation	SQLSetCursorNameW	Specifies a cursor name.
	SQLGetCursorNameW	Returns the cursor name related to the statement handle.
SQL execution	SQLExecDirectW	Executes a statement.
	SQLNativeSqlW	Returns the text of the SQL statement that the driver converted.
Acquisition of execution results and execution results information	SQLDescribeColW	Describes the results set columns.
	SQLColAttributeW	Describes the attributes of the results set columns.
	SQLGetDiagFieldW	Returns additional diagnosis information (one field of the diagnosis data structure).
	SQLGetDiagRecW	Returns additional diagnosis information (multiple fields of the diagnosis data structure).
	SQLColumnPrivilegesW	Returns a list of columns and privileges related to one or more tables.
Acquisition of data source system information	SQLColumnsW	Returns a list of column names of specified tables.
	SQLForeignKeysW	Returns a list of column names that compose an external key when there is an external key in a specified table.
	SQLPrimaryKeysW	Returns a list of column names that compose a main key of a specified table.
	SQLProcedureColumnsW	Returns a list of input or output parameters and columns that compose the results set of a specified procedure.
	SQLProceduresW	Returns a list of procedure names in a specified data source.

Classification	Function name	Function
	`SQLSpecialColumnsW`	Returns the optimum column for identifying lines in a specified table or the column information that is corrected automatically when line values are changed by a transaction.
	`SQLStatisticsW`	Returns statistical information related to a single table and a list of indexes related to the table.
	`SQLTablePrivilegesW`	Returns a list of tables and the privileges related to each table.
	`SQLTablesW`	Returns a list of table names in the specified data source.

(2) Notes

The following notes apply when `UCS2_UJIS` or `UCS2_UTF8` is set in `PDCLTCNVMODE` of the client environment definition:

- The SQL data type returned when the column attribute is acquired is as follows:

  When the HiRDB data type is `CHAR`, `MCHAR`, or `NCHAR`: `SQL_WCHAR`

  When the HiRDB data type is `VARCHAR`, `MVARCHAR`, or `NVARCHAR`: `SQL_WVARCHAR`

- When the column attribute is acquired, if the HiRDB data type is character string system data type, the column definition length x 2 is set for the column length. For example, in case of `char(10), 20` is returned for the column length.

14.11 Tuning and troubleshooting

This section explains how to tune and troubleshoot ODBC UAPs.

(1) Poor performance in a UAP that retrieves multiple rows

Use the block transfer facility. To use this facility, specify the PDBLKF operand in the client environment definition. A specification value between 40 and 50 is recommended. Specifying a larger value has little effect in reducing the number of communications and instead may delay processing because of the increased processing overhead. For details about the block transfer facility, see *4.7 Block transfer facility*.

(2) If a UAP executes connect and disconnect processing frequently

Use the high-speed connection facility. To use this facility, specify the PDFESHOST, PDSERVICEPORT, and PDSERVICEGRP operands in the client environment definition. The high-speed connection facility shortens the time for connection to HiRDB. For details about the PDFESHOST, PDSERVICEPORT, and PDSERVICEGRP operands of the client environment definition, see *6.6.4 Environment definition information*.

(3) Checking SQL statements requested of HiRDB

If a UAP accesses the HiRDB system via ODBC, the SQL statements specified in the UAP may differ from the SQL statements requested of the HiRDB system, depending on the environment in which the UAP was created. To check what kind of SQL statements are issued to the HiRDB system, use the SQL trace facility. To use this facility, specify the PDSQLTRACE operand in the client environment definition. It is recommended to also specify the trace output destination directory in the PDCLTPATH operand at this time. For details about the SQL trace facility, see *11.1.1 SQL tracing*.

(4) Other

- If an application program, such as Microsoft Access, specifies the lock option in a retrieval SQL statement, a syntax error may occur in that application program. If this happens, examine whether the problem can be corrected by specifying the PDISLLVL operand of the client environment definition.
- If you use Microsoft Access to access HiRDB, a lock error might occur in HiRDB during updating, depending on how the UAP is written. This occurs when Microsoft Access establishes multiple connections to HiRDB, and referencing or updating is executed on the same line from different connections. To avoid this, specify 0 or 1 for the PDISLLVL operand in the client environment definition. In the Sampleap directory on the installation floppy disk for the ODBC driver, there is a sample UAP that uses DAO (Data Access Object) of Visual Basic and no lock error occurs during access; refer to this UAP.

14.12 Facilities that cannot be used when HiRDB is accessed with ODBC

When an application program accesses the HiRDB system with ODBC, some of the facilities cannot be used.

- Access using the row interface

  Queries with `ROW` specifications, `UPDATE` statements, and `INSERT` statements cannot be executed.

- Update and deletion using a cursor

  Update and deletion using `CURRENT OF` *cursor-name* cannot be executed. However, if the cursor library facility is used, the cursor library can sometimes execute such operations to change `CURRENT OF` *cursor-name* to a `WHERE` condition.

- Portable cursors

  Portable cursors (cursors with the `WITH HOLD` specification or cursors defined by queries with the `UNTIL DISCONNECT` specification) cannot be used.

14.13 Notes about using the Linux edition of the HiRDB ODBC 3.5 driver

- The Linux edition of the HiRDB ODBC 3.5 driver processes one character of the SQL_C_WCHAR type as two bytes in the UTF-16LE format. Therefore, when you pass data of the SQL_C_WCHAR type to a Linux edition of the HiRDB ODBC 3.5 driver, make sure that the data is in UTF-16LE format.
- When an ODBC function supporting Unicode is used, the Linux edition of the HiRDB ODBC 3.5 driver checks the character codes of the connection-target HiRDB server, and converts a character string received from the driver manager from UTF-16LE format to the HiRDB server's character codes. This driver converts data received from the HiRDB server from the HiRDB server's codes to UTF-16LE format and then returns the data to the driver manager. However, if the PDCLTCNVMODE client environment definition is specified, the Linux edition of the HiRDB ODBC 3.5 driver converts character codes according to the PDCLTCNVMODE specification regardless of the character codes of the connection-target HiRDB server. The following table shows the combinations of character codes that can be used when ODBC functions supporting Unicode are used, and the character code conversion methods.

Table 14-25: Combinations of character codes that can be used when ODBC functions supporting Unicode are used and the character code conversion methods

PDCLTCNVMODE specification	ODBC driver	Client library	HiRDB server
UCS2_UTF8	UTF-16LE ←→ UTF-16LE	UTF-16LE ←→ UTF-8	UTF-8
UCS2_UJIS	UTF-16LE ←→ UTF-16LE	UTF-16LE ←→ EUC	EUC
UCS2_HJ	UTF-16LE ←→ UTF-16LE	UTF-16LE ←→ EUC-HJ	EUC-HJ
UTF8	UTF-16LE ←→ SJIS	SJIS ←→ UTF-8	UTF-8
UJIS	UTF-16LE ←→ SJIS	SJIS ←→ EUC	EUC
EUCHJ	UTF-16LE ←→ SJIS	SJIS ←→ EUC-HJ	EUC-HJ
Omitted	UTF-16LE ←→ SJIS	SJIS ←→ SJIS	SJIS
	UTF-16LE ←→ UTF-8	UTF-8 ←→ UTF-8	UTF-8
	UTF-16LE ←→ EUC	EUC ←→ EUC	EUC

- When an ODBC function that does not support Unicode is used, the Linux edition of the HiRDB ODBC 3.5 driver does not convert character strings passed as arguments. In such a case, you must ensure that the character codes used in the UAP execution environment match the character codes used on the connection-target HiRDB server. If the PDCLTCNVMODE client environment definition is specified, character codes are converted according to the PDCLTCNVMODE specification.

- When a connection function of an ODBC function that supports Unicode is used, the Linux edition of the HiRDB ODBC 3.5 driver converts a data source name to the character codes specified in the LANG environment variable in the execution environment. Therefore, if the character codes in the execution environment's LANG environment variable do not match the character codes for the registered data source name, the driver might not be able to read a data source name that is not in ASCII characters.

- The Linux edition of the HiRDB ODBC 3.5 driver does not support display of dialog boxes. Therefore, if SQLDriverConnect is called with a value other than SQL_DRIVER_NOPROMPT specified when connection is established with the HiRDB server, the driver returns SQL_ERROR.

- The Linux edition of the HiRDB ODBC 3.5 driver does not support registration of data sources that use the unixODBC-provided GUI tool ODBC Config. When you register data sources, you must follow the procedure described in *14.3.1(4) Installation procedure (in the Linux edition)*.

14.14 Automatic SQL statement generation by using .NET Framework Data Provider for ODBC

.NET Framework Data Provider for ODBC is one of the .NET Framework class libraries provided by Microsoft. .NET Framework Data Provider for ODBC enables you to automatically generate SQL statements for updating a data source by using the `OdbcCommandBuilder` class.

The following limitations apply when the HiRDB ODBC 3.5 driver is used with .NET Framework Data Provider for ODBC to automatically generate SQL statements:

- Automatic generation of SQL statements via .NET Framework Data Provider for ODBC is supported by HiRDB version 08-04 or later. If a HiRDB version earlier than 08-04 is used, operation is not guaranteed.
- If you specify a correlation name as a table name in the `SELECT` statement that is specified in the `SelectCommand` property of the `OdbcDataAdapter` class, which is the base of automatic generation of SQL statements, the HiRDB server returns the base table name to the HiRDB ODBC 3.5 driver. Therefore, a valid SQL statement is generated for the HiRDB server. However, if an alias is specified as a table name, the HiRDB server does not return a table name to the HiRDB ODBC 3.5 driver. As a result, no table name can be returned to .NET Framework Data Provider for ODBC that requires the value of `SQL_DESC_BASE_TABLE_NAM` for the `SQLColAttribute` function of the HiRDB ODBC 3.5 driver. In such a case, automatic generation of SQL statements is not guaranteed.
- If you specify an alias as a column name in a `SELECT` statement that is specified in the `SelectCommand` property of the `OdbcDataAdapter` class, which is the base of automatic generation of SQL statements, the HiRDB server returns the alias of the column to the HiRDB ODBC 3.5 driver. Therefore, the alias of the column is used in the automatically generated SQL statement. If an SQL statement containing an alias of a column is executed, the HiRDB server returns an error because no such column actually exists.

Chapter 15. HiRDB Access from OLE DB Application Programs

This chapter provides an overview of the OLE DB and discusses its connection interface, schema information, and error handling procedures.

This chapter contains the following sections:

15.1 Overview
15.2 Connection interface
15.3 Schema information
15.4 Data type correspondences
15.5 Error handling procedures
15.6 Notes

15.1 Overview

(1) What is OLE DB?

OLE DB is an API, like ODBC, for accessing a wide range of data sources. Unlike ODBC, OLE DB contains interface definitions suitable for accessing data other than SQL data.

(2) HiRDB OLE DB Provider

To access HiRDB from an OLE DB-supported application program, you need a HiRDB OLE DB provider. The HiRDB OLE DB provider is included in HiRDB/Run Time and HiRDB/Developer's Kit.

(3) Installing the HiRDB OLE DB provider

To install the HiRDB OLE DB provider when installing HiRDB/Run Time or HiRDB/Developer's Kit, in the Setup Type dialog box, choose **Custom**, and in the Select Components dialog box, select **OLE DB provider for HiRDB**.

When you install the HiRDB OLE DB provider, the following files are created:

- `PDOLEDB.DLL`
- `PDCLTM32.DLL`

(4) HiRDB OLE DB provider name

The name of the HiRDB OLE DB provider (provider program ID) is `HiRDBProvider`. When using an interface that requires the provider name (such as ActiveX Data Object (ADO)), you can use the HiRDB OLE DB provider by specifying this provider name in the `connection` object's `Provider` property.

15.2 Connection interface

This section explains the registry information and connection property.

15.2.1 Registry information

(1) Adding to the HKEY_CLASSES_ROOT key

(a) Provider program ID = provider name

```
"HiRDBProvider"="Hitachi HiRDB OLE DB Provider"
```

(b) Provider class ID

```
"HiRDBProvider\\ClSID"
  ="{6A708561-748A-11d3-B810-0000E2212E58}"
```

(2) Adding to the HKEY_CLASSES_ROOT\CLSID subkey

(a) Provider program ID

```
{"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}"
  ="HiRDBProvider"
```

(b) Provider name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}\\ProgID"
  ="HiRDBProvider"
```

(c) Program ID by version

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\VersionIndependentProgID"="HiRDBProvider"
```

(d) Provider DLL name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\InprocServer32"="pdoledb.dll"
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\InprocServer32\\ThreadingModel"="Both"
```

(e) Comment

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\OLE DB Provider"="Hitachi HiRDB OLE DB Provider"
```

(f) Extended error name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\ExtendedErrors"="Hitachi HiRDB OLE DB Provider"
```

(g) Extended error comment

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\ExtendedErrors\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  = "Hitachi HiRDB OLE DB Provider"
```

(3) Adding to the HKEY_CLASSES_ROOT key

(a) Provider error program ID

```
" HiRDBProviderErrors"="Hitachi HiRDB OLE DB Provider"
```

(b) Provider error class ID

```
"HiRDBProviderErrors\\ClSID"
  ="{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
```

(4) Adding to the HKEY_CLASSES_ROOT\CLSID subkey

(a) Provider error program ID

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  ="HiRDBProvider Error Lookup"
```

(b) Provider error lookup name

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}\\ProgID"
  ="HiRDBProvider Error Lookup"
```

(c) Error lookup program ID by version

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\VersionIndependentProgID"="HiRDBProvider Error Lookup"
```

(d) Provider error lookup DLL name

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\InprocServer32"="pdoledb.dll"
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\InprocServer32\\ThreadingModel"="Both"
```

15.2.2 Connection properties

Three `Initialization` properties are used for connection. These three properties are optional.

(1) DBPROP_INIT_DATASOURCE

This is the client's environment variable group name. If this property is omitted, the system assumes `HiRDB.INI`. For details about the client's environment variable group, see *6.7 Registering an environment variable group*.

(2) DBPROP_AUTH_USERID

This is the authorization identifier used for connection.

If this property is omitted, the authorization identifier is acquired from `PDUSER` of the applicable client environment variables group. If there is no `DBPROP_INIT_DATASOURCE` specification, the authorization identifier is acquired from `HiRDB.INI`.

(3) DBPROP_AUTH_PASSWORD

This is the password to be used for connection. If this property is omitted, but DBPROP_INIT_DATASOURCE is specified, the system obtains the password from PDUSER in the corresponding client environment variable group. If DBPROP_INIT_DATASOURCE is also omitted, the system obtains the password from HiRDB.INI.

15.3 Schema information

The following table lists the schema information provided by the HiRDB OLE DB provider.

Table 15-1: Schema information provided by the HiRDB OLE DB provider

Type of OLE DB schema information	Description	Provided
ASSERTIONS	Assertion information	--
CATALOGS	Catalog information	--
CHARACTER_SETS	Character set identification	--
CHECK_CONSTRAINTS	CHECK constraint identification	--
COLLATIONS	Character collation identification	--
COLUMN_DOMAIN_USAGE	Domain-dependent column information	--
COLUMN_PRIVILEGES	Column privilege information	--
COLUMNS	Column information	P (required)
CONSTRAINT_COLUMN_USAGE	Various constraint (reference, UNIQUE, CHECK) column information	--
CONSTRAINT_TABLE_USAGE	Various constraint (reference, UNIQUE, CHECK) table information	--
FOREIGN_KEYS	External key information	--
INDEXES	Index information	P
KEY_COLUMN_USAGE	Key column information	--
PRIMARY_KEYS	Primary key information	--
PROCEDURE_COLUMNS	Column information for row set returned by procedure	--
PROCEDURE_PARAMETERS	Procedure parameter information	P
PROCEDURES	Procedure information	P
PROVIDER_TYPES	Provider data type identification	P (required)
REFERENTIAL_CONSTRAINTS	Reference constraints	--

Type of OLE DB schema information	Description	Provided
SCHEMATA	Schema information	P
SQL_LANGUAGES	Match level for processing SQL installation and language type	--
STATISTICS	Statistical information	--
TABLE_CONSTRAINTS	Table constraints	--
TABLE_PRIVILEGES	Table privilege information	P
TABLES	Table information	P (required)
TRANSLATIONS	Character conversion identification	--
USAGE_PRIVILEGES	User privilege information	--
VIEW_COLUMN_USAGE	View column information	--
VIEWS	View information	--

P: Provided.

--: Not provided.

15.4 Data type correspondences

The following table shows the correspondences between the HiRDB data types and the OLE DB type indicators.

Table 15-2: Correspondences between the HiRDB data types and the OLE DB type indicators

HiRDB data types	OLE DB type indicators
CHAR, MCHAR, and NCHAR	DBTYPE_STR
VARCHAR, MVARCHAR, and NVARCHAR	
DECIMAL(*p*,*s*)	DBTYPE_NUMERIC
SMALLINT (signed)	DBTYPE_I2
INTEGER (signed)	DBTYPE_I4
REAL	DBTYPE_R4
SMALLFLT	
FLOAT	DBTYPE_R8
DOUBLE PRECISION	
BLOB	DBTYPE_BYTES
BINARY	
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
INTERVAL YEAR TO DAY	DBTYPE_DECIMAL
INTERVAL HOUR TO SECOND	DBTYPE_DECIMAL

15.5 Error handling procedures

15.5.1 Troubleshooting facility

This facility collects trace information about the OLE DB interface (for each method) issued by consumers.

(1) Collection method

Specify appropriate values in the following registry keys:
`HKEY_LOCAL_MACHINE`

Trace information is collected only when the value of `Software\HITACHI\HiRDB\oleprovtrc` is `1`.

Specify the absolute path of the output file name in `Software\HITACHI\HiRDB\oletrcfile`. (If `oletrcfile` is omitted, the system outputs trace information to `c:\temp\pdoletrc.txt`.)

Trace information is output to `Software\HITACHI\HiRDB\oletrcdumpsize` with `GetData()` and input with `Execute()`. Specify the `void*` type data dump output size in bytes. (If `oletrcdumpsize` is omitted, the system assumes `256`.)

15.6 Notes

(1) About a cursor in ADO

In HiRDB, you use the client cursor (`adUseClient` specified in the `CursorLocation` property of the `RecordSet` object) to scroll the cursor down and to update rows in ADO.

You use the server cursor (`adUseServer` specified in the `CursorLocation` property of the `Recordset` object) for all other purposes.

When you execute `CALL` statements, you must use the server cursor because result sets from procedures are not supported.

Chapter

16. HiRDB Access from ADO.NET-compatible Application Programs

This chapter describes the installation and functions of HiRDB.NET Data Provider, which is required in order to access HiRDB from ADO.NET-compatible application programs. It also provides examples of a UAP.

Although HiRDB.NET Data Provider for ADO.NET 1.1 has become obsolete, it is still supported in order to maintain compatibility with applications; however, this support will be discontinued in the future. Please use HiRDB.NET Data Provider that supports ADO.NET 2.0.

16.1 Overview
16.2 Installing HiRDB.NET Data Provider
16.3 List of classes provided by HiRDB.NET Data Provider
16.4 List of members provided by HiRDB.NET Data Provider
16.5 Interfaces of HiRDB.NET Data Provider
16.6 Notes about HiRDB.NET Data Provider
16.7 Data types of HiRDB.NET Data Provider
16.8 Connection pooling function
16.9 Provider-independent codes using DbProviderFactory
16.10 Troubleshooting function of HiRDB.NET Data Provider
16.11 Example of a UAP using HiRDB.NET Data Provider

16.1 Overview

16.1.1 HiRDB.NET Data Provider

.NET Framework provides a common-language runtime that does not depend on the platform or development language being used. It also provides the .NET Framework class libraries. ADO.NET is a library that can be used when .NET Framework applications that access databases are created.

HiRDB provides HiRDB.NET Data Provider, which is required to access HiRDB using ADO.NET. HiRDB.NET Data Provider complies with ADO.NET specifications.

HiRDB.NET Data Provider provides the common basic interface group that is provided in .NET Framework's System.Data address space. It also provides the `INSERT` facility using arrays and accesses to repetition columns as unique extended functions.

16.1.2 Prerequisite programs for HiRDB.NET Data Provider

(1) Supported platforms

ADO.NET 1.1 support (32-bit mode)

- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Server 2003 R2
- Windows Server 2008
- Windows Vista
- Windows 7

ADO.NET 2.0 support (32-bit mode)

- Windows 2000 SP3 or later
- Windows XP SP2 or later
- Windows Server 2003 SP1 or later
- Windows Server 2003 R2
- Windows Server 2008
- Windows Vista
- Windows 7

ADO.NET 2.0 support (64-bit mode)

- Windows XP
- Windows Server 2003
- Windows Server 2003 R2
- Windows Server 2008
- Windows Server 2008 R2
- Windows Vista
- Windows 7

(2) Required programs

The environment requirements in order to develop and execute application programs are explained in this subsection.

To develop ADO.NET 1.1-compatible UAPs:

Development environment

- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio 2005

Execution environment

- Microsoft Internet Explorer 5.01 or later
- Redistributable package of .NET Framework Version 1.1 or later

To develop ADO.NET 2.0-compatible UAPs:

Development environment

Select one of the following:

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008

Execution environment

- Microsoft Internet Explorer 5.01 or later

Select one of the following:

- Redistributable package of .NET Framework Version 2.0
- Redistributable package of .NET Framework Version 3.0
- Redistributable package of .NET Framework 3.5
- Redistributable package of .NET Framework 4

If you use an ADO.NET2.0-compatible HiRDB.NET Data Provider in an environment where only .NET Framework 4 is installed, set the `useLegacyV2RuntimeActivationPolicy` attribute to `true` in the `<startup>` element in the application configuration file.

To develop ADO.NET2.0-compatible UAPs (64-bit mode):

Development environment

Select one of the following:

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008

Execution environment

Microsoft Internet Explorer 5.01 or later

Select one of the following:

- Redistributable package of .NET Framework Version 2.0
- Redistributable package of .NET Framework Version 3.0
- Redistributable package of .NET Framework 3.5
- Redistributable package of .NET Framework 4

If you use an ADO.NET2.0-compatible HiRDB.NET Data Provider in an environment where only .NET Framework 4 is installed, set the `useLegacyV2RuntimeActivationPolicy` attribute to `true` in the `<startup>` element in the application configuration file.

You can install a redistributable package of .NET Framework as a Windows update. Its operation is not guaranteed in Microsoft Visual Studio .NET 2002 + .NET Framework SDK Version 1.1.

16.2 Installing HiRDB.NET Data Provider

16.2.1 Installation procedure

When you install HiRDB/Run Time or HiRDB/Developer's Kit, you can install HiRDB.NET Data Provider by choosing **Typical** or **Custom** in the Setup Type dialog box, and then selecting **HiRDB Data Provider** in the Select Features dialog box.

A group of DLLs and publisher policies for HiRDB.NET Data Provider are placed in the global assembly cache. Because the most recent HiRDB.NET Data Provider that has been installed is used based on the publisher policies, there is no need to re-build UAPs when you upgrade HiRDB.NET Data Provider.

For details about placing HiRDB.NET Data Provider in the global assembly cache, see *16.6.1 Placing in global assembly cache*.

16.2.2 Files that are installed

When HiRDB.NET Data Provider is installed, the following files are created:

For ADO.NET version 1.1:

- `pddndp.dll`
- `pddndpcore.dll`

For ADO.NET version 2.0:

- `pddndp20.dll`
- `pddndpcore20.dll`
- `pddndp20x.dll`
- `pddndpcore20x.dll`

If you use HiRDB.NET Data Provider, make the above files accessible.

16.2.3 Checking the version information

You can check the version information of HiRDB.NET Data Provider by displaying the DLL properties provided by HiRDB.NET Data Provider.

16.3 List of classes provided by HiRDB.NET Data Provider

HiRDB.NET Data Provider complies with ADO.NET specifications.

The following table lists and describes the classes provided by HiRDB.NET Data Provider.

Table 16-1: List of HiRDB.NET Data Provider classes

Class	Function	Supported ADO.NET version
`HiRDBCommand`	Represents an SQL statement or stored procedure that is executed on a database.	1.1 or later
`HiRDBCommandBuilder`	Automatically creates a single table command to make a change to `DataSet` that has been associated with a database.	1.1 or later
`HiRDBConnection`	Represents an open connection to a database.	1.1 or later
`HiRDBDataAdapter`	Represents a series of data commands and database connections that are used to store data in `DataSet` and update a database.	1.1 or later
`HiRDBDataReader`	Provides a method for reading a forward stream in data rows from a database.	1.1 or later
`HiRDBException`	Represents an exception that is created when a warning or error is returned from HiRDB.NET Data Provider.	1.1 or later
`HiRDBParameter`	Represents a `HiRDBCommand` parameter and a map for `DataColumn` as an option.	1.1 or later
`HiRDBParameterCollection`	Represents a parameter collection associated with `HiRDBCommand` and a map of each parameter for `DataSet` columns.	1.1 or later
`HiRDBProviderFactory`	Creates an instance of each class provided by HiRDB.NET Data Provider	2.0 or later

Class	Function	Supported ADO.NET version
`HiRDBRowUpdatedEventArgs`	Provides data for a `RowUpdated` event.	1.1 or later
`HiRDBRowUpdatingEventArgs`	Provides data for a `RowUpdating` event.	1.1 or later
`HiRDBTransaction`	Represents a transaction that is executed on a database.	1.1 or later

16.4 List of members provided by HiRDB.NET Data Provider

This section presents a list of interface members provided by HiRDB.NET Data Provider.

16.4.1 List of HiRDBCommand members

(1) Constructor

HiRDBCommand

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

Component, IDbCommand, ICloneable

When the supported ADO.NET version is 2.0:

DbCommand

(3) Properties

Member	Function	Supported ADO.NET version
CommandText	Acquires or sets the text command that is executed on a database.	1.1 or later
CommandTimeout	Acquires or sets the SQL execution timeout value (seconds).	1.1 or later
CommandType	Acquires or sets a value that indicates how to interpret the CommandText property.	1.1 or later
Connection	Acquires or sets the HiRDBConnection that is used by this HiRDBCommand.	1.1 or later
DesignTimeVisible	Acquires or sets a value indicating whether a HiRDBCommand object is to be displayed on the control when the HiRDBCommand object is linked to the interface control.	2.0 or later
Parameters	Acquires HiRDBParameterCollection.	1.1 or later
Transaction	Acquires or sets the HiRDBTransaction on which this HiRDBCommand is executed.	1.1 or later
UpdatedRowSource	Acquires or sets how to apply the command result to DataRow when HiRDBDataAdapter's Update method uses the command result.	1.1 or later

(4) Methods

Member	Function	Supported ADO.NET version
`Cancel ()`	Cancels an SQL statement during `HiRDBCommand` execution.	1.1 or later
`Clone ()`	Creates a new object which is a copy of the current instance.	1.1 or later
`CreateParameter ()`	Creates a new instance of the `HiRDBParameter` object.	1.1 or later
`ExecuteNonQuery ()`	Executes an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.	1.1 or later
`ExecuteNonQuery (int)`		1.1 or later
`ExecuteReader ()`	Executes `CommandText` on `HiRDBConnection` and creates `HiRDBDataReader`.	1.1 or later
`ExecuteReader (CommandBrhavior)`		1.1 or later
`ExecuteScalar ()`	Executes a query and returns the first column of the first row in the result set returned by the query. Any excess column or row will be ignored.	1.1 or later
`Prepare ()`	Creates a prepared version of a command (compiled) in a database.	1.1 or later

16.4.2 List of HiRDBCommandBuilder members

(1) Constructor

`HiRDBCommandBuilder`

(2) Inheritance class

When the supported ADO.NET version is 1.1:

`Component`

When the supported ADO.NET version is 2.0:

`DbCommandBuilder`

(3) Properties

Member	Function	Supported ADO.NET version
`CatalogLocation`	Indicates the location of the catalog name when a server name, catalog name, schema name, and table name are used to express a qualified table name.	2.0 or later
`CatalogSeparator`	Sets or acquires the character string used as the catalog delimiter for the `HiRDBCommandBuilder` object.	2.0 or later
`ConflictOption`	Sets or acquires a combination of columns (concurrent execution check type) that are to be added to the `WHERE` clause of `UpdateCommand` or `DeleteCommand` that is created by the corresponding object.	2.0 or later
`DataAdapter`	Acquires or sets the `HiRDBDataAdapter` object for which an SQL statement is to be created automatically.	1.1 or later
`QuotePrefix`	Acquires or sets the start character used for specifying a column or table identifier.	2.0 or later
`QuoteSuffix`	Acquires or sets the end character used for specifying a column or table identifier.	2.0 or later
`SchemaSeparator`	Acquires or sets the character used as the delimiter between an authorization identifier and another identifier.	2.0 or later
`SetAllValues`	Acquires or sets a value indicating whether all columns are to be subject to updating of their values by the `UPDATE` statement.	2.0 or later

(4) Methods

Member	Function	Supported ADO.NET version
`GetDeleteCommand ()`	Acquires the automatically created `HiRDBCommand` object for executing deletion processing on the database.	2.0 or later
`GetDeleteCommand (bool)`		2.0 or later
`GetDeleteCommand (string)`		1.1 or later

Member	Function	Supported ADO.NET version
`GetInsertCommand ()`	Acquires the automatically created `HiRDBCommand` object for executing insertion processing on the database.	2.0 or later
`GetInsertCommand (bool)`		2.0 or later
`GetInsertCommand (string)`		1.1 or later
`GetUpdateCommand ()`	Acquires the automatically created `HiRDBCommand` object for executing update processing on the database.	2.0 or later
`GetUpdateCommand (bool)`		2.0 or later
`GetUpdateCommand (string)`		1.1 or later
`QuoteIdentifier (string)`	Returns a specified character string enclosed by the `HiRDBCommandBuilder#QuotePrefix` and `HiRDBCommandBuilder#QuoteSuffix` property values.	2.0 or later
`RefreshSchema ()`	Updates database schema information to create the `INSERT`, `UPDATE`, or `DELETE` statement.	2.0 or later
`RefreshSchema (string)`		1.1 or later
`UnquoteIdentifier (string)`	Returns the value obtained by removing the `HiRDBCommandBuilder#QuotePrefix` property value at the beginning and the `HiRDBCommandBuilder#QuoteSuffix` property value at the end of a specified character string.	2.0 or later

16.4.3 List of HiRDBConnection members

(1) Constructor

`HiRDBConnection`

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

`Component`, `IDbConnection`, `ICloneable`

When the supported ADO.NET version is 2.0:

DbConnection

(3) Properties

Member	Function	Supported ADO.NET version
ConnectionString	Acquires or sets the character string that is used to open a database.	1.1 or later
ConnectionTimeout	Acquires the wait time for establishing a connection before retries are cancelled and an error is generated.	1.1 or later
Database	Acquires the name of the current database or the database that is used when a connection is established.	1.1 or later
DataSource	Acquires the name of the connected database server.	2.0 or later
LifeTime	Acquires or sets the time remaining before actual disconnection occurs.	1.1 or later
Pooling	Acquires or sets whether or not pooling is to be performed.	1.1 or later
ServerVersion	Acquires the version of the connected server.	2.0 or later
State	Acquires the current connection status.	1.1 or later

(4) Methods

Member	Function	Supported ADO.NET version
BeginTransaction ()	Starts the database transaction using the specified IsolationLevel value.	1.1 or later
BeginTransaction (IsolationLevel)		1.1 or later
ChangeDatabase (string)	Changes the current database for the open HiRDBConnection object.	1.1 or later
Clone ()	Creates a new object which is a copy of the current instance.	1.1 or later
Close ()	Closes the connection to the database.	1.1 or later
CreateCommand ()	Creates and returns the HiRDBCommand object associated with the connection.	1.1 or later
EnlistTransaction (Transaction)	Registers in the specified transaction.	2.0 or later

Member	Function	Supported ADO.NET version
`GetSchema ()`	Returns schema information.	2.0 or later
`GetSchema (string)`		2.0 or later
`GetSchema (string, string[])`		2.0 or later
`Open ()`	Opens the database connection with the settings specified in the `ConnectionString` property of the `HiRDBConnection` object.	1.1 or later

16.4.4 List of HiRDBDataAdapter members

(1) Constructor

`HiRDBDataAdapter`

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

`DbDataAdapter`, `IDbDataAdapter`

When the supported ADO.NET version is 2.0:

`DbDataAdapter`

(3) Properties

Member	Function	Supported ADO.NET version
`DeleteCommand`	Acquires or sets the SQL statement for deleting records from a data set.	1.1 or later
`InsertCommand`	Acquires or sets the SQL statement for inserting new records in a database.	1.1 or later
`SelectCommand`	Acquires or sets the SQL statement for selecting records in a database.	1.1 or later
`UpdateCommand`	Acquires or sets the SQL statement for updating records in a database.	1.1 or later

(4) Events

Member	Function	Supported ADO.NET version
RowUpdated	Event occurring after a command has been executed on the data source during update processing.	1.1 or later
RowUpdating	Event occurring before a command is executed on the data source during update processing.	1.1 or later

16.4.5 List of HiRDBDataReader members

(1) Constructor

HiRDBDataReader

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataRecord

When the supported ADO.NET version is 2.0:

DbDataReader

(3) Properties

Member	Function	Supported ADO.NET version
Depth	Acquires the value indicating the nesting level of the current row.	1.1 or later
FieldCount	Acquires the number of columns in the current row.	1.1 or later
HasRows	Acquires a value indicating whether the corresponding HiRDBDataReader contains at least one row.	2.0 or later
IsClosed	Acquires the value indicating whether or not the data reader is closed.	1.1 or later
Item[int]	Acquires data by treating the HiRDBDataReader object as an array.	1.1 or later
Item[int,int]		1.1 or later
Item[string]		1.1 or later
RecordsAffected	Acquires the number of rows changed, inserted, or deleted by execution of the SQL statement.	1.1 or later

Member	Function	Supported ADO.NET version
`VisibleFieldCount`	Acquires the number of hidden rows for `HiRDBDataReader`.	2.0 or later

(4) Methods

Member	Function	Supported ADO.NET version
`Close ()`	Closes the `HiRDBDataReader` object.	1.1 or later
`GetBoolean (int)`	Acquires the value of the specified column as a Boolean value.	1.1 or later
`GetByte (int)`	Acquires an unsigned 8-bit integer value in the specified column.	1.1 or later
`GetBytes (int, long, byte[], int, int)`	Reads a byte stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.	1.1 or later
`GetChar (int)`	Acquires the character string value in the specified column.	1.1 or later
`GetChars (int, long, byte[], int, int)`	Reads a character stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.	1.1 or later
`GetData (int)`	The purpose of this member is to support the .NET Framework infrastructure. It cannot be used directly in a unique coding that has been created.	1.1 or later
`GetDataTypeName (int)`	Acquires data-type information for the specified field.	1.1 or later
`GetDateTime (int)`	Acquires or sets the date and time data value in the specified field.	1.1 or later
`GetDecimal  (int)`	Acquires the fixed position value in the specified field.	1.1 or later
`GetDouble (int)`	Acquires the double-precision floating-point number in the specified field.	1.1 or later
`GetEnumerator ()`	Returns the enumerator that can perform iterative operation on a collection.	1.1 or later
`GetFieldArrayCount (int)`	Acquires the size of field array.	1.1 or later

Member	Function	Supported ADO.NET version
GetFieldType (int)	Acquires Type information corresponding to the type of Object that is returned from GetValue.	1.1 or later
GetFloat (int)	Acquires the single-precision floating-point number in the specified field.	1.1 or later
GetGuid (int)	Returns the GUID value of the specified field.	1.1 or later
GetInt16 (int)	Acquires a signed 16-bit integer value in the specified field.	1.1 or later
GetInt32 (int)	Acquires a signed 32-bit integer value in the specified field.	1.1 or later
GetInt64 (int)	Acquires a signed 64-bit integer value in the specified field.	1.1 or later
GetName (int)	Acquires the name of the field to be searched.	1.1 or later
GetOrdinal (string)	Returns the index of the specified field.	1.1 or later
GetProviderSpecificFieldType (int)	Acquires the data type of a specified column.	2.0 or later
GetProviderSpecificValue (int)	Acquires the value of a specified column as an instance of Object.	2.0 or later
GetProviderSpecificValues (Object[])	Acquires all attribute columns in the current collection of rows.	2.0 or later
GetSchemaTable ()	Returns the DataTable that describes HiRDBDataReader's column metadata.	1.1 or later
GetString (int)	Acquires a character string in the specified field.	1.1 or later
GetValue (int)	Returns a value in the specified field.	1.1 or later
GetValue (int, int)	Acquires the value of a specified element in a specified column as an Object-type object.	1.1 or later
GetValues (object[])	Acquires all attribute fields in the current record collection.	1.1 or later
IsDBNull (int)	Returns a value indicating whether or not the specified field is set to null.	1.1 or later
NextResult ()	Advances the data reader to the next result when the result of a batch SQL statement is read.	1.1 or later
Read ()	Advances HiRDBDataReader to the next record.	1.1 or later

16.4.6 List of HiRDBException members

(1) Constructor

`HiRDBException`

(2) Inheritance class

When the supported ADO.NET version is 1.1:

`Exception`

When the supported ADO.NET version is 2.0:

`DbException`

(3) Properties

Member	Function	Supported ADO.NET version
`ErrorCode`	Acquires the error code part as an `int`.	1.1 or later
`Message`	Acquires text with a complete error.	1.1 or later

16.4.7 List of HiRDBParameter members

(1) Constructor

`HiRDBParameter`

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

`MarshalByRefObject`, `IDbDataParameter`, `IDataParameter`, `ICloneable`

When the supported ADO.NET version is 2.0:

`DbParameter`

(3) Properties

Member	Function	Supported ADO.NET version
`DbType`	Acquires or sets `DbType` for a parameter. When `DbType` is to be set, this member sets the corresponding data type in the `HiRDBType` property according to *Table 16-3*.	1.1 or later
`Direction`	Acquires or sets a value indicating whether the parameter is input only, output only, bidirectional, or the stored procedure's return value.	1.1 or later

Member	Function	Supported ADO.NET version
HiRDBType	Acquires or sets an enumeration indicating the data type in HiRDB. When an enumeration is to be set, this member sets the corresponding data type in the DbType property according to *Table 16-4*. HiRDBType enumeration: Integer, SmallInt, Decimal, Float, SmallFlt, Char, VarChar, NChar, NVarChar, MChar, MVarChar, Date, Time, TimeStamp, IntervalYearToDay, IntervalHourToSecond, Blob, Binary	1.1 or later
IsNullable	Acquires a value indicating whether or not the parameter accepts the null value.	1.1 or later
ParameterName	Acquires or sets the name of HiRDBParameter.	1.1 or later
Precision	Acquires or sets the number of significant digits (including decimal places) in the definition length of a DECIMAL-type parameter. Specification of this member is mandatory when the HiRDBType property value is HiRDBType.Decimal.	1.1 or later
Repetition	Acquires or sets an array structure in HiRDB.	1.1 or later
Scale	Acquires or sets the number of decimal places in the definition length of a DECIMAL-type parameter. Specification of this member is mandatory when the HiRDBType property value is HiRDBType.Decimal.	1.1 or later
Size	Acquires or sets a parameter definition length. If the length is fixed (such as numeric type or date/time type), specify 0. If the length is variable (such as character string type), specify the number of bytes to be stored in the table or the maximum column length. For TIMESTAMP (DateTime), this value is the number of digits in the fractional part. Note that if the entered character string is longer than the size specified for the Size property, only up to the specified number of characters are stored. No exception occurs.	1.1 or later
SourceColumn	Acquires or sets the name of the source column that has been assigned to DataSet and is used to read or return Value.	1.1 or later
SourceColumnNullMapping	Acquires or sets a value that indicates whether the column of the DataTable object corresponding to the parameter permits the NULL value.	1.1 or later

Member	Function	Supported ADO.NET version
`SourceVersion`	Acquires or sets the `DataRowVersion` that is used to read `Value`.	1.1 or later
`Value`	Acquires or sets a parameter value.	1.1 or later

(4) Method

Member	Function	Supported ADO.NET version
`Clone`	Creates a new object which is a copy of the current instance.	1.1 or later
`ResetDbType ()`	Resets the `DbType` property to its initial value.	2.0 or later

16.4.8 List of HiRDBParameterCollection members

(1) Constructor

`HiRDBParameterCollection`

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

`MarshalByRefObject`, `IDataParameterCollection`, `IList`, `ICollection`, `IEnumerable`

When the supported ADO.NET version is 2.0:

`DbParameterCollection`

(3) Properties

Member	Function	Supported ADO.NET version
`Count`	Acquires the number of `HiRDBParameter` objects stored in `HiRDBParameterCollection`.	1.1 or later
`IsFixedSize`	Acquires a value indicating whether the size of `HiRDBParameterCollection` is fixed.	1.1 or later
`IsReadOnly`	Acquires a value indicating whether or not `HiRDBParameterCollection` is read only.	1.1 or later
`IsSynchronized`	Acquires a value indicating whether or not an access to `HiRDBParameterCollection` is synchronized (thread-safe).	1.1 or later

Member	Function	Supported ADO.NET version
`Item[int]`	Acquires the `HiRDBParameter` object for a specified index or sets the `HiRDBParameter` object in a specified index.	1.1 or later
`Item[string]`		1.1 or later
`SyncRoot`	Acquires an object that can be used to synchronize an access to `HiRDBParameterCollection`.	1.1 or later

(4) Methods

Member	Function	Supported ADO.NET version
`Add (object)`	Adds items to `HiRDBParameterCollection`.	1.1 or later
`Add (HiRDBParameter)`		1.1 or later
`Add (string, object)`		1.1 or later
`Add (string, HiRDBType)`		1.1 or later
`Add (string, HiRDBType, int)`		1.1 or later
`Add (string, HiRDBType, int, string)`		1.1 or later
`AddRange (Array)`	Adds an array in the `HiRDBParameter` object to `HiRDBParameterCollection`.	1.1 or later
`AddRange (HiRDBParameter[])`		1.1 or later
`Clear ()`	Deletes all items from `HiRDBParameterCollection`.	1.1 or later
`Contains (object)`	Acquires a value indicating whether or not `HiRDBParameter` is in the collection.	1.1 or later
`Contains (HiRDBParameter)`		1.1 or later
`Contains (string)`		1.1 or later

Member	Function	Supported ADO.NET version
CopyTo (Array, int)	Copies the elements of HiRDBParameterCollection to Array using Array's specific index as the start position.	1.1 or later
GetEnumerator ()	Returns an enumerator that can perform iterative operation on a collection.	1.1 or later
IndexOf (object)	Acquires the location of HiRDBParameter in a collection.	1.1 or later
IndexOf (string)		1.1 or later
Insert (int, object)	Inserts an item at the specified location in HiRDBParameterCollection.	1.1 or later
Insert (int, HiRDBParameter)		1.1 or later
Remove	Deletes the first occurrence of the specified object in HiRDBParameterCollection.	1.1 or later
RemoveAt (int)	Deletes HiRDBParameter from the collection.	1.1 or later
RemoveAt (string)		1.1 or later

16.4.9 List of HiRDBProviderFactory members

(1) Constructor

HiRDBProviderFactory

(2) Inheritance classes

DbProviderFactory

(3) Field

Instance

(4) Properties

Member	Function	Supported ADO.NET version
CanCreateDataSourceEnumerator	Indicates whether a class derived from the DbDataSourceEnumerator class is to be supported.	2.0 or later

(5) Methods

Member	Function	Supported ADO.NET version
`CreateCommand ()`	Creates and returns the `HiRDBCommand` object.	2.0 or later
`CreateCommandBuilder ()`	Creates and returns a `HiRDBCommandBuilder` object.	2.0 or later
`CreateConnection ()`	Creates and returns a `HiRDBConnection` object.	2.0 or later
`CreateConnectionStringBuilder ()`	Creates and returns a `DbConnectionStringBuilder` object.	2.0 or later
`CreateDataAdapter ()`	Creates and returns a `HiRDBDataAdapter` object.	2.0 or later
`CreateDataSourceEnumerator ()`	Returns `System.NotSupportedException`.	2.0 or later
`CreateParameter ()`	Creates and returns a `HiRDBParameter` object.	2.0 or later

16.4.10 List of HiRDBRowUpdatedEventArgs members

(1) Constructor

`HiRDBRowUpdatedEventArgs`

(2) Inheritance class

`RowUpdatedEventArgs`

(3) Property

Member	Function	Supported ADO.NET version
`Command`	Acquires the `HiRDBCommand` that is executed when `Update` is called.	1.1 or later

16.4.11 List of HiRDBRowUpdatingEventArgs members

(1) Constructor

`HiRDBRowUpdatingEventArgs`

(2) Inheritance class

`RowUpdatingEventArgss`

(3) Property

Member	Function	Supported ADO.NET version
`Command`	Acquires or sets the `HiRDBCommand` that is executed during `Update` processing.	1.1 or later

16.4.12 List of HiRDBTransaction members

(1) Constructor

`HiRDBTransaction`

(2) Inheritance classes

When the supported ADO.NET version is 1.1:

`MarshalByRefObject`, `IDbTransaction`, `IDisposable`

When the supported ADO.NET version is 2.0:

`DbTransaction`

(3) Properties

Member	Function	Supported ADO.NET version
`Connection`	Acquires the `HiRDBConnection` object used to associate a transaction.	1.1 or later
`IsCompleted`	Acquires a value indicating whether or not the transaction is completed.	1.1 or later
`IsolationLevel`	Specifies this transaction's `IsolationLevel`.	1.1 or later

(4) Methods

Member	Function	Supported ADO.NET version
`Commit ()`	Commits a database transaction.	1.1 or later
`Rollback ()`	Rolls back a database transaction from the hold status.	1.1 or later

16.5 Interfaces of HiRDB.NET Data Provider

16.5.1 HiRDBCommand

(1) Constructor

(a) HiRDBCommand

`void HiRDBCommand ()`

Description: Initializes a new instance of `HiRDBCommand`.

`void HiRDBCommand (string)`

Argument

`string cmdText`: SQL text (`CommandText` property)

Description: Specifies an SQL text to initialize a new instance of the `HiRDBCommand` class.

`void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection)`

Arguments

`string cmdText`: SQL text (`CommandText` property)

`HiRDBConnection rConnection`: `HiRDBConnection` object representing the connection to the database (`Connection` property)

Description: Uses an SQL text and `HiRDBConnection` object to initialize a new instance of the `HiRDBCommand` class.

`void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection, Hitachi.HiRDB.HiRDBTransaction)`

Arguments

`string cmdText`: SQL text (`CommandText` property)

`HiRDBConnection rConnection`: `HiRDBConnection` object representing the connection to the database (`CommandText` property)

`HiRDBTransaction rTransaction`: `HiRDBTransaction` object that executes `HiRDBCommand` (`Transaction` property)

Description: Uses an SQL text and the `HiRDBConnection` and `HiRDBTransaction` objects to initialize a new instance of the `HiRDBCommand` class.

(2) Properties

(a) CommandText

Type: `string`

Default value: `""`

Description: Acquires or sets the text command that is executed on a database.

(b) CommandTimeout

Type: `int`

Default value: `30`

Description: Acquires or sets the SQL execution timeout value (seconds).

Exception: `HiRDBException`

(c) CommandType

Type: `System.Data.CommandType`

Default value: `CommandType.Text`

Description: Acquires or sets how to interpret the `CommandText` property.

(d) Connection

Type: `HiRDBConnection`

Default value: `null`

Description: Acquires or sets the `HiRDBConnection` that is used by this `HiRDBCommand`.

Exception: `HiRDBException`

(e) DesignTimeVisible

Type: `bool`

Default value: `true`

Description: Acquires or sets a value indicating whether a `HiRDBCommand` object is to be displayed on the control when the `HiRDBCommand` object is linked to the interface control.

(f) Parameters

Type: `HiRDBParameterCollection`

Description: Acquires `HiRDBParameterCollection` (read only).

(g) Transaction

Type: `HiRDBTransaction`

Default value: `null`

Description: Acquires or sets the `HiRDBTransaction` on which this `HiRDBCommand` is executed.

(h) UpdatedRowSource

Type: `System.Data.UpdateRowSource`

Default value: `UpdatedRowSource.None`

Description: Acquires or sets how to apply the command result to `DataRow` when `HiRDBDataAdapter`'s `Update` method uses the command result.

Exception: `HiRDBException`

(3) Methods

(a) Cancel

`void Cancel ()`

Return: `void`

Description: Cancels an SQL statement during `HiRDBCommand` execution.

(b) Clone

`object Clone ()`

Return

`object`: New object which is a copy of this instance

Description: Creates a new object which is a copy of the current instance.

(c) CreateParameter

`Hitachi.HiRDB.HiRDBParameter CreateParameter ()`

Return

`HiRDBParameter`: `HiRDBParameter` object

Description: Creates a new instance of the `HiRDBParameter` object.

(d) ExecuteNonQuery

`int ExecuteNonQuery ()`

Return

`int`: Number of affected rows

Description: Executes an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.

Exception: `HiRDBException`

`int ExecuteNonQuery (int)`

Argument

`int nArraySize`: Number of array elements

Return

`int`: Number of affected rows

Description: Uses the `INSERT` facility using arrays to execute an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.

Exception: `HiRDBException`

(e) ExecuteReader

`Hitachi.HiRDB.HiRDBDataReader ExecuteReader ()`

Return

`HiRDBDataReader`: `HiRDBDataReader` object

Description: Executes `CommandText` on `HiRDBConnection` to create `HiRDBDataReader`.

Exception: `HiRDBException`

`ExecuteReader (System.Data.CommandBehavior)`

Argument

`System.DataCommandBehavior behavior`: One of the `CommandBehavior` values

Return

`HiRDBDataReader`: `HiRDBDataReader` object

Description: Executes `CommandText` on `HiRDBConnection` and creates `HiRDBDataReader` using one of the `CommandBehavior` values.

Exception: `HiRDBException`

(f) ExecuteScalar

`object ExecuteScalar ()`

Return

`object`: First column of the first row in the result set

Description: Executes a query and returns the first column of the first row in the result set returned as .NET Framework's data type by that query. Any remaining column or row will be ignored.

Exception: `HiRDBException`

(g) Prepare

`void Prepare ()`

Return: `void`

Description: Creates a prepared version of a command (compiled) in a database.

Exception: `HiRDBException`

16.5.2 HiRDBCommandBuilder

(1) Constructor

(a) HiRDBCommandBuilder

`void HiRDBCommandBuilder ()`

Description: Initializes a new instance of `HiRDBCommandBuilder`.

`void HiRDBCommandBuilder (HiRDBDataAdapter adapter)`

Argument

`HiRDBDataAdapter adapter`: `HiRDBDataAdapter` object (`DataAdapter` property)

Description: Specifies the `HiRDBDataAdapter` object and initializes a new instance of `HiRDBCommandBuilder`.

(2) Properties

(a) CatalogLocation

Type: CatalogLocation

Default value: CatalogLocation.Start

Description: Indicates the location of the catalog name when a server name, catalog name, schema name, and table name are used to express a qualified table name.

Exception: HiRDBException

(b) CatalogSeparator

Type: string

Default value: `""`

Description: Sets or acquires the character string used as the catalog delimiter for the `HiRDBCommandBuilder` object.

Exception: HiRDBException

(c) ConflictOption

Type: ConflictOption

Default value: ConflictOption.CompareAllSearchableValues

Description: Sets or acquires a combination of columns (concurrent execution check type) that are to be added to the `WHERE` clause of `UpdateCommand` or `DeleteCommand` that is created by the corresponding object.

Exception: HiRDBException

(d) DataAdapter

Type: `HiRDBDataAdapter`

Default value: `null`

Description: Acquires or sets the `HiRDBDataAdapter` object for which an SQL statement is to be created automatically.

(e) QuotePrefix

Type: string

Default value: `"` (double-quotation mark)

Description: Acquires or sets the start character used for specifying a column or table identifier.

Exception: HiRDBException

(f) QuoteSuffix

Type: string

Default value: `"` (double-quotation mark)

Description: Acquires or sets the end character used for specifying a column or table identifier.

Exception: HiRDBException

(g) SchemaSeparator

Type: HiRDBDataAdapter

Default value: `.` (period)

Description: Acquires or sets the character used as the delimiter between an authorization identifier and another identifier.

Exception: HiRDBException

(h) SetAllValues

Type: bool

Default value: true

Description: Acquires or sets a value that indicates whether all columns are to be subject to updating of their values by the `UPDATE` statement.

(3) Methods

(a) GetDeleteCommand

HiRDBCommand GetDeleteCommand()

Return

`HiRDBCommand`: Automatically created `HiRDBCommand` object for executing deletion processing

Description: Acquires the automatically created `HiRDBCommand` object for executing deletion processing on the database. If the retrieval SQL statement specified in `SelectCommand` of the `DataAdapter` property satisfies either of the following conditions, the deletion SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

HiRDBCommand GetDeleteCommand(bool)

Argument

useColumnsForParameterNames

`true`: Parameter name based on a column name (such as `@ID`)

`false`: Parameter name in the `@p`*X* format (*X*: ordinal number beginning at 1)

Return

`HiRDBCommand`: Automatically created `HiRDBCommand` object for executing deletion processing

Description: Acquires the automatically created `HiRDBCommand` object for executing deletion processing on the database. If the retrieval SQL statement specified in `SelectCommand` of the `DataAdapter` property satisfies either of the following conditions, the deletion SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

`HiRDBCommand GetDeleteCommand (string)`

Argument

`string s TableName`: Table name

Return

HiRDBCommand: HiRDBCommand object that was automatically created to execute deletion processing

Description: Acquires the automatically created HiRDBCommand object for executing deletion processing on the database.

Exception: HiRDBException

(b) GetInsertCommand

```
HiRDBCommand GetInsertCommand ()
```

Return

HiRDBCommand: HiRDBCommand object that was automatically created to execute insertion processing

Description: Acquires the automatically created HiRDBCommand object for executing insertion processing on the database. If the retrieval SQL statement specified in SelectCommand of the DataAdapter property satisfies either of the following conditions, the insertion SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

HiRDBCommand GetInsertCommand (bool)

Argument

useColumnsForParameterNames

true: Parameter name based on a column name (such as @ID)

false: Parameter name in the @p*X* format (*X*: ordinal number beginning at 1)

Return

HiRDBCommand: Automatically created HiRDBCommand object for executing insertion processing

Description: Acquires the automatically created HiRDBCommand object for executing insertion processing on the database. If the retrieval SQL statement specified in SelectCommand of the DataAdapter property satisfies either of the following conditions, the insertion SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

HiRDBCommand GetInsertCommand (string)

Argument

`string sTableName`: Table name

Return

`HiRDBCommand`: Automatically created `HiRDBCommand` object for executing insertion processing

Description: Acquires the automatically created `HiRDBCommand` object for executing insertion processing on the database.

Exception: HiRDBException

(c) GetUpdateCommand

`HiRDBCommand GetUpdateCommand ()`

Return

`HiRDBCommand`: Automatically created `HiRDBCommand` object for executing update processing

Description: Acquires the automatically created `HiRDBCommand` object for executing update processing on the database. If the retrieval SQL statement specified in `SelectCommand` of the `DataAdapter` property satisfies either of the following conditions, the updating SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

HiRDBCommand GetUpdateCommand (bool)

Argument

useColumnsForParameterNames

`true`: Parameter name based on a column name (such as `@ID`)

`false`: Parameter name in the `@p`*X* format (*X*: ordinal number beginning at 1)

Return

`HiRDBCommand`: Automatically created `HiRDBCommand` object for executing update processing

Description: Acquires the automatically created `HiRDBCommand` object for executing update processing on the database. If the retrieval SQL statement specified in `SelectCommand` of the `DataAdapter` property satisfies either of the following conditions, the updating SQL statement cannot be created:

- This is not a retrieval on a single table.
- An alias is specified for the table.

Exception: HiRDBException

HiRDBCommand GetUpdateCommand (string)

Argument

`string s TableName`: Table name

Return

`HiRDBCommand`: `HiRDBCommand` object that was automatically created to execute update processing

Description: Acquires the automatically created `HiRDBCommand` object for executing update processing on the database.

Exception: `HiRDBException`

(d) QuoteIdentifier

string QuoteIdentifier(string)

Argument

`string unquotedIdentifier`: Character string enclosed by the `HiRDBCommandBuilder#QuotePrefix` and `HiRDBCommandBuilder#QuoteSuffix` property values

Return

`string`: Character string enclosed by the `HiRDBCommandBuilder#QuotePrefix` and `HiRDBCommandBuilder#QuoteSuffix` property values

Description: Returns a specified character string enclosed by the `HiRDBCommandBuilder#QuotePrefix` value and the `HiRDBCommandBuilder#QuoteSuffix` property value.

Exception: HiRDBException

(e) RefreshSchema

`void RefreshSchema ()`

Return: `void`

Description: Updates database schema information to create an `INSERT`, `UPDATE`, or `DELETE` statement.

void RefreshSchema (string)

Argument

`string s TableName`: Table name

Return: `void`

Description: Updates database schema information to create the `INSERT`, `UPDATE`, or `DELETE` statement.

Exception: `HiRDBException`

(f) UnquoteIdentifier

string UnquoteIdentifier (string)

Argument

`string quotedIdentifier`: Character string enclosed by the `HiRDBCommandBuilder#QuotePrefix` and `HiRDBCommandBuilder#QuoteSuffix` property values

Return

`string`: Value obtained by removing the `HiRDBCommandBuilder#QuotePrefix` property value at the beginning and the `HiRDBCommandBuilder#QuoteSuffix` property value at the end of a character string

Description: Returns the value obtained by removing the `HiRDBCommandBuilder#QuotePrefix` property value at the beginning and the `HiRDBCommandBuilder#QuoteSuffix` property value at the end of a specified character string.

Exception: `HiRDBException`

16.5.3 HiRDBConnection

(1) Constructor

(a) HiRDBConnection

`void HiRDBConnection ()`

Description: Initializes a new instance of `HiRDBConnection`.

`void HiRDBConnection (string)`

Argument

`string ConnectionString`: Character string storing the connection settings (`ConnectionString` property)

Description: Specifies a connection character string and initializes a new instance of the HiRDBConnection class.

(2) Properties

(a) ConnectionString

Type: string

Default value: ""

Description: Acquires or sets the character string that is used to open a database.

Exception: HiRDBException

For this property, you must specify one string-type argument. The character string to be specified is called a *connection character string*. This is the same type of connection character string as those used for Connection in ADO and ADO.NET. The following table lists and describes the character strings that can be specified:

Character string	Description
• datasource • dsn • env	Settings for the registry to be used. Specify the name of the environment variable group that was created using the tool for registering HiRDB client environment variables.
• uid • userid	Authorization identifier used for DB connection
• password • Pwd	Password to be used for the database connection
• PD*	Settings in the client environment definition

If nothing is specified, the default setting (HiRDB.ini) is used to establish the connection. If a client environment variable group name is available, this name is used. If the authorization identifier, password, and client environment definition are specified, their use takes precedence. This character string is not case sensitive. To distinguish upper-case letters from lower-case letters, enclose the applicable part in quotation marks. All spaces and tabs are ignored (except those enclosed in quotation marks).

If the specified character string is not one of the connection character strings listed above, an exception occurs. However, for Provider, the specified invalid character string is ignored; no exception occurs. This maintains compatibility with OleDb Data Provider in the DataProvider layer.

(b) ConnectionTimeout

Type: int

Default value:

When the supported ADO.NET version is 1.1: `15`

When the supported ADO.NET version is 2.0: `0`

Description: Acquires the wait time for establishing a connection before retries are cancelled and an error is generated (read only).

(c) Database

Type: `string`

Default value: `""`

Description: Acquires the name of the current database or the database that is used when a connection is established (read only).

(d) DataSource

Type: string

Default value: `""`

Description: Acquires the name of the connected database server (read only).

(e) ServerVersion

Type: string

Default value: `""`

Description: Acquires the version of the connected server (read only).

This member returns the version in the normalized format that allows comparison by using `String.Compare()`. The format of the version is as follows:

XX.YY.ZZZZ

XX: Major version

YY: Minor version

ZZZZ: Always `0000`

(f) LifeTime

Type: `int`

Default value: `60`

Description: Acquires or sets the time remaining before actual disconnection occurs.

Exception: `HiRDBException`

(g) Pooling

Type: `bool`

Default value: `true`

Description: Acquires or sets whether or not pooling is to be performed. If pooling is performed, the value is `true`; if not, the value is `false`.

Exception: `HiRDBException`

(h) State

Type: `System.Data.ConnectionState`

Default value: `ConnectionState.Closed`

Description: Acquires the current connection status (read only).

(3) Methods

(a) BeginTransaction

`BeginTransaction ()`

Return

`HiRDBTransaction`: Object representing a new transaction

Description: Starts the database transaction.

Exception: `HiRDBException`

`BeginTransaction (System.Data.IsolationLevel)`

Argument

`System.Data.IsolationLevel`: One of the `IsolationLevel` values

Return

`HiRDBTransaction`: Object representing a new transaction

Description: Starts the database transaction using the specified `IsolationLevel` value.

Exception: `HiRDBException`

(b) ChangeDatabase

`void ChangeDatabase (string)`

Argument

`string databaseName`: Name of the database to be changed

Return: `void`

Description: Changes the current database for the open `HiRDBConnection` object.

Exception: `HiRDBException`

(c) Clone

`HiRDBConnection Clone ()`

Return: Returns null unconditionally.

Description: Returns null unconditionally.

(d) Close

`void Close ()`

Return: `void`

Description: Closes the connection to the database.

(e) CreateCommand

`Hitachi.HiRDB.HiRDBCommand CreateCommand ()`

Return

`HiRDBCommand`: `HiRDBCommand` object

Description: Creates and returns the `HiRDBCommand` object associated with the connection.

(f) EnlistTransaction

`void EnlistTransaction (Transaction)`

Argument

`transaction`: Existing target `Transaction` object

Return: `void`

Description: Registers in the specified transaction.

Exception: HiRDBException

(g) GetSchema

DataTable GetSchema ()

Return: `DataTable` object

Description: Returns schema information.

DataTable GetSchema (string)

Argument

`collectionName`: Name of the schema to be returned

Return: `DataTable` object

Description: Returns schema information.

DataTable GetSchema (string, string[])

Argument

`collectionName`: Name of the schema to be returned

`restrictionValues`: Restriction value for the requested schema

Return: `DataTable` object

Description: Returns schema information.

(h) Open

`void Open ()`

Return: `void`

Description: Opens the database connection with the settings specified in the `ConnectionString` property of the `HiRDBConnection` object.

Exception: `HiRDBException`

16.5.4 HiRDBDataAdapter

(1) Constructor

(a) HiRDBDataAdapter

`void HiRDBDataAdapter ()`

Description: Initializes a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (Hitachi.HiRDB.HiRDBCommand)`

Argument

`HiRDBCommand selectCommand`: `HiRDBCommand` object representing the `SQL SELECT` statement (`SelectCommand` property)

Description: Uses the specified `HiRDBCommand` to initialize a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (string, Hitachi.HiRDB.HiRDBConnection)`

Arguments

`string selectCommandText`: `SQL SELECT` statement

`HiRDBConnection selectConnection`: `HiRDBConnection` object representing the connection

Description: Uses the `HiRDBConnection` specifying the `SQL SELECT` statement to create `HiRDBCommand` (`SelectCommand` property). This constructor initializes a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (string, string)`

Arguments

`string selectCommandText`: `SQL SELECT` statement

`string selectConnectionString`: connection character string

Description: Uses a connection character string to create `HiRDBConnection`. The constructor then uses the created `HiRDBConnection` to create `HiRDBCommand` (`SelectCommand` property). This constructor initializes a new instance of the `HiRDBDataAdapter` class.

(2) Properties

(a) DeleteCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for deleting records from a data set.

(b) InsertCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for inserting new records in a database.

(c) SelectCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for selecting records in a database.

(d) UpDateCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for updating records in a database.

(3) Events

(a) RowUpdated

Type: HiRDBRowUpdatedEventHandler

Description: This event occurs after a command has been executed on the data source during update processing.

(b) RowUpdating

Type: HiRDBRowUpdatingEventHandler

Description: This event occurs before a command is executed on the data source during update processing.

16.5.5 HiRDBDataReader

(1) Constructor

`HiRDBDataReader`

Description: To create `HiRDBDataReader`, you must call the `ExecuteReader` method of the `HiRDBCommand` object without directly using the constructor.

(2) Properties

(a) Depth

Type: `int`

Default value: `0`

Description: Acquires the value indicating the nesting level of the current row.

(b) FieldCount

Type: `int`

Description: Acquires the number of columns in the current row.

(c) HasRows

Type: bool

Default value: false

Description: Acquires a value indicating whether the corresponding `HiRDBDataReader` contains at least one row. If `HiRDBDataReader` contains any rows at all, the value is `true`; if it doesn't, the value is `false`.

(d) IsClosed

Type: `bool`

Default value: `false`

Description: Acquires the value indicating whether or not the data reader is closed. If the data reader is closed, the value is `true`; if not, the value is `false`.

(e) Item

Item[string]

Type: Object this[string name]

Description: Acquires data by treating the `HiRDBDataReader` object as an array.

Item[int]

Type: Object this[int ordinal]

Description: Acquires data by treating the `HiRDBDataReader` object as an array.

Item[int, int]

Type: Object this[int colIdx, int elmIdx]

Description: Acquires data by treating the `HiRDBDataReader` object as an array.

(f) RecordsAffected

Type: `int`

Default value: `0`

Description: Acquires the number of rows changed, inserted, or deleted by execution of an SQL statement.

(g) VisibleFieldCount

Type: int

Description: Acquires the number of hidden rows for `HiRDBDataReader`.

(3) Methods

(a) Close

`void Cancel ()`

Return: `void`

Description: Closes the `HiRDBDataReader` object.

(b) GetBoolean

`bool GetBoolean (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`bool`: Column value

Description: Acquires the value of the specified column as a Boolean value.

Exception: `HiRDBException`

(c) GetByte

`byte GetByte (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`byte`: Unsigned 8-bit integer value in the specified column

Description: Acquires an unsigned 8-bit integer value in the specified column.

Exception: `HiRDBException`

(d) GetBytes

```
long GetBytes (int, long, byte[ ], int,int)
```

Arguments

`int i`: Ordinal number of the column that begins at 0

`long fieldOffset`: Index of the row where the read operation begins

`byte[] buffer`: Buffer for reading byte streams

`int bufferoffset`: Index of `buffer` where the read operation begins

`int length`: Number of bytes to be read

Return

`long`: Number of bytes actually read

Description: Reads a byte stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.

Exception: `HiRDBException`

(e) GetChar

```
char GetChar (int)
```

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`char`: Character value in the specified column

Description: Acquires the character string value in the specified column.

Exception: `HiRDBException`

(f) GetChars

```
long GetChars (int, long,char[ ], int, int)
```

Arguments

`int i`: Ordinal number of the column that begins at 0

`long fieldOffset`: Index of the row where the read operation begins

`char[] buffer`: Buffer for reading byte streams

`int bufferoffset`: Index of `buffer` where the read operation begins

`int length`: Number of bytes to be read

Return

`long`: Number of characters actually read

Description: Reads a character stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.

Exception: `HiRDBException`

(g) GetData

`GetData (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return: Currently not supported.

Description: The purpose of this member is to support the .NET Framework infrastructure. It cannot be used directly in a unique coding that has been created.

(h) GetDataTypeName

`string GetDataTypeName (int)`

Argument

`int i`: Index of the field to be searched

Return

`string`: Data-type information for the specified field

Description: Acquires data-type information for the specified field.

Exception: `HiRDBException`

(i) GetDateTime

`System.DateTime GetDateTime (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.DateTime`: Date and time data value in the specified field

Description: Acquires or sets the date and time data value in the specified field.

Exception: `HiRDBException`

(j) GetDecimal

```
decimal GetDecimal (int)
```

Argument

`int i`: Index of the field to be searched

Return

`decimal`: Fixed position value in the specified field

Description: Acquires the fixed position value in the specified field.

Exception: `HiRDBException`

(k) GetDouble

```
double GetDouble (int)
```

Argument

`int i`: Index of the field to be searched

Return

`double`: Double-precision floating-point number in the specified field

Description: Acquires the double-precision floating-point number in the specified field.

Exception: `HiRDBException`

(l) GetEnumerator

```
System.Collections.IEnumerator GetEnumerator ()
```

Return

`System.Collections.IEnumerator`: `IEnumerator` that can be used to perform iterative operation on a collection

Description: Returns the enumerator that can perform iterative operation on a collection.

(m) GetFieldArrayCount

```
int GetFieldArrayCount (int)
```

Argument

`int i`: Index of the field to be searched

Return

`int`: Size of field array

Description: Acquires the size of field array.

Exception: `HiRDBException`

(n) GetFieldType

`System.Type GetFieldType (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.Type`: `Type` information corresponding to the type of `object` that is returned from `GetValue`

Description: Acquires `Type` information corresponding to the type of `Object` that is returned from `GetValue`.

Exception: `HiRDBException`

(o) GetFloat

`float GetFloat (int)`

Argument

`int i`: Index of the field to be searched

Return

`float`: Single-precision floating-point number in the specified field

Description: Acquires the single-precision floating-point number in the specified field.

Exception: `HiRDBException`

(p) GetGuid

`System.Guid GetGuid (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.Guid`: `GUID` value of the specified field

Description: Returns the `GUID` value of the specified field.

(q) GetInt16

`short GetInt16 (int)`

Argument

`int i`: Index of the field to be searched

Return

`short`: Signed 16-bit integer value in the specified field

Description: Acquires a signed 16-bit integer value in the specified field.

Exception: `HiRDBException`

(r) GetInt32

`int GetInt32 (int)`

Argument

`int i`: Index of the field to be searched

Return

`int`: Signed 32-bit integer value in the specified field

Description: Acquires a signed 32-bit integer value in the specified field.

Exception: `HiRDBException`

(s) GetInt64

`long GetInt64 (int)`

Argument

`int i`: Index of the field to be searched

Return

`long`: Signed 64-bit integer value in the specified field

Description: Acquires a signed 64-bit integer value in the specified field.

Exception: `HiRDBException`

(t) GetName

`string GetName (int)`

Argument

`int i`: Index of the field to be searched

Return

`string`: Field name (if there is no value to be returned, returns the `null` character string (""))

Description: Acquires the name of the field to be searched.

Exception: `HiRDBException`

(u) GetOrdinal

`int GetOrdinal (string)`

Argument

`string name`: Name of the field to be searched

Return

`int`: Index of the specified field

Description: Returns the index of the specified field.

Exception: `HiRDBException`

(v) GetProviderSpecificFieldType

Object GetProviderSpecificFieldType (int)

Argument

`int ordinal`: Ordinal number of the column, beginning at 0

Return

`Object`: Data type of the specified column

Description: Acquires the data type of a specified column. Because HiRDB.NET Data Provider does not support any unique data types, it returns the `Type` object of the data type provided by the common language runtime of .NET Framework. Its operation is the same as for the `HiRDBDataReader#GetFieldType` method.

Exception: HiRDBException

(w) GetProviderSpecificValue

Object GetProviderSpecificValue (int)

Argument

`int ordinal`: Ordinal number of the column, beginning at 0

Return

`Object`: Object that has the value of the specified column

Description: Acquires the value of a specified column as an instance of `Object`.

Exception: HiRDBException

(x) GetProviderSpecificValues

int GetProviderSpecificValues (Object[])

Argument

values: Object array that is the target of a copy operation on attribute columns (Object-type array that stores the column data for the current rows)

Return

int: Number of Object instances in the array

Description: Acquires all attribute columns in the current collection of rows.

Exception: HiRDBException

(y) GetSchemaTable

`System.Data.DataTable GetSchemaTable ()`

Return

System.Data.DataTable: DataTable that describes column metadata

Description: Returns the DataTable that describes HiRDBDataReader's column metadata.

Exception: HiRDBException

(z) GetString

`string GetString (int)`

Argument

int i: Index of the field to be searched

Return

string: Character string in the specified field

Description: Acquires a character string in the specified field.

Exception: HiRDBException

(aa) GetValue

`object GetValue (int)`

Argument

int i: Index of the field to be searched

Return

object: Object for storing the returned field value, if any

Description: Returns a value in the specified field.

Exception: `HiRDBException`

`object GetValue (int, int)`

Arguments

`int i`: Index of the field to be searched

`int j`: Index of the field to be searched

Return

`object`: `Object` for storing the returned field value, if any

Description: Returns a value in the specified field (for array).

Exception: `HiRDBException`

(ab) GetValues

`int GetValues (object[ ])`

Argument

`object values`: `Object` array which is the target of a copy operation on the attribute field

Return

`int`: Number of `Object` instances in array

Description: Acquires all attribute fields in the current record collection.

(ac) IsDBNull

`bool IsDBNull (int)`

Argument

`int i`: Index of the field to be searched

Return

`bool`: If the specified field is set to `null`, the value is `true`; if not, the value is `false`.

Description: Returns a value indicating whether or not the specified field is set to `null`.

Exception: `HiRDBException`

(ad) NextResult

`bool NextResult ()`

Return

`bool`: If there are further rows, the value is `true`; if not, the value is `false`.

Description: Advances the data reader to the next result when the result of a batch SQL statement is read.

Exception: `HiRDBException`

(ae) Read

```
bool Read ()
```

Return

`bool`: If there are further rows, the value is `true`; if not, the value is `false`.

Description: Advances `HiRDBDataReader` to the next record.

Exception: `HiRDBException`

16.5.6 HiRDBException

(1) Properties

(a) ErrorCode

Type: `int`

Default value: `0`

Description: Acquires an error code as an `int`.

(b) Message

Type: `String`

Default value: `""`

Description: Acquires text with a complete error.

16.5.7 HiRDBParameter

(1) Constructor

(a) HiRDBParameter

```
void HiRDBParameter ()
```

Description: Initializes a new instance of the `HiRDBParameter` class.

```
void HiRDBParameter (string, object)
```

Arguments

`string name`: Name of the parameter to be allocated (`ParameterName` property)

`object value`: Value of the new `HiRDBParameter` object (`Value` property)

Description: Specifies the parameter name and HiRDBParameter object to initialize a new instance of the HiRDBParameter class.

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType)

Arguments

string name: Name of the parameter to be allocated (ParameterName property)

Hitachi.HiRDB.HiRDBType dataType: One of the HiRDBType values (HiRDBType property)

Description: Specifies a parameter name and data type to initialize a new instance of the HiRDBParameter class.

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int)

Arguments

string name: Name of the parameter to be allocated (ParameterName property)

Hitachi.HiRDB.HiRDBType dataType: One of the HiRDBType values (HiRDBType property)

int size: Parameter definition length (Size property)

Description: Uses a parameter name, data type, and length to initialize a new instance of the HiRDBParameter class.

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int, string)

Arguments

string name: Name of the parameter to be allocated (ParameterName property)

Hitachi.HiRDB.HiRDBType dataType: One of the HiRDBType values (HiRDBType property)

int size: Parameter definition length (Size property)

string srcColumn: Name of the source column (SourceColumn property)

Description: Specifies a parameter name, data type, length, and source column name to initialize a new instance of the HiRDBParameter class.

void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int, System.Data.ParameterDirection, byte, byte, string, System.Data.DataRowVersion, object)

Arguments

string parameterName: Parameter name (ParameterName property)

Hitachi.HiRDB.HiRDBType dataType: One of the HiRDBType values (HiRDBType property)

int size: Parameter definition length (Size property)

System.Data.ParameterDirection direction: One of the ParameterDirection values (Direction property)

byte precision: Total length in digits used to resolve Value (Precision property)

byte scale: Length of the fractional part in digits used to resolve Value (Scale property)

string srcColumn: Name of the source column (SourceColumn property)

System.Data.DataRowVersion srcVersion: One of the DataRowVersion values (SourceVersion property)

object value: Object which is the value of HiRDBParameter (Value property)

Description: Specifies a parameter name, data type, length, source column name, parameter direction, precision of numeric value, and other properties to initialize a new instance of the HiRDBParameter class.

(2) Properties

(a) DbType

Type: System.Data.DbType

Default value: DbType.String

Description: Acquires or sets DbType for a parameter. When DbType is to be set, this member sets the corresponding data type in the HiRDBType property according to *Table 16-3*.

(b) Direction

Type: System.Data.ParameterDirection

Default value: ParameterDirection.Input

Description: Acquires or sets a value indicating whether the parameter is input only, output only, bidirectional, or the stored procedure's return value.

(c) HiRDBType

Type: `Hitachi.HiRDB.HiRDBType`

Default value: `HiRDBType.MVarChar`

Description: Acquires or sets an enumeration indicating the data type in HiRDB. When the enumeration is to be set, this member sets the corresponding data type in the `DbType` property according to *Table 16-4*.

`HiRDBType` enumeration:

> `Integer`, `SmallInt`, `Decimal`, `Float`, `SmallFlt`, `Char`, `VarChar`, `NChar`, `NVarChar`, `MChar`, `MVarChar`, `Date`, `Time`, `TimeStamp`, `IntervalYearToDay`, `IntervalHourToSecond`, `Blob`, `Binary`

(d) IsNullable

Type: `bool`

Default value: `true` (fixed)

Description: Acquires a value indicating whether or not the parameter accepts the `null` value (read only). If the `null` value is accepted, the value is `true`; if not, the value is `false`.

(e) ParameterName

Type: `string`

Default value: `""`

Description: Acquires or sets the name of the `HiRDBParameter`.

(f) Precision

Type: `byte`

Default value: `0`

Description: Acquires or sets the number of significant digits (including decimal places) in the definition length of a `DECIMAL`-type parameter.

If the `HiRDBType` property value is `HiRDBType.Decimal`:

Specify the precision for a `DECIMAL`-type column on the server that corresponds to the parameter.

If the `HiRDBType` property value is not `HiRDBType.Decimal`:

The specified setting is ignored.

(g) Repetition

Type: `short`

Default value: `1`

Description: Acquires or sets an array structure in HiRDB. The value 1 indicates that the target column is a non-repetition column, and 2 or greater is the maximum number of elements in the repetition column.

(h) Scale

Type: `byte`

Default value: `0`

Description: Acquires or sets the number of decimal places in the definition length of a `DECIMAL`-type parameter.

If the `HiRDBType` property value is `HiRDBType.Decimal`:

Specify the decimal places for a `DECIMAL`-type column on the server that corresponds to the parameter.

If the `HiRDBType` property value is not `HiRDBType.Decimal`:

The specified setting is ignored.

(i) Size

Type: `int`

Default value: `0`

Description: Acquires or sets a parameter definition length. If the length is fixed (such as numeric type or date/time type), specify `0`. If the length is variable (such as character string type), specify the number of bytes to be stored in the table or the maximum column length.

For `TIMESTAMP (DateTime)`, this value is the number of digits in the fractional part.

Note

Note that if the entered character string is longer than the size specified for the `Size` property, only up to the specified number of characters are stored. No exception occurs.

(j) SourceColumn

Type: `string`

Default value: `""`

Description: Acquires or sets the name of the source column that has been assigned to `DataSet` and is used to read or return `Value`.

(k) SourceColumnNullMapping

Type: bool

Default value: true

Description: Acquires or sets a value that indicates whether the column of `DataTable` object corresponding to the parameter permits the NULL value.

(l) SourceVersion

Type: `System.Data.DataRowVersion`

Default value: `DataRowVersion.Default`

Description: Acquires or sets the `DataRowVersion` that is used to read `Value`.

(m) Value

Type: `object`

Default value: `null`

Description: Acquires or sets a parameter value.

(3) Methods

(a) Clone

`object Clone ()`

Return

`object`: New object which is a copy of this instance

Description: Creates a new object which is a copy of the current instance.

(b) ResetDbType

void ResetDbType ()

Return: `void`

Description: Resets the `DbType` property to its initial value.

16.5.8 HiRDBParameterCollection

(1) Constructor

(a) HiRDBParameterCollection

`void HiRDBParameterCollection ()`

Description: Initializes a new instance of the `HiRDBParameterCollection` class.

(2) Properties

(a) Count

Type: `int`

Default value: `0`

Description: Acquires the number of `HiRDBParameter` objects stored in `HiRDBParameterCollection` (read only).

(b) IsFixedSize

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether the size of `HiRDBParameterCollection` is fixed (read only). The value is always `false`.

(c) IsReadOnly

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether or not `HiRDBParameterCollection` is read only (read only). The value is always `false`.

(d) IsSynchronized

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether or not an access to `HiRDBParameterCollection` is synchronized (thread-safe) (read only). The value is always `false`.

(e) Item

Item[int]

Type: HiRDBParameter this[int index]

Description: Acquires the `HiRDBParameter` object for a specified index or sets the `HiRDBParameter` object in a specified index.

Item[string]

Type: HiRDBParameter this[string parameterName]

Description: Acquires the `HiRDBParameter` object that has the parameter name specified in the argument, or sets a new `HiRDBParameter` object in the index of the `HiRDBParameter` object that has the parameter name specified in the argument.

(f) SyncRoot

Type: `object`

Default value: `null`

Description: Acquires an object that can be used to synchronize an access to

HiRDBParameterCollection (read only).

(3) Methods

(a) Add

int Add (object)

Argument

object value: HiRDBParameter object to be added to HiRDBParameterCollection

Return

int: Index in the new HiRDBParameter object's collection

Description: Adds items to HiRDBParameterCollection.

int Add (Hitachi.HiRDB.HiRDBParameter)

Argument

HiRDBParameter value: HiRDBParameter to be added to HiRDBParameterCollection

Return

int: Index of the new HiRDBParameter

Description: Adds items to HiRDBParameterCollection.

int Add (string, object)

Arguments

string parameterName: Parameter name

object parameterValue: Parameter value

Return

int: Index of the new HiRDBParameter

Description: Specifies the name and value of the parameter to add items to HiRDBParameterCollection.

int Add (string, HiRDBType)

Arguments

string parameterName: Parameter name

HiRDBType dataType: One of the HiRDBType values

Return

int: Index of the new HiRDBParameter

Description: Specifies the name and data type of the parameter to add items to HiRDBParameterCollection.

`int Add (string, HiRDBType, int)`

Arguments

`string parameterName`: Parameter name

`HiRDBType dataType`: One of the `HiRDBType` values

`int size`: Parameter size

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name, data type, and size of the parameter to add items to `HiRDBParameterCollection`.

`int Add (string, HiRDBType, int, string)`

Arguments

`string parameterName`: Parameter name

`HiRDBType dataType`: One of the `HiRDBType` values

`int size`: Parameter size

`string srcColumn`: Name of the source column

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name, data type, size, and source column of the parameter to add items to `HiRDBParameterCollection`.

(b) AddRange

void AddRange(Array)

Argument

`values`: Array in the `HiRDBParameter` object to be added to `HiRDBParameterCollection`

Return: void

Description: Adds an array in the specified `HiRDBParameter` object to `HiRDBParameterCollection`.

Exception: HiRDBException

void AddRange(HiRDBParameter[])

Argument

`value`: Array in the `HiRDBParameter` object to be added to `HiRDBParameterCollection`

Return: void

Description: Adds an array in the specified `HiRDBParameter` object to `HiRDBParameterCollection`.

Exception: HiRDBException

(c) Clear

`void Clear ()`

Return: `void`

Description: Deletes all items from `HiRDBParameterCollection`.

(d) Contains

`bool Contains (object)`

Argument

`object value`: `Object` that is searched for in `HiRDBParameterCollection`

Return

`bool`: If `Object` is in `HiRDBParameterCollection`, the value is `true`; if not, the value is `false`.

Description: Acquires a value indicating whether or not `HiRDBParameter` is in the collection.

Exception: HiRDBException

bool Contains (HiRDBParameter)

Argument

`HiRDBParameter value`: `HiRDBParameter` object to be searched for in `HiRDBParameterCollection`

Return

`bool`: If `Object` is in `HiRDBParameterCollection`, the value is `true`; if not, the value is `false`.

Description: Acquires a value indicating whether `HiRDBParameter` is in the collection.

Exception: HiRDBException

`bool Contains (string)`

Argument

`string parameterName`: Parameter name

Return

`bool`: If the parameter is stored in the collection, the value is `true`; if not, the value is `false`.

Description: Acquires a value indicating whether or not `HiRDBParameter` is in the collection.

Exception: HiRDBException

(e) CopyTo

`void CopyTo (System.Array, int)`

Arguments

`System.Array array`: One-dimensional `Array` to which elements are copied from `HiRDBParameterCollection`

`int index`: Index number, beginning at 0, at the location where `value` is inserted

Return: `void`

Description: Copies the elements of `HiRDBParameterCollection` to `Array` using `Array`'s specific index as the start position.

(f) GetEnumerator

`System.Collections.IEnumerator GetEnumerator ()`

Return

`System.Collections.Ienumerator`: `IEnumerator` that can be used to perform iteration processing on a collection

Description: Returns the enumerator that can perform iterative operation on a collection.

(g) IndexOf: overload

`int IndexOf (string)`

Argument

`string parameterName`: Parameter name

Return

`int`: Location of `HiRDBParameterCollection` in the collection that begins at 0

Description: Acquires the location of `HiRDBParameter` in a collection.

Exception: `HiRDBException`

`int IndexOf (object)`

Argument

`object value`: `Object` that is searched for in `HiRDBParameterCollection`

Return

`int`: If the object is in the list, the value is the index of `value`; if not, the value is `-1`.

Description: Acquires the location of `HiRDBParameter` in a collection.

(h) Insert

void Insert(int, object)

Arguments

`int index`: Index number, beginning at 0, at the location where `value` is inserted

`object value`: `HiRDBParameter` to be added to `HiRDBParameterCollection`

Return: `void`

Description: Inserts an item at the specified location in `HiRDBParameterCollection`.

Exception: HiRDBException

`void Insert (int, Hitachi.HiRDB.HiRDBParameter)`

Arguments

`int index`: Index number, which begins at 0, at the location where `value` is inserted

`HiRDBParameter value`: `HiRDBParameter` to be added to `HiRDBParameterCollection`

Return: `void`

Description: Inserts an item at the specified location in `HiRDBParameterCollection`.

Exception: HiRDBException

(i) Remove

`void Remove (object)`

Argument

`object value`: `HiRDBParameter` to be deleted from `HiRDBParameterCollection`

Return: `void`

Description: Deletes the first occurrence of the specified object in `HiRDBParameterCollection`.

(j) RemoveAt

`void RemoveAt (string)`

Argument

`string parameterName`: Parameter name

Return: `void`

Description: Deletes `HiRDBParameter` from a collection.

Exception: `HiRDBException`

`void RemoveAt (int)`

Argument

`int index`: Index of the item to be deleted that begins at 0

Return: `void`

Description: Deletes `HiRDBParameter` from a collection.

16.5.9 HiRDBProviderFactory

(1) Constructor

(a) HiRDBProviderFactory

HiRDBProviderFactory()

Description: Initializes a new instance of the `HiRDBProviderFactory` class.

(2) Fields

(a) Instance

Type: HiRDBProviderFactory

Description: Holds an instance of `HiRDBProviderFactory` (read only).

(3) Properties

(a) CanCreateDataSourceEnumerator

Type: `bool`

Description: Indicates whether a class derived from the `DbDataSourceEnumerator` class is to be supported (read only). If it is supported, the value is `true`; if not, the value is `false`.

(4) Methods

(a) CreateCommand

DbCommand CreateCommand ()

Return: `HiRDBCommand` object

Description: Creates and returns the `HiRDBCommand` object.

(b) CreateCommandBuilder

DbCommandBuilder CreateCommandBuilder ()

Return: `HiRDBCommandBuilder` object

Description: Creates and returns a `HiRDBCommandBuilder` object.

(c) CreateConnection

DbConnection CreateConnection ()

Return: `HiRDBConnection` object

Description: Creates and returns a `HiRDBConnection` object.

(d) CreateConnectionStringBuilder

DbConnectionStringBuilder CreateConnectionStringBuilder ()

Return: `DbConnectionStringBuilder` object

Description: Creates and returns a `DbConnectionStringBuilder` object.

(e) CreateDataAdapter

DbDataAdapter CreateDataAdapter ()

Return: `HiRDBDataAdapter` object

Description: Creates and returns a `HiRDBDataAdapter` object.

(f) CreateDataSourceEnumerator

DbDataSourceEnumerator CreateDataSourceEnumerator ()

Return: There is no return value because an exception always occurs.

Description: This member is not supported because no HiRDB enumeration type is provided. It returns `NotSupportedException` unconditionally.

Exception: System.NotSupportedException

(g) CreateParameter

DbParameter CreateParameter ()

Return: `HiRDBParameter` object

Description: Creates and returns a `HiRDBParameter` object.

16.5.10 HiRDBRowUpdatedEventArgs

(1) Constructor

(a) HiRDBRowUpdatedEventArgs

```
void HiRDBRowUpdatedEventArgs (System.Data.DataRow,
System.Data.IdbCommand, System.Data.StatementType,
System.Data.Common.DataTableMapping)
```

Arguments

`System.Data.DataRow dataRow`: `DataRow` that was sent through `Update`

`System.Data.IDbCommand command`: `IDbCommand` that was executed when `Update` was called

`System.Data.StatementType statementType`: Type of SQL statement that was executed

`System.Data.Common.DataTableMapping tableMapping`: `DataTableMapping` that was sent through `Update`

Description: Initializes a new instance of the `HiRDBRowUpdatedEventArgs` class.

(2) Properties

(a) Command

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires the `HiRDBCommand` that is executed when `Update` is called (read only).

16.5.11 HiRDBRowUpdatingEventArgs

(1) Constructor

(a) HiRDBRowUpdatingEventArgs

```
void HiRDBRowUpdatingEventArgs (System.Data.DataRow,
```

System.Data.IDbCommand, System.Data.StatementType, System.Data.Common.DataTableMapping)

Arguments

System.Data.DataRow dataRow: DataRow that executes Update

System.Data.IDbCommand command: IDbCommand that is executed when Update is called

System.Data.StatementType statementType: Type of SQL statement to be executed

System.Data.Common.DataTableMapping tableMapping: DataTableMapping that is sent through Update

Description: Initializes a new instance of the HiRDBRowUpdatingEventArgs class.

(2) Properties

(a) Command

Type: HiRDBCommand

Default value: null

Description: Acquires or sets the HiRDBCommand that is executed during Update processing.

16.5.12 HiRDBTransaction

(1) Properties

(a) Connection

Type: HiRDBConnection

Default value: null

Description: Specifies the HiRDBConnection object used to associate a transaction (read only).

(b) IsCompleted

Type: bool

Default value: false

Description: Acquires a value indicating whether or not the transaction is completed (read only). If the transaction is completed, the value is true; if not, the value is false.

(c) IsolationLevel

Type: System.Data.IsolationLevel

Default value:

When the supported ADO.NET version is 1.1:
`IsolationLevel.ReadCommitted`

When the supported ADO.NET version is 2.0:
`IsolationLevel.RepeatableRead`

Description: Specifies this transaction's `IsolationLevel` (read only).

(2) Methods

(a) Commit

`void Commit ()`

Return: `void`

Description: Commits a database transaction.

Exception: `HiRDBException`

(b) Rollback

`void Rollback ()`

Return: `void`

Description: Rolls back a database transaction from the hold status.

Exception: `HiRDBException`

16.6 Notes about HiRDB.NET Data Provider

16.6.1 Placing in global assembly cache

(1) Notes about building and executing UAPs

When a UAP is built, HiRDB.NET Data Provider's DLL files are not copied to the executable file storage directory. Instead, when a UAP is executed, the DLL files in the global assembly cache are referenced. For this reason, there is no need to have HiRDB.NET Data Provider's DLL files in the same directory as the UAP when the UAP executes.

Publisher policies for HiRDB.NET Data Provider's DLL files are also placed in the global assembly cache. These publisher policies contain rules used during UAP execution for redirecting the version of HiRDB.NET Data Provider's DLL files for the installed version of HiRDB client. Therefore, when a UAP executes, it references the current version of HiRDB.NET Data Provider and there is no need to rebuild existing UAPs.

The following HiRDB.NET Data Provider version is referenced during UAP execution:

- 09-00 or earlier

  UAPs reference the version of HiRDB.NET Data Provider that has been built. To use a newer version of HiRDB.NET Data Provider, you must first install the newer version of HiRDB client and then rebuild the UAPs.

- 09-01 or later

  UAPs always reference the HiRDB.NET Data Provider on the installed HiRDB client.

(2) How to avoid applying the publisher policies

The Windows default settings provide that the publisher policies will be applied. The user can add a setting that will prevent redirecting of the version based on the publisher policies that have been placed on the HiRDB client.

This subsection describes how to avoid applying the publisher policies. For details, see the documentation provided by Microsoft.

(a) Using an application configuration file

When you develop a UAP, you can create an application configuration file ({*executable-file-name*}`.config`) and place it together with the application. By adding the setting for disabling publisher policies in the application configuration file (`<publisherPolicy apply=no/>`*element*), you can avoid the application of publisher policies for each application.

(b) Using the Control Panel

From **Control Panel**, choose **Administrative Tools**, then **Microsoft .NET Framework 1.1 Configuration** or **Microsoft .NET Framework 2.0 Configuration**. You can avoid application of publisher policies for each application by changing an application property.

16.6.2 Notes about individual methods and properties

The following table provides notes about individual methods and properties of HiRDB.NET Data Provider.

Table 16-2: Notes about individual methods and properties

Object	Method or property	Details
HiRDBCommand	CommandTimeout property	The setting is ignored because the timeout value during execution depends on the settings in the client environment definition (PDCWAITTIME, PDSWAITTIME, PDSWATCHTIME).
	Cancel method	System.NotSupportedException is returned because there is no cancellation function.
	ExecuteReader method	When the CommandBehavior.KeyInfo, CommandBehavior.SchemaOnly, or CommandBehavior.SequentialAccess argument is specified, it is treated as CommandBehavior.Default because a function for acquiring only column or primary key information is not available.
	UpdatedRowSource property	When UpdatedRowSource.Both or UpdatedRowSource.FirstReturnedRecord is specified, HiRDBException is returned because there is no batch query function that returns rows.
HiRDBCommandBuilder	CatalogLocation property	No function is affected by this property value because a cataloging function is not available.
	CatalogSeparator property	No function is affected by this property value because a cataloging function is not available.
	SetAllValues property	This property value is always true and the UPDATE statement includes all column information.
HiRDBConnection	ConnectionTimeout property	The following value is always returned: • When the supported ADO.NET version is 1.1: 15 • When the supported ADO.NET version is 2.0: 0
	Database property	The null character always results because there is no function for acquiring database names.

Object	Method or property	Details
	State property	ConnectionState.Connecting, ConnectionState.Executing, ConnectionState.Fetching, or ConnectionState.Broken will never result because this property is a reserved value for future product versions.
	BeginTransaction method	IsolationLevel is ignored, if specified, because this method is set for each SQL statement or acquired from HiRDB environment variables. No more than one transaction can be executed using a single connection.
	ChangeDatabase method	System.NotSupportedException is returned because a function for changing the connected database is not available.
	EnlistTransaction method	System.NotSupportedException is returned because a distributed transaction function is not available.
	GetSchema method	Schema information is not returned. An empty DataTable object is always returned.
HiRDBDataReader	Depth property	Always 0 because there is no hierarchy concept.
	VisibleFieldCount property	The value is the same as for FieldCount because a column specified in the retrieval SQL statement is always acquired.
	GetBoolean method	NotSupportedException is returned because there is no corresponding type.
	GetByte method	NotSupportedException is returned because there is no corresponding type.
	GetChar method	NotSupportedException is returned because there is no corresponding type.
	GetData method	NotSupportedException is returned because there is no corresponding type.
	GetGuid method	NotSupportedException is returned because there is no corresponding type.
	GetProviderSpecificFieldType	An object with the data type provided in .NET Framework's common language runtime is returned because no unique data types are supported.
	GetProviderSpecificValue	An object with the data type provided in .NET Framework's common language runtime is returned because no unique data types are supported.

Object	Method or property	Details
	GetProviderSpecificValues	An object with the data type provided in .NET Framework's common language runtime is returned because no unique data types are supported.
	NextResult method	false is returned because there is no multiple record set function.
HiRDBParameter	DbType property	If DbType.Boolean, DbType.Currency, DbType.Guid, or DbType.VarNumeric is specified, HiRDBException is returned because there is no corresponding type.
	Direction property	If HiRDBCommand class's ExecuteNonQuery, ExecuteReader, ExecuteScalar, or Prepare method is executed while Direction.ReturnValue is specified, HiRDBException is returned because there is no function for acquiring the stored procedure's return value.
	IsNullable property	Acquisition only; setting is not available. (The null value can always be specified.)
HiRDBProviderFactory	CreateDataSourceEnumerator method	NotSupportedException is returned because no HiRDB enumeration type is provided.
HiRDBTransaction	IsolationLevel property	The following value is always returned: • When the supported ADO.NET version is 1.1: IsolationLevel.ReadCommitted • When the supported ADO.NET version is 2.0: IsolationLevel.RepeatableRead

16.7 Data types of HiRDB.NET Data Provider

16.7.1 DbType and HiRDBType properties

When the DbType property of the HiRDBParameter class is set, the HiRDBType property of the same class is automatically set. When the HiRDBType property is set, the DbType property is automatically set. *Table 16-3* lists the HiRDBType property values that are automatically set when the DbType property is set, and *Table 16-4* lists the DbType property values that are automatically set when the HiRDBType property is set.

Table 16-3: HiRDBType property values that are automatically set when the DbType property is set

DbType property	HiRDBType property
AnsiString	VarChar
AnsiStringFixedLength	Char
Binary	Binary
Boolean	[NotSupportedException exception]
Byte	SmallInt
Currency	[NotSupportedException exception]
Date	Date
DateTime	TimeStamp
Decimal	Decimal
Double	Float
Guid	[NotSupportedException exception]
Int16	SmallInt
Int32	Integer
Int64	Decimal
Object	Binary
SByte	SmallInt
Single	SmallFlt
String	MvarChar

DbType property	HiRDBType property
StringFixedLength	Mchar
Time	Time
UInt16	Integer
UInt32	Decimal
UInt64	Decimal
VarNumeric	[NotSupportedException exception]

Table 16-4: DbType property values that are automatically set when the HiRDBType property is set

HiRDBType property	DbType property
Binary	Object
Blob	Object
Char	AnsiStringFixedLength
Date	Date
Decimal	Decimal
Float	Double
Integer	Int32
IntervalYearToDay	String
IntervalHourToSecond	String
MChar	StringFixedLength
MVarChar	String
NChar	StringFixedLength
NVarChar	String
SmallFlt	Single
SmallInt	Int16
Time	Time
TimeStamp	DateTime
VarChar	AnsiString

16.7.2 Data types and accessories used by a UAP

The table below lists the data types that are set in the Value property of the HiRDBParameter class, for example, during execution of the INSERT and Get*XXXX* methods of the HiRDBDataReader class that are used during execution of SELECT. Note that HiRDB's NULL is represented by DBNull.Value of the .NET Framework type.

Table 16-5: Data types and accessories for HiRDB-type UAPs

Classification	HiRDB data type	.NET Framework type used by UAPs, for example in INSERT	Accessory used by UAP for SELECT
Character	CHAR[ACTER]	String	GetString()
	VARCHAR/CHAR[ACTER]VARYING	String	GetString()
	NCHAR/NATIONAL CHAR[ACTER]	String	GetString()
	NVARCHAR/NCHAR VARYING	String	GetString()
	MCHAR	String	GetString()
	MVARCHAR	String	GetString()
Numeric value	[LARGE]DEC[IMAL]/NUMERIC	Decimal	GetDecimal()
	SMALLINT	Int16	GetInt16()
	INT[EGER]	Int32	GetInt32()
	SMALLFLT/REAL	Single	GetFloat()
	FLOAT/DOUBLE PRECISION	Double	GetDouble()
Date and time	DATE	DateTime	GetDateTime()
	TIME	DateTime	GetDateTime()
	TIMESTAMP	DateTime	GetDateTime()
Other	BINARY	Byte[]	GetBytes()
	BLOB	Byte[]	GetBytes()
	INTERVAL YEAR TO DAY	String	GetString()
	INTERVAL HOUR TO SECOND	TimeSpan	GetString()

16.7.3 Type conversion by HiRDB.NET Data Provider

When no .NET Framework type or accessory listed in *Table 16-5* is used, type conversion takes place automatically within the HiRDB data provider. No .NET Framework type or accessory is used when `Int32`-type data is inserted in a table that contains items with the `CHAR` attribute or the `GetInt32` method is used for acquisition.

Tables 16-6 and *16-7* list the type conversions for `INSERT`, and *Tables 16-8* and *16-9* list the type conversions for `SELECT`.

For the definition of symbols used in *Tables 16-6* through *16-9*, see *16.7.3(1) Definition of symbols*.

Table 16-6: List of type conversions for INSERT (1/2)

.NET Framework type	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`Boolean`	E1	E1	E1	E1	E1	E1	E1	E1	E1
`Int16`	N	N	N	N	N	N	N	E1	E1
`Int32`	N	C1	N	N	N	N	N	E1	E1
`Int64`	C2	C1	N	N	N	N	N	E1	E1
`UInt16`	N	C1	N	N	N	N	N	E1	E1
`UInt32`	C2	C1	N	N	N	N	N	E1	E1
`UInt64`	C2	C1	N	N	N	N	N	E1	E1
`Single` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Single` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Double` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Double` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Decimal` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Decimal` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Char`	N1	N1	E1	E1	E1	N	N	N	N
`Char[]`	E1	E1	E1	E1	E1	E1	E1	E1	E1

.NET Framework type	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`String`	C2	C1	N	N	N	N	N	N	N
`DateTime`	E1	E1	E1	E1	E1	N	N	E1	E1
`TimeSpan`	E1	E1	E1	E1	E1	N	N	E1	E1
`Guid`	E1	E1	E1	E1	E1	N	N	E1	E1
`Byte`	N	N	N	N	N	N	N	E1	E1
`Byte[]`	E1	E1	E1	E1	E1	E1	E1	E1	E1
`Sbyte`	N	N	N	N	N	N	N	E1	E1
`SByte[]`	E1	E1	E1	E1	E1	E1	E1	E1	E1

Table 16-7: List of type conversions for INSERT (2/2)

.NET Framework type	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
`Boolean`	E1	E1	E1	E1	E1	E1	E1	E1	E1
`Int16`	N	N	E1	E1	E1	E2	E2	E1	E1
`Int32`	N	N	E1	E1	E1	E2	E2	E1	E1
`Int64`	N	N	E1	E1	E1	E2	E2	E1	E1
`UInt16`	N	N	E1	E1	E1	E2	E2	E1	E1
`UInt32`	N	N	E1	E1	E1	E2	E2	E1	E1
`UInt64`	N	N	E1	E1	E1	E2	E2	E1	E1
`Single` data with fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Single` data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Double` data with fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Double` data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Decimal` data with fractional part	N	N	E1	E1	E1	E2	E2	E1	E1

.NET Framework type	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
`Decimal` data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Char`	N	N	E1	E1	E1	E2	E2	E1	E1
`Char[]`	E1	E1	E1	E1	E1	E2	E2	E1	E1
`String`	N	N	N	N	N	N	N	E1	E1
`DateTime`	N	N	N	N	N	E2	E2	E1	E1
`TimeSpan`	N	N	E1	E1	E1	E2	N	E1	E1
`Guid`	N	N	E1	E1	E1	E2	E2	E1	E1
`Byte`	N	N	E1	E1	E1	E2	E2	N	N
`Byte[]`	E1	E1	E1	E1	E1	E2	E2	N	N
`Sbyte`	N	N	E1	E1	E1	E2	E2	N	N
`SByte[]`	E1	E1	E1	E1	E1	E2	E2	N	N

Note 1: `INSERT` operation on `NCHAR`/`NVARCHAR`

If the size of data obtained after S-JIS conversion consists of an odd number of bytes, the `[Hitachi.HiRDB.HiRDBException]` `KFPZ24026-E` format conversion error occurs.

Note 2: During array `INSERT`

If the type is not an `Object` array type, the `[Hitachi.HiRDB.HiRDBException] KFPZ24026-E` format conversion error occurs. Because no array can be inserted in `BLOB`, the same error occurs if an attempt is made.

Table 16-8: List of type conversions for SELECT (1/2)

Accessory	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`GetBoolean`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetByte`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetBytes`	N	N	E1	N	N	N	N	N	N
`GetChar`	E3	E3	E3	E3	E3	E3	E3	E3	E3

Accessory	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`GetChars`	E1	E1	E1	E1	E1	N	N	N	N
`GetData`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetDateTime`	E1	E1	E1	E1	E1	C6	C6	C6	C6
`GetDecimal`	N	N	N	N	N	C7	C7	C7	C7
`GetDouble`	N	N	N	N	N	C8	C8	C8	C8
`GetFloat`	N	N	N	N	N	C9	C9	C9	C9
`GetGuid`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetInt16`	C1	N	C1	C1	C1	C1	C1	C1	C1
`GetInt32`	N	N	C2	C2	C2	C2	C2	C2	C2
`GetInt64`	N	N	C10	C10	C10	C10	C10	C10	C10
`GetString`	N	N	N	N	N	N	N	N	N
`GetValue`	N	N	N	N	N	N	N	N	N
`GetValues`	N	N	N	N	N	N	N	N	N

Table 16-9: List of type conversions for SELECT (2/2)

Accessory	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
`GetBoolean`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetByte`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetBytes`	N	N	E1	E1	E1	E1	E1	N	N
`GetChar`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetChars`	N	N	E1	E1	E1	E1	E1	E1	E1
`GetData`	E3	E3	E3	E3	E3	E3	E3	E3	E3
`GetDateTime`	C6	C6	N	N	N	E1	E1	E1	E1
`GetDecimal`	C7	C7	E1	E1	E1	E1	E1	E1	E1
`GetDouble`	C8	C8	E1	E1	E1	E1	E1	E1	E1

Accessory	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
GetFloat	C9	C9	E1	E1	E1	E1	E1	E1	E1
GetGuid	E3	E3	E3	E3	E3	E3	E3	E3	E3
GetInt16	C1	C1	E1	E1	E1	E1	E1	E1	E1
GetInt32	C2	C2	E1	E1	E1	E1	E1	E1	E1
GetInt64	C10	C10	E1	E1	E1	E1	E1	E1	E1
GetString	N	N	N	N	N	N	N	N	N
GetValue	N	N	N	N	N	N	N	N	N
GetValues	N	N	N	N	N	N	N	N	N

Note 1: During DATE acquisition

When the GetDateTime method is used, 00:00:00 is set in the time field. When the GetString method is used, the value is set in the format *YYYY/MM/DD*.

Note 2: During TIME/TIMESTAMP acquisition

When the GetDateTime method is used, the current date is set in the date field. When the GetString method is used, the value is set in the following format:

TIME: *hh*:*mm*:*ss*

TIMESTAMP(0): *YYYY/MM/DD hh*:*mm*:*ss*

TIMESTAMP(2): *YYYY/MM/DD hh*:*mm*:*ss*.*nn*

TIMESTAMP(4): *YYYY/MM/DD hh*:*mm*:*ss*.*nnnn*

TIMESTAMP(6): *YYYY/MM/DD hh*:*mm*:*ss*.*nnnnnn*

Note 3: During INTERVALYEARTODAY acquisition

When the GetString method is used, the value is set in the format ± *YYYY/MM/DD*.

Note 4: During INTERVALHOURTOSECOND acquisition

When the GetString method is used, the value is set in the format ± *hh*:*mm*:*ss*.

(1) Definition of symbols

(a) HiRDB data types

The following table defines the symbols used for the HiRDB data types:

Symbol	Definition
I	INTEGER
SI	SMALLINT
DE	DECIMAL and LARGE DECIMAL
F	FLOAT/DOUBLE PRECISION
SF	SMALLFLT and REAL
C	CHARACTER
VC	VARCHAR
NC	NCHAR and NATIONAL CHARACTER
NVC	NVARCHAR
MC	MCHAR
MVC	MVARCHAR
DA	DATE
T	TIME
TS	TIMESTAMP
IY	INTERVAL YEAR TO DAY
IHS	INTERVAL HOUR TO SECOND
BI	BINARY
BL	BLOB

(b) Whether or not type conversion is supported

N indicates normal; C indicates a conditional; and E indicates error. Some of these letters are followed by a number; they are defined as follows:

Symbol	Definition
N	Numeric character code is set.
C1	For Int32 type, Int64 type, Single data type with no fraction part, Double data type with no fraction part, Decimal data type with no fraction part, and String data type with no fraction part: -32768 to 32767: Normal For UInt16, UInt32, and UInt64 types: 0 to 32767: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error

Symbol	Definition
C2	For Int64 type, Single data type with no fraction part, Double data type with no fraction part, Decimal data type with no fraction part, and String data type with no fraction part: -2147483648 to 2147483647: Normal For UInt32 and UInt64 types: 0 to 2147483647: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C3	-32768 to 32767: Normal (rounded) Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C4	-2147483648 to 2147483647: Normal (rounded) Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C5	0 to 255: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C6	DateTime format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C7	Decimal format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C8	Double format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C9	Float format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C10	-9223372036854775808 to 9223372036854775807: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
E1	[Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
E2	[Hitachi.HiRDB.HiRDBException] KFPZ24107-E Decimal, date and time, time interval type overflow [Hitachi.HiRDB.HiRDBException] KFPZ24106-E Date and time, time interval type format error
E3	[System.NotSupportedException] unsupported error

16.8 Connection pooling function

(1) About the connection pooling function

The connection pooling function reduces the number of new connections by keeping reusable connections in effect.

When a UAP calls the `Open` method of the `Connection` object, HiRDB.NET Data Provider checks the connection pool for a reusable connection. If there is an available connection in the pool, HiRDB.NET Data Provider returns that connection to the calling program without opening a new connection. If there is no available connection in the pool, HiRDB.NET Data Provider opens a new connection. When a UAP calls the `Close` method of the `Connection` object, HiRDB.NET Data Provider does not actually close the connection. Instead, the next time the `Open` method is called, HiRDB.NET Data Provider reuses the connection kept in the connection pool without opening a new connection.

A connection is kept in the connection pool for a specified period of time. If the connection is not reused within that period, it is discarded from the pool.

Hint:

If you make frequent use of the `Open` and `Close` methods of the `Connection` object, you might be able to significantly improve UAP performance and scalability by using connection pooling.

(2) How to use the function

You specify the following settings to use the connection pooling function:

- Set `true` (default value) in the `HiRDBConnection.Pooling` property.
- Set in the `HiRDBConnection.LifeTime` property the amount of time a connection is to be kept in the connection pool.

When these settings have been specified, a connection in the pool that satisfies both the following conditions is reused:

- The connection is not being used currently.
- The connection character string matches perfectly.

16.9 Provider-independent codes using DbProviderFactory

ADO.NET2.0 can now create instances of other classes, such as `Command` and `Parameter`, by using the `DbProviderFactory` instance. You can create an instance specific to a provider based on the provided information by creating a `DbProviderFactory` instance with a character string indicating the specified provider's namespace (referred to hereafter as the *provider name*). You can register a provider name in a configuration file for later acquisition. This enables you to create provider-independent codes and select a desired provider during execution.

(1) Adding provider information

You create a `DbProviderFactory` instance by calling the `GetFactory` method of the `DbProviderFactories` class. When you install .NET Framework version 2.0 or 3.0, you can specify the following four types of provider names with this method:

- System.Data.Odbc
- System.Data.OleDb
- System.Data.OracleClient
- System.Data.SqlClient

You can specify provider names in the `GetFactory` method if you add provider information to the `DbProviderFactories` element of the `system.data` section in the `machine.config` file that is installed when you install .NET Framework version 2.0 or 3.0.

For HiRDB.NET Data Provider, the `invariant` value to be specified is `Hitachi.HiRDB`.

The following shows an example that adds information about HiRDB.NET Data Provider.

■ Adding provider information

```
<system.data>
  <DbProviderFactories>
          :
          :
          :
    <add name="HiRDB Data Provider" invariant="Hitachi.HiRDB"
      description=".NET Framework Data Provider for HiRDB"
      type="Hitachi.HiRDB.HiRDBProviderFactory, pddndp20,
      Version=X.X.X.X, Culture=neutral,
      PublicKeyToken=YYYYYYYYYYYYYYYY" />
  </DbProviderFactories>
</system.data>
```

Legend:

X.X.X.X: Assembly version. You can determine the assembly version by checking the properties of `pddndp20.dll`.

YYYYYYYYYYYYYYYY: Assembly public key token. You can determine the assembly public key token by entering the following command at the command prompt or MS-DOS prompt:

```
sn -T pddndp20.dll
```

(2) Specifying provider names by using a configuration file

You can register provider names in a configuration file for later acquisition when you create a `DbProviderFactory` instance. For details about the configuration file, see the documentation of .NET Framework.

For HiRDB.NET Data Provider, the provider name (`value`) to be specified is `Hitachi.HiRDB`. Note that a configuration file is not required if you specify a provider name directly in the argument of the `GetFactory` method of the `DbProviderFactories` class.

The following shows an example of a configuration file. The key name (`provider`) can be any character string.

- Example of configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="provider" value="Hitachi.HiRDB" />  ...HiRDB.NET Provider
  </appSettings>
</configuration>
```

(3) Creating a DbProviderFactory instance

(a) When a configuration file is not used

This example specifies the provider name (`Hitachi.HiRDB`) directly in the argument of the `GetFactory` of the `DbProviderFactories` class and then creates a `DbProviderFactory` instance. The following shows sample coding:

```
DbProviderFactory dataFactory =
           DbProviderFactories.GetFactory("Hitachi.HiRDB");
```

(b) When a configuration file is used

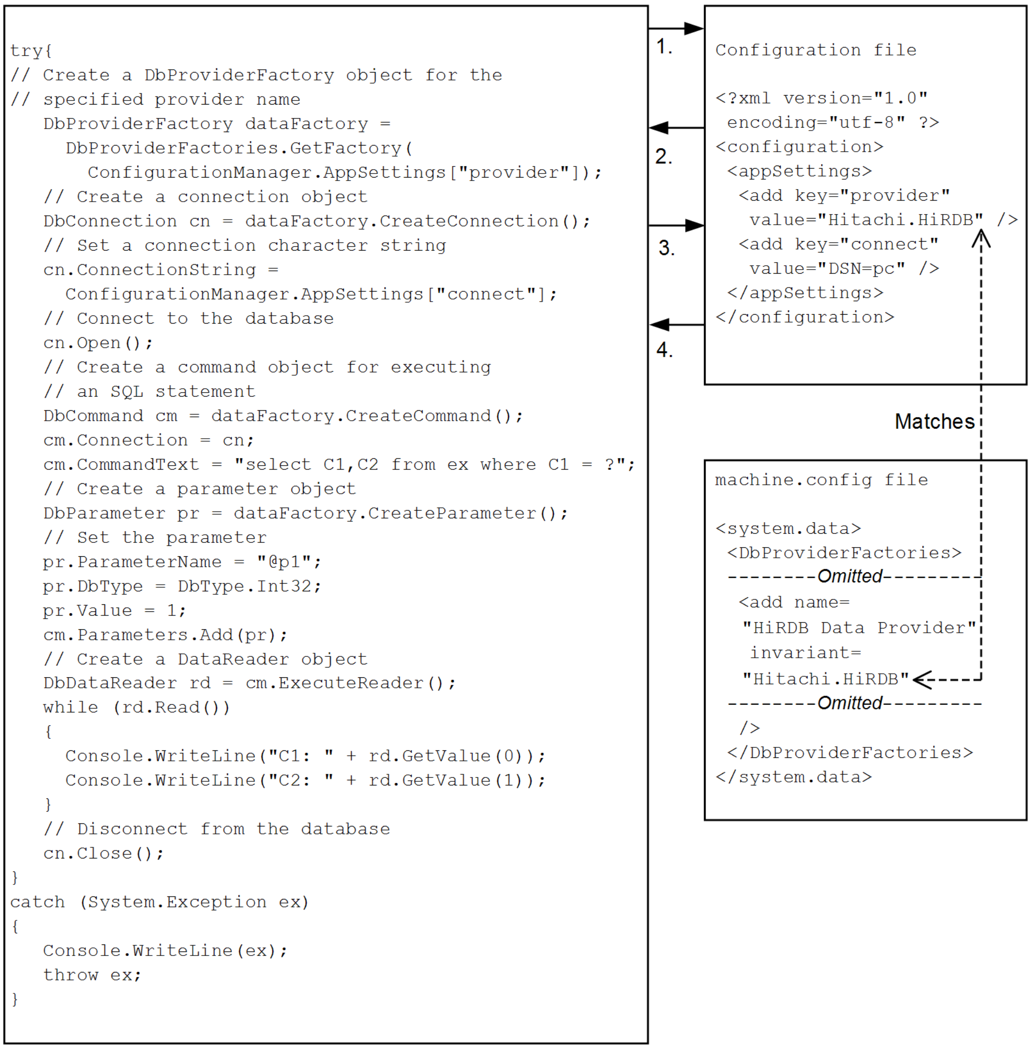
This example acquires the value of the key name (`provider`) from the configuration file and then creates a `DbProviderFactory` instance by specifying that value in the argument of the `GetFactory` method of the `DbProviderFactories` class. The following shows sample coding:

```
DbProviderFactory dataFactory =

DbProviderFactories.GetFactory(ConfigurationManager.AppSettings["provider"]);
```

(4) Sample coding

By acquiring the provider name and connection character string that depend on the provider from the configuration file, you can use different providers without having to change the program.

The following shows sample coding for using a configuration file:



```
try{
// Create a DbProviderFactory object for the
// specified provider name
   DbProviderFactory dataFactory =
     DbProviderFactories.GetFactory(
       ConfigurationManager.AppSettings["provider"]);
   // Create a connection object
   DbConnection cn = dataFactory.CreateConnection();
   // Set a connection character string
   cn.ConnectionString =
     ConfigurationManager.AppSettings["connect"];
   // Connect to the database
   cn.Open();
   // Create a command object for executing
   // an SQL statement
   DbCommand cm = dataFactory.CreateCommand();
   cm.Connection = cn;
   cm.CommandText = "select C1,C2 from ex where C1 = ?";
   // Create a parameter object
   DbParameter pr = dataFactory.CreateParameter();
   // Set the parameter
   pr.ParameterName = "@p1";
   pr.DbType = DbType.Int32;
   pr.Value = 1;
   cm.Parameters.Add(pr);
   // Create a DataReader object
   DbDataReader rd = cm.ExecuteReader();
   while (rd.Read())
   {
     Console.WriteLine("C1: " + rd.GetValue(0));
     Console.WriteLine("C2: " + rd.GetValue(1));
   }
   // Disconnect from the database
   cn.Close();
}
catch (System.Exception ex)
{
   Console.WriteLine(ex);
   throw ex;
}
```

```
Configuration file

<?xml version="1.0"
 encoding="utf-8" ?>
<configuration>
 <appSettings>
  <add key="provider"
   value="Hitachi.HiRDB" />
  <add key="connect"
   value="DSN=pc" />
 </appSettings>
</configuration>
```

```
machine.config file

<system.data>
 <DbProviderFactories>
 --------Omitted---------
  <add name=
   "HiRDB Data Provider"
   invariant=
   "Hitachi.HiRDB"
 --------Omitted---------
  />
 </DbProviderFactories>
</system.data>
```

Explanation:

1. Searches the configuration file by key name (provider).
2. Returns the value (Hitachi.HiRDB).
3. Searches the configuration file by key name (connect).
4. Returns the value (DSN=pc).

16.10 Troubleshooting function of HiRDB.NET Data Provider

ADO.NET 2.0-compatible HiRDB.NET Data Provider can collect method traces as troubleshooting information.

(1) How to collect method traces

You can collect method trace information by specifying appropriate values in the PDCLTPATH and PDDNDPTRACE client environment definitions. For details about the client environment definitions, see *6.6 Client environment definitions (setting environment variables)*.

(2) Method trace output rules

The following rules apply to method trace output:

- Two method trace files for collecting information are created in a specified directory.
- A method trace is output at the following times:
  - When a method is called
  - When a method returns control
  - When a property is specified
  - When a property is acquired
- UTF-8 character encoding is used.
- The names of the created files are pddndp*xxxxx*_*yyyy*_1.trc and pddndp*xxxxx*_*yyyy*_2.trc, where *xxxxx* is the process ID and *yyyy* is the connection number.

(3) How to interpret the method trace information

This subsection shows and explains an example of method trace output.

Header

```
[1][1742][sds01][12345678][HiRDB_Data_Provider20][08.04.0.0]
 1  2     3      4         5                      6
```

Explanation:

The header is output at the beginning of the file.

1. Connection number
2. Connection-target server's process ID

3. Name of single server or front-end server

4. UAP's process ID

5. Trace ID information

6. HiRDB.NET Data Provider's assembly version

Method trace information

```
[0000000001][E][HiRDBCommand@12345678 ExecuteNonQuery][SID(2)][2008/08/27
1:29:10.123]
 1           2  3               4          5                   6         7

  [Return=0]
   8
  [nArraySize=10]                            -|
                                              |
        :                                     |-9
                                              |
  [CommandText=INSERT INTO T1 VALUES(100)]  -|

  [MessageText=KFPA11117-E Number of insert values not equal to number of insert
columns] 10
  [SQLCODE=-117] 11
  [SQLWARN=0000] 12
  location Hitachi.HiRDB.native.HiRDBcore.ClearSectionItems()      -|
  location Hitachi.HiRDB.HiRDBConnection.Close()                    |
  location Hitachi.HiRDB.HiRDBConnection.Dispose(Boolean disposing) |-13
  location Hitachi.HiRDB.HiRDBConnection.Finalize()                -|
```

Explanation:

Method trace information is output each time a method is called, a method returns control, a property is set, or a property is acquired.

1. Thread ID

2. Access type:

   `E`: Method call

   `R`: Return from method

   `S`: Value setting in property

   `G`: Acquisition of property value

   The output information depends on the access type, as shown in the following table:

Access type		Call type	Argument or property value	Return value	Error information
Method	E	Call	Y	--	--
	R	Return (normal)	--	Y	--
		Return (error)	--	--	Y
Property	S	Setting (normal)	Y	--	--
		Setting (error)	Y	--	Y
	G	Acquisition (normal)	--	Y	--
		Acquisition (error)	--	--	Y

Legend:

Y: Output

--: Not output

3. Class name. In this example, the provider name is omitted.
4. Hash code. In this example, an at mark (`@`) separates the hash code from the class name.
5. Method name or property name
6. Section number. For a method or property that has no effect on SQL execution, an asterisk (`*`) is output because the section number cannot be identified.
7. Trace collection date and time
8. Return value. In the event of an exception, the `Exception` class name is output.
9. Argument names and values or property names and values. A name and a value are separated by an equal sign (`=`).
10. Error message
11. `SQLCODE` that resulted from the SQL statement execution.
12. `SQLWARN`. This is warning information (displayed in hexadecimal). For details, see *11.1.1(2) Examining SQL trace information*.
13. Stack trace

For some methods, property information is output. The following table shows the output format for each access type:

Access type		Caller	Format	Remarks
Method	E	HiRDBCommand.Execute HiRDBCommand.ExecuteDbDataReader HiRDBCommand.ExecuteNonQuery HiRDBCommand.ExecuteReader HiRDBCommand.ExecuteScalar	`CommandText=VALUE`[#1] `Parameters.Count=VALUE` `PARAMETER_VALUE`[#2]	If there are arguments, the argument information is output.
		Other	`ARGUMENT`[#3]`=VALUE`	--

	R	HiRDBConnection.Open	ConnectionString=VALUE
			ServerVersion=VALUE
				--

		Other	Return=VALUE	--
Property	S	--	`PROPERTY`[#4]`=VALUE`	--
	G	--	Return=VALUE	--

Legend:

--: Not applicable

#1

`VALUE` indicates the property value, argument value, or return value that was set or acquired.

#2

`PARAMETER_VALUE` indicates information about each parameter that has been registered in `HiRDBParameterCollection`. The information consists of the following:

ParameterName,HiRDBType,Value,Precision,Scale,Repetition

#3

`ARGUMENT` indicates an argument name.

#4

`PROPERTY` indicates a property name.

(4) Making a backup of a method trace file

When the capacity of the current method trace file reaches a specified size, the system starts using the other method trace file for output of method trace information. When this occurs, the oldest method trace information in the switched-in method trace file is overwritten by new method trace information. For this reason, you should back up the contents of the method trace file upon termination of a UAP.

To determine which method trace file is being used currently, check the two files' last update dates and times. The file with the most recent update date and time is the current method trace file. Use the `dir` command or Explorer to check the update dates and times.

16.11 Example of a UAP using HiRDB.NET Data Provider

This section describes an example of a UAP using HiRDB.NET Data Provider.

16.11.1 Connecting to the database

The following example connects to HiRDB and then disconnects from HiRDB:

- Example of Visual C# .NET code

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {

            try
            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");  ...1

                // Connect to the database
                cn.Open();  ..........................................2

                // Disconnect from the database
                cn.Close();  .........................................3
            }

            catch (HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            }  ......................................................4
        }
    }
}
```

- Example of Visual Basic.NET code

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1

    Sub Main()

        Dim cn As HiRDBConnection
        Dim cm As HiRDBCommand


        Try
            ' Create a Connection object
            cn = New HiRDBConnection("dsn=pc;")  ........................1

            ' Connect to the database
            cn.Open()  ..............................................2

            ' Disconnect from the database
            cn.Close()  .............................................3


        Catch ex As HiRDBException
            Console.WriteLine(ex)

        Catch ex As System.Exception

            Console.WriteLine(ex)

        End Try  ....................................................4

    End Sub

End Module
```

Explanation

1. First, create a `HiRDBConnection` object. This object manages all communications with HiRDB. Because `Disconnect` is called from within the `HiRDBConnection: Dispose` method, when this object disappears, the database is automatically disconnected.

   For this method, you must specify one `string`-type argument. The character string to be specified is called a *connection character string*. This is the same type of connection character string as those used for `Connection` in ADO and ADO.NET. For details about the character strings that can be specified, see *16.5.3(2)(a) ConnectionString*.

2. To connect to the database, use the `Open` method.

3. To disconnect from the database, use the `Close` method. Using the `Close` method while a connection is not established does not result in an exception.

4. An exception occurs if the server is not running, communication is disabled, the SQL statement is invalid, or in similar cases. Basically, a block using HiRDB.NET Data Provider detects exceptions by `try` through `catch`, and then displays an exception message.

   In the case of an overall HiRDB error, `System.Exception` occurs, and in the case of a HiRDB.NET Data Provider-specific error, `HiRDBException` occurs. Make sure that `System.Exception` is not abbreviated as `Exception`.

   A HiRDB Client Library or HiRDB.NET Data Provider-specific error code is stored in the `ErrorCode` property of the exception object that is created by HiRDB.NET Data Provider.

   A 3-digit (-*XXX*) or 4-digit (-*XXXX*) error code indicates `KFPA1`*XXXX* and a 5-digit error code (`-24`*XXX*) indicates `KFPZ24`*XXX*.

16.11.2 Executing the SQL statement

This example creates a table named `ex`:

- Example of Visual C# .NET code

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            try

            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

                // Connect to the database
                cn.Open();

                // Create a Command object
                HiRDBCommand cm = new HiRDBCommand();
                // Create a table
                cm.Connection = cn;
                cm.CommandText = "create table ex (a int)";
                cm.ExecuteNonQuery();  .................................1

                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)

            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

- Example of Visual Basic.NET code

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1

    Sub Main()

        Dim cn As HiRDBConnection
        Dim cm As HiRDBCommand


        Try
            ' Create a Connection object
            cn = New HiRDBConnection("dsn=pc;")

            ' Connect to the database
            cn.Open()

            ' Create in the Command object
            cm = New HiRDBCommand()


            ' Create a table
            cm.Connection = cn
            cm.CommandText = "create table ex (a int)"
            cm.ExecuteNonQuery()  ....................................1

            ' Disconnect from the database
            cn.Close()
        Catch ex As HiRDBException
            Console.WriteLine(ex)

        Catch ex As System.Exception

            Console.WriteLine(ex)

        End Try

    End Sub

End Module
```

Explanation

1. To execute an SQL statement, use the `Execute` method. Specify a `string`-type SQL statement as is in the `CommandText` property of `HiRDBCommand`. This method can execute most SQL statements. Special SQL statements such as `commit` cannot be executed by this method, as well as statements such as `select` that must receive a result set. To execute these SQL statements, use dedicated methods.

16.11.3 Executing a transaction

This example inserts data `1` to the `ex` table:

- Example of Visual C# .NET code

```

using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Create a Connection object
            HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

            // Connect to the database
            cn.Open();

            // Create a Transaction object
            HiRDBTransaction tran;
            // Start of transaction
            tran = cn.BeginTransaction(IsolationLevel.ReadCommitted);  ..1
            // Create a Command object
            HiRDBCommand cm = new HiRDBCommand();
            cm.Connection = cn;
            cm.Transaction = tran;
            try

            {
                // Insert data in the table
                cm.CommandText = "insert into ex values (1)";
                cm.ExecuteNonQuery();

                // Transaction was successful
                tran.Commit();  .......................................2

                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)
            {
                // Transaction failed
                tran.Rollback();  .....................................3


                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                // Transaction failed
                tran.Rollback();  .....................................3

                Console.WriteLine(ex);
            }
        }
    }
}
```

- Example of Visual Basic.NET code

```
Imports System
Imports System.Data
Imports Hitachi.HiRDB

Module Module1

    Sub Main()

        Dim cn As HiRDBConnection
        Dim tran As HiRDBTransaction
        Dim cm As HiRDBCommand


        ' Create a Connection object
        cn = New HiRDBConnection("dsn=pc;")

        ' Connect to the database
        cn.Open()

        ' Start a transaction
        tran = cn.BeginTransaction(IsolationLevel.ReadCommitted)  .......1
        ' Create a Command object
        cm = New HiRDBCommand()
        cm.Connection = cn
        cm.Transaction = tran

        Try


            ' Insert data in the table
            cm.CommandText = "insert into ex values (1)"
            cm.ExecuteNonQuery()

            ' Transaction is successful
            tran.Commit()  ..............................................2

            ' Disconnect from the database
            cn.Close()

        Catch ex As HiRDBException


            ' Transaction fails
            tran.Rollback()  ............................................3
            Console.WriteLine(ex)

        Catch ex As System.Exception

            ' Transaction fails
            tran.Rollback()  ............................................3
            Console.WriteLine(ex)

        End Try

    End Sub

End Module
```

Explanation

1. To start a transaction, use the `BeginTransaction` method.
2. To complete the transaction, call the `Commit` method.
3. To restore, call the `Rollback` method.

16.11.4 Executing a search statement

This example displays all table data:

Although the sample program is coded in Visual C# .NET, the program in Visual Basic.NET would be almost the same (if necessary, change information as appropriate).

```
using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)

        {
            try
            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

                // Connect to the database
                cn.Open();

                // Create a Command object
                HiRDBCommand cm = new HiRDBCommand();
                cm.Connection = cn;
                cm.CommandText = "select a from ex";


                // Create a DataReader object
                HiRDBDataReader rd = cm.ExecuteReader();  ...............1
                int i;
                while(rd.Read())
                {
                    for (i = 0 ; i < rd.FieldCount ; i++)
                    {
                     Console.WriteLine(rd.GetName(i) + " - " +rd.GetValue(i));
                    }
                }  ....................................................2
                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

Explanation

1. To execute a search, use the `ExecuteReader` method to create a `HiRDBDataReader`.
2. Use the `Read` method to move on to the next row. Use the `GetName` method to acquire a column name, and use the `GetValue` method to acquire a column value.

16.11.5 Executing the INSERT facility using arrays

This example inserts `123`, `200`, and `null` in the `ex` table:

Although the sample program is coded in Visual C# .NET, the program in Visual Basic.NET would be almost the same (if necessary, change information as appropriate).

```
// Create objects such as a connection object
HiRDBConnection    pConn = new HiRDBConnection("connection-character-string");
HiRDBCommand    pCom = pConn.CreateCommand();

// Connect to the database
pConn.Open();

// Create a parameter object
HiRDBParameter    pPar = pCom.CreateParameter();

// Set parameters
pPar.Direction = ParameterDirection.Input;
pPar.HiRDBType = HiRDBType.Integer;
object [] aValue = new object[3];
aValue[0] = 123;
aValue[1] = 200;
aValue[2] = null;
pPar.Value = aValue;
pCom.Parameters.Add(pPar);  .........................................1
```

```
// Use parameters to execute SQL statement
pCom.CommandText = "insert into ex values(?)";
pCom.ExecuteNonQuery(aValue.Length);  ..............................2

// Disconnect from the database
pConn.Close();
```

Explanation

1. Set the parameter value in the `value` parameter. Because `value` is the `object` type, it can reference all types. The `Int32` type is specified in normal `INSERT` statements, but in the `INSERT` statement using an array, the array of `object` is set in `value`, and each element of the `object` array is set to point to the `Int32` type. The same applies when other types are used; always set an array of `object` in `value`.
2. To execute the SQL statement, use `:overload` of `ExecuteNonQuery`. The normal `ExecuteNonQuery` has no argument, but when the `INSERT` statement using an array is used, specify the size of the array.

Note

The codes for setting `value` for parameters and for executing SQL statements vary depending on whether or not an array is used.

16.11.6 Executing a repetition column

This example inserts `123` and `456` in the first column of the `ex` table:

Although the sample program is coded in Visual C# .NET, the program in Visual Basic.NET would be almost the same (if necessary, change information as appropriate).

```
// Create objects such as a connection object
HiRDBConnection    pConn = new HiRDBConnection("connection-character-string");
HiRDBCommand    pCom = pConn.CreateCommand();

// Connect to the database
pConn.Open();

// Create a table
pCom.Connection = pConn;
pCom.CommandText = "create table ex(a int array[3])";
pCom.ExecuteNonQuery();
```

```
// Create a parameter object
HiRDBParameter    pPar = pCom.CreateParameter();

// Set parameters
pPar.Direction = ParameterDirection.Input;
pPar.HiRDBType = HiRDBType.Integer;
object [] aValue = new object[2];
aValue[0] = 123;
aValue[1] = 456;
pPar.Value = aValue;
// Set the maximum number of elements for column a of table ex
pPar.Repetition =3;
pCom.Parameters.Add(pPar);  .........................................1


// Use parameters to execute SQL statement
pCom.CommandText = "insert into ex values(?)";
pCom.ExecuteNonQuery();

// Execute the select statement
pCom.CommandText = "select * from ex";
HiRDBDataReader pReader = pCom.ExecuteReader();


// Fetch until there is no more data
while (pReader.Read())
{
    for (int i = 0; i < pReader.FieldCount; ++ i)
        for (int j = 0; j < pReader.GetFieldArrayCount(i); ++ j)
            Console.WriteLine(pReader.GetValue(i, j));
}  ................................................................2

// Disconnect from the database
pConn.Close();
```

Explanation

1. The `object` array is set in `value` for the same reason as for the `INSERT` statement using an array. For a repetition column, also set the `Repetition` extended property. This property specifies the number of repetition columns. There is no argument during the execution of the SQL statement.
2. For `FETCH`, an extended method for repetition columns is also provided with `DataReader`. First, use `GetFieldArrayCount` to acquire the number of repetition columns for the data obtained by `FETCH`. To acquire the value of the data obtained by `FETCH`, use `:overload` of `GetValue`. In the second argument, specify the number of the repetition column. An indexer `[int,int]` equivalent to this method is also provided.

Note

The usage of repetition columns is similar to that of the `INSERT` facility using arrays. The differences occur in the part that specifies the repetition count in the parameter and the part that executes the SQL statement.

16.11.7 Checking for an error in SQL statements and acquiring error information

This example checks for an error in the SQL statements and acquires error information.

Although the sample program is coded in Visual C# .NET, the program in Visual Basic.NET would be almost the same (if necessary, change information as appropriate).

```
using System;
using System.IO;
using System.Data;
using System.Windows.Forms;
using Hitachi.HiRDB;                // .NET Framework Data Provider for HiRDB

namespace SAMPLE
{
    public class SAMPLE
    {
        static void Main()
        {

            HiRDBConnection connection1 = new HiRDBConnection
            ("datasource=C:\\Windows\\HiRDB.ini;UID=USER1;PWD=USER1;");
            HiRDBCommand cm = new HiRDBCommand();


                try
                {
                    // Connect  ........................................1
                    connection1.Open();
                    cm.Connection = connection1;

                    // ***********************************************
               // Example of retrieval by SAMPLE1 (C1 INT, C2 INT, C3 VARCHAR(30))  ...2
                    // ***********************************************
                    // Create a parameter object
                    HiRDBParameter    par = cm.CreateParameter();
                    // Set parameter attributes
                    // Input parameter
                    par.Direction = ParameterDirection.Input;
                    // INTEGER type
                    par.HiRDBType = HiRDBType.Integer;
                    int aValue;
                    aValue = 200;
                    // Set parameter value
                    par.Value = aValue;
                    // Set SQL statement
                    cm.CommandText = "SELECT C2,C3 FROM SAMPLE1 WHERE C1=?
                                      WITH EXCLUSIVE LOCK NO WAIT";
                    // Assign parameter object
                    cm.Parameters.Add(par);
                    // Acquire DataReader object
                    HiRDBDataReader dr = cm.ExecuteReader();
                    int cnt=1;
                    Console.WriteLine("**** Searching ****");
                    while(dr.Read())
                    {
                        Console.WriteLine("**** "+cnt+"Searching row ***");
                        // Display data in C2
                        Console.WriteLine("C2="+dr.GetInt32(0));
                        // Display data in C3
                        Console.WriteLine("C3="+dr.GetString(1));
                        cnt ++;
                    }
                    // Free DataReader
                    dr.Close();
                    // Disconnect  ......................................3
                    connection1.Close();
                    connection1.Dispose();


                }
                catch (HiRDBException ex)  ..............................4
                {
                    // Output error information
                    Console.WriteLine(ex);
                    // Acquire individual information by the following process
                    // Output SQLCODE
                    Console.WriteLine("SQLCODE="+ex.ErrorCode);
                    // Output SQLERRM (SQL message)
                    Console.WriteLine("SQLERRM=" + ex.Message);
                }
        }
    }
}
```

Explanation:

1. Uses the `Open` method to connect to HiRDB.
2. Executes the SQL statement that displays a row that satisfies a specified condition.
3. Uses the `Close` method to disconnect from HiRDB.
4. Returns `SQLException` and outputs error information in the event of an error.

Chapter
17. Type2 JDBC Driver

This chapter explains the JDBC driver installation, environment setup, and JDBC functions. Note that the JDBC driver cannot be used in the Linux for AP8000 edition client.

Hereafter in this chapter, the Type2 JDBC driver is referred to as *JDBC driver*.

The Type2 JDBC driver has become obsolete. It is still supported in order to maintain compatibility with applications, but this support will be discontinued in the future. Please use the Type4 JDBC driver.

17.1 Installation and environment setup
17.2 JDBC1.0 facility
17.3 JDBC2.0 basic facility
17.4 JDBC2.0 Optional Package
17.5 JAR file access facility
17.6 Array class
17.7 Specifying a value when using a repetition column as the ? parameter
17.8 Functions provided by the HiRDB JDBC driver
17.9 Notes on using the BLOB type
17.10 Setting system properties
17.11 Connection information setup/acquisition interface
17.12 Data types and character codes
17.13 Classes and methods with limitations

17.1 Installation and environment setup

17.1.1 Installing

You can select the installation of a JDBC driver when installing HiRDB.

The following table shows the JDBC driver's installation directories and files.

Table 17-1: JDBC driver's installation directory and file

Platform	Type	Installation directory	File
UNIX	HiRDB server	`$PDDIR/client/lib/`	`pdjdbc.jar`#1, #3 `libjjdbc.sl(libjjdbc.so)`#2, #3
	HiRDB client	_*/HiRDB*_`/client/lib/`	`pdjdbc.jar`#1, #3 `libjjdbc.sl(libjjdbc.so)`#2, #3
Windows	HiRDB server	`%PDDIR%\CLIENT\UTL\`	`pdjdbc.jar`#3 `jjdbc.dll`#3
	HiRDB client	_*\HiRDB*_`\CLIENT\UTL\`	`pdjdbc.jar`#3 `jjdbc.dll`#3

Note

The underline indicates the HiRDB client's installation directory.

#1: For the 32-bit mode HP-UX (IPF) edition, the file is `pdjdbc32.jar`.

#2: For the 32-bit mode HP-UX (IPF) edition, the file is `libjjdbc32.so`.

#3: For the Windows (x64) and Linux (EM64T) editions, run in 32-bit mode.

To use the JDBC driver in an HP-UX (IPF), Linux (IPF), or Windows Server 2003 (IPF) environment, you need J2SDK v1.4.2. Note that J2SDK v1.4.2 must be run on an IPF-compliant Java Virtual Machine.

17.1.2 Environment setup

The following shows the environment variable definition required for JDBC driver operation.

(1) UNIX environment

Specify the following information in the environment variable for the execution environment:

`CLASSPATH=$CLASSPATH:`[*installation-directory*]`/pdjdbc.jar`#

#: For the 32-bit mode HP-UX (IPF) edition, the file is `pdjdbc32.jar`. Do not set `pdjdbc.jar` and `pdjdbc32.jar` at the same time.

(2) Windows environment

From **Control Panel**, choose **System**, then in the System Properties dialog box, choose **Advanced**, and specify the following information as the environment variable:
`CLASSPATH=%CLASSPATH%;[`*installation-directory*`]\pdjdbc.jar`

17.1.3 Abbreviation of methods

- Methods that begins with `get` are referred to collectively as `get`*XXX* methods.
- Methods that begins with `set` are referred to collectively as `set`*XXX* methods.

17.2 JDBC1.0 facility

17.2.1 Driver class

(1) Overview

The `Driver` class provides the following functions:

- Database connection
- Validity checking on a specified URL
- Acquisition of the connection properties specified with the `DriverManager.getConnection` method
- Acquisition of driver version information

For details about and usage of each method provided with the `Driver` class, see the applicable JDBC manual. This section explains the database connection procedure and the URL syntax unique to this JDBC driver.

(2) Database connection using the DriverManager

To execute DB connection using the `DriverManager` class provided by the Java execution environment:

1. Register the `Driver` class in the Java Virtual Machine.
2. Call the `DriverManager.getConnection` method using the connection information as the argument.

(a) Registering in Java Virtual Machine with the Driver class

Register the `Driver` class in the Java Virtual Machine by using the `Class.forName` method or by registering in the system properties. The package name and `Driver` class name of the JDBC driver specified for registration are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`Driver` class name: `PrdbDriver`

- **Using the Class.forName method**

  Call the `Class.forName` method from within the application as follows:

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

- **Registering in the system properties**

  Call the `System.setProperty` method from within the application as follows:

```
System.setProperty("jdbc.drivers","JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

(b) Defining the connection information and establishing a database connection

To connect to the database, use one of the following methods:

- **Using the DriverManager.getConnection method**

```
Connection con = DriverManager.getConnection(String url, String user, String password)
;
or
Connection con = DriverManager.getConnection(String url, Properties info) ;
```

- **Specification with an internal driver**

  When an internal driver is used, the information called by the routine at the HiRDB side is assumed as the connection information (such as the authorization identifier). However, when a trace is acquired within the JDBC driver, `INNER` is assumed as the authorization identifier.

```
Specification for internal driver only:
Connection con = DriverManager.getConnection(String url) ;
```

- **Directly calling the connect method in the Driver class**

```
Driver drv = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver();
Connection con = drv.connect(String url, Properties info) ;
```

In the arguments of the previous methods, specify the information required for database connection.

If a database connection is successful, the JDBC driver returns a `Connection` object as a result of the method call. If required information is not specified in each argument, or invalid information is specified, the JDBC driver throws an `SQLException` as a result of the method call.

Table 17-2 lists the arguments of the `getConnection` method, and *Table 17-3* lists the information to be specified for `Properties info`.

Table 17-2: Arguments of the getConnection method

Argument	Description	Specification
`String url`	URL; For URL, see *(3) URL syntax*.	R
`String user`	Authorization identifier[#1]	R[#2]
`String password`	Password	O
`Properties info`	See *Table 17-3*.	--

Legend:

R: Required.

O: Optional.

--: Not applicable.

#1: If null or space characters are specified for the authorization identifier, this method throws an `SQLException`. The method also throws an `SQLException` if the driver-converted character codes and, as a result, the size of the character string specified for the authorization identifier exceed 30 bytes. For details about character code conversion, see *17.12.2 Character code conversion facility*.

#2: The argument can be omitted, if specified with the internal driver.

Table 17-3: Information to be specified for Properties info

Key	Description	Specification
`user`	Authorization identifier[#1]	R[#2]
`password`	Password	O
`ENCODELANG`	In a Java program, Unicode is used for the character codes. Therefore, during character data processing with HiRDB, the JDBC driver performs mutual character code conversion between HiRDB's character data and Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the Java Virtual Machine. You must specify the character set names specified by the JDBC driver for the provided encoder and decoder. The settings can be for any character set (such as MS932) supported by Java. For details about this operation if you specify OFF or have not specified anything in `Properties info` (including the settings using the `DataSource.setEncodeLang` method and `ENCODELANG` of the URL), see *17.11.5 setEncodeLang*.	O
`COMMIT_BEHAVIOR`	When HiRDB commits, this key specifies whether or not the following classes are to remain valid after commit has executed: • `ResultSet` class • `Statement` class, `PreparedStatement` class, and `CallableStatement` class For details about the specification values, see *17.11.19 setCommit_Behavior*. Note: See *Notes on COMMIT_BEHAVIOR* following this table.	O

Key	Description	Specification
BLOCK_UPDATE	Specifies whether or not multiple parameters are to be processed at one time when the ? parameter is used to update databases. When this information is omitted, FALSE is assumed. TRUE: Processes multiple parameters at one time. FALSE: Processes parameter sets individually. Other: Assumes that FALSE is specified. Notes: • When TRUE is set, the batch update function supports HiRDB facilities using arrays. • Only INSERT, UPDATE, and DELETE SQL statements can use facilities using arrays. All other SQL statements are processed sequentially, not in batch mode. • Even the SQL statements that can use facilities using arrays are processed sequentially, not in batch mode, if they do not satisfy the conditions for facilities using arrays. • To use facilities using arrays, see *17.3.2 Batch updating*. • For details about the facilities using arrays, see *4.8 Facilities using arrays*. • This function can also be specified using the HiRDB_for_Java_BLOCK_UPDATE system property. However, when BLOCK_UPDATE is set, the HiRDB_for_Java_BLOCK_UPDATE system property setting is ignored.	O
LONGVARBINARY_ACCESS	Specifies the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY). When this key is omitted, REAL is assumed. REAL: Accesses real data from HiRDB. LOCATOR: Uses the HiRDB locator. Other: Assumes that REAL is specified.	O

Key	Description	Specification
HiRDB_for_Java_SQL_IN_NUM	Specifies the maximum number of input or input/output ? parameters in the SQL statements to be executed. This is the number of input or input/output ? parameter information items that is acquired during SQL preprocessing. If the actual number of input or input/output ? parameters is greater than this property value, the input or input/output ? parameter information is acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or non-numeric value results in an error. Notes: • This item can also be specified using the HiRDB_for_Java_SQL_IN_NUM system property. However, when HiRDB_for_Java_SQL_IN_NUM is specified for Properties info, the system property setting is ignored. • If you do not execute any SQL statement that uses input or input/output ? parameters, we recommend that you specify a value of 1. • This property value is applicable only when the version of the connected HiRDB server is 07-02 or later.	O
HiRDB_for_Java_SQL_OUT_NUM	Specifies the maximum number of output items for the SQL statement to be executed. This is the number of output items that is acquired during SQL preprocessing. If the actual number of output items is greater than this property value, the output items are acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or non-numeric value results in an error. Note: • This item can also be specified using the HiRDB_for_Java_SQL_OUT_NUM system property. However, when HiRDB_for_Java_SQL_OUT_NUM is specified for Properties info, the system property setting is ignored. • If you do not execute any SQL statement that contains a search item or output or input/output ? parameter, we recommend that you specify a value of 1. • This property value is applicable only when the version of the connected HiRDB server is 07-02 or later.	O
HiRDB_for_Java_SQLWARNING_LEVEL	Specifies the retention level of warning information that has been issued during execution of the SQL statement. The permitted warning retention levels are as follows: • IGNORE • SQLWARN (default) • ALLWARN In this method, information specified in the arguments is not case sensitive. For details about the above values, see *17.2.9 SQLWarning class*.	O

Key	Description	Specification
HiRDB_for_Java_CLEAR_ENV	Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection. TRUE: Ignores the HiRDB client environment definition registered as OS environment variables when the database is connected for the first time after the process has started. When TRUE is specified, you can apply the value of the HiRDB client environment definition that has been set by a method other than the OS environment variables (such as environment variable groups). FALSE (default): Does not ignore the HiRDB client environment definition registered as OS environment variables. Notes: • In this method, information specified in the arguments is not case sensitive. • Once the database is connected, the HiRDB client environment definition set as OS environment variables is not ignored even if an attempt is made to specify TRUE within a native method installed by a method such as C language. • Once the database is connected with TRUE specified, the client environment definition value remains ignored even if FALSE is specified the next time the database is connected.	O

Legend:

R: Required.

O: Optional.

#1: If null or space characters are specified for the authorization identifier, this method throws an SQLException. This method also throws an SQLException if the driver-converted character codes and, as a result, the size of the character string specified for the authorization identifier exceed 30 bytes. For details about character code conversion, see *17.12.2 Character code conversion facility*

#2: The key can be omitted, if specified with the internal driver.

Notes on COMMIT_BEHAVIOR

- If another user specifies CLOSE or PRESERVE to execute a definition SQL statement on a resource (such as a table or index) that is being accessed by SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL, and the PDDDLDEAPRPEXE and PDDDLDEAPRP client environment definitions are both set to NO, the definition SQL statement is placed on lock-release wait status until the resource is disconnected.

  If YES is set in the PDDDLDEAPRPEXE or the PDDDLDEAPRP client environment definition, the preprocessing result becomes invalid. If an SQL for which the preprocessing result has been invalidated in this manner is executed, an SQLException exception occurs (the value acquired by the getErrorCode method is -1542).

- When PRESERVE is specified, the JDBC driver uses HiRDB's holdable cursor.
- By specifying[#1] CLOSE or PRESERVE, the only precompiled SQL statements that are valid after commit[#2] are SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, and CALL (SQL statements can be precompiled by executing the Connection.prepareStatement method or the Connection.prepareCall method).

  Other precompiled SQL statements become invalid during commit even though you specify CLOSE or PRESERVE for COMMIT_BEHAVIOR.

  When SQL statements that include these invalid SQL statements are executed with the PreparedStatement class object or CallableStatement object, an error occurs. An example of such an error is shown below:

  Example

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();         //No error occurs.
con.commit();
pstmt2.execute();         //An error occurs.
pstmt1.close();
pstmt2.close();
```

Explanation

Because the SQL statement to be executed is a LOCK statement, even though COMMIT_BEHAVIOR specifies CLOSE, PreparedStatement becomes invalid after commit and an error occurs.

#1: Refers to specification of one of the following:

- COMMIT_BEHAVIOR=CLOSE specified for the URL specified by the getConnection method.
- COMMIT_BEHAVIOR=PRESERVE specified for the URL specified by the getConnection method.
- setCommit_Behavior method of the JdbhDataSource,

JdbhConnectionPoolDataSource, or JdbhXADataSource class used to specify CLOSE.

- setCommit_Behavior method of the JdbhDataSource, JdbhConnectionPoolDataSource, or JdbhXADataSource class used to specify PRESERVE.

#2: Means one of the following:

- Explicit commit using the commit method
- Implicit commit by automatic commit
- Execution of a definition SQL statement
- Execution of a PURGE TABLE statement
- Explicit rollback by rollback method
- Implicit rollback by an SQL execution error

(3) URL syntax

This section explains the URL syntax supported by the JDBC driver. Do not place any space inside each item or between items in a URL. To specify both an additional connection information item and a database host name item, separate them by a comma (,).

(a) URL syntax

```
jdbc:hitachi:PrdbDrive[://[DBID=additional-connection-information]
                      [[{://|,}]DBHOST=database-host-name]
                      [[{://|,}]ENCODELANG=conversion-character-set]
                      [[{://|,}]COMMIT_BEHAVIOR=cursor-operation-mode]
                      [[{://|,}]CLEAR_ENV=environment-variable-invalidation-setting]]
```

(b) URL items

jdbc:hitachi:PrdbDrive

This is the protocol name and the subprotocol name. This item is required.

additional-connection-information

Specify HiRDB's port number (this corresponds to PDNAMEPORT in the client definitions). Alternatively, specify a HiRDB environment variable group.

If this item is omitted, the default value for PDNAMEPORT is assumed.

Notes about specifying a HiRDB environment variable group in additional connection information

- When you specify the name of a HiRDB environment variable group, place @ at the beginning of the group name.

- If the environment variable name contains single-byte spaces or single-byte @ characters, enclose the name in single-byte quotation marks ("). When an environment variable group name is enclosed in single-byte quotation marks, all characters following the last single-byte quotation mark up to the next item or all characters through the end of the character string are ignored. An environment variable group name containing single-byte quotation marks or single-byte commas cannot be specified.
- The environment variables registered in an environment variable group have precedence over the user environment variables and the environment variables registered by HiRDB.INI.
- The following priority applies to the specification of additional connection information and database host name:

  1. HiRDB environment variable group specified in the additional connection information

  2. Database host name or the port number specified in the additional connection information

  For example, if a HiRDB environment variable group name has been specified in DBID, information about the HiRDB environment variable group takes effect. A database host name does not take effect even if it is specified in DBHOST in the URL. In this case, if PDHOST is omitted in the HiRDB environment variable group, a connection error results.

database-host-name

Specify HiRDB's host name. This corresponds to PDHOST in the client definitions.

If this item is omitted, the default value for PDHOST is assumed.

conversion-character-set

Specify the conversion character set to be used for character type conversion.

cursor-operation-mode

Specify whether the cursor is valid following COMMIT.

environment-variable-invalidation-setting

Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection. For details about the specification value and notes, see HiRDB_for_Java_CLEAR_ENV in *Table 17-3*.

(c) Example of specifying a HiRDB environment variable group name in additional connection information

- In UNIX

In this example, the path of the HiRDB environment variable group name is `/HiRDB_P/Client/HiRDB.ini`:

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini";
```

- In Windows

1. In this example, the environment variable group name registered using the tool for registering HiRDB client environment variables is `HiRDB_ENV_GROUP`:

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=HiRDB_ENV_GROUP";
```

2. In this example, the path of the HiRDB environment variable group name is `C:\HiRDB_P\Client\HiRDB.ini`:

```
String url = "jdbc:hitachi:PrdbDrive://
DBID=@HIRDBENVGRP=C:\\HiRDB_P\\Client\\HiRDB.ini";
```

3. In this example, the path of the HiRDB environment variable group name is `C:\ProgramΔFiles\HITACHI\HiRDB\HiRDB.ini` (Δ: single-byte space character):

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=" +
           "\"C:\ProgramΔFiles\\HITACHI\\HiRDB\\HiRDB.ini\"";
```

17.2.2 Connection class

(1) Overview

The `Connection` class provides the following functions:

- Creation of objects in the `Statement`, `PreparedStatement`, and `CallableStatement` classes
- Transaction settlement (`COMMIT` or `ROLLBACK`)
- Specification of `AUTO` commit mode

For details about and usage of each method provided with the `Connection` class, see the applicable JDBC manual.

(2) Notes

(a) Catalog

The JDBC driver does not use a catalog regardless of the connected database type. Therefore, the `getCatalog` method unconditionally returns the `null` value, and the `setCatalog` method does nothing.

(b) Access mode

The JDBC driver does not allow the access mode to be changed. Therefore, the `isReadOnly` method unconditionally returns `false`, and the `setReadOnly` method processes nothing.

(c) Transaction isolation mode

The JDBC driver does not allow the transaction mode to be changed. Therefore, the `getTransactionIsolation` method unconditionally returns `TRANSACTION_READ_COMMITTED`, and the `setTransactionIsolation` method does nothing.

17.2.3 Statement class

(1) Overview

The `Statement` class provides the following functions:

- SQL execution
- Creation of a result set (`ResultSet` object) as a retrieval result
- Return of the number of updated rows as an updating result

For details about and usage of each method provided with the `Statement` class, see the applicable JDBC manual.

(2) Notes

(a) Multi-thread

To use a single `Statement` object with multiple threads, a series of processing, such as SQL execution, acquisition of result set, and closing of the result set, needs to be serialized per thread. If they are processed in parallel, operation cannot be guaranteed. Therefore, you should allocate a separate `Statement` object for each thread.

(b) Cursor name

The JDBC driver does not support positioned updating or deletion. Therefore, the `setCursorName` method does nothing.

(c) Limitation of retrieval time

The JDBC driver does not support the monitoring of a retrieval time. Therefore, the `setQueryTimeout` method, if specified, is ignored.

(d) Specification of the maximum number of rows to be retrieved

The maximum number of rows to be retrieved cannot be specified in the JDBC driver.

17.2.4 PreparedStatement class

(1) Overview

The `PreparedStatement` class provides the following functions:

- Execution of SQL specifying the `?` parameter
- Specification of the `?` parameter
- Generation and return of the `ResultSet` object as a search result
- Return of the number of updated rows as an updating result

Because the `PreparedStatement` class is a subclass of the `Statement` class, it inherits all of the `Statement` class functions.

For details about and usage of each method provided with the `PreparedStatement` class, see the applicable JDBC manual.

(2) Notes

All of the notes about the `Statement` class are applicable to the `PreparedStatement` class, because the `PreparedStatement` class is a subclass of the `Statement` class. The following describes the other notes about the `PreparedStatement` class.

(a) Specification of the ? parameter

For details about the `set`*XXX* method used to set the `?` parameter, see *17.3.3(2) Data mapping when the ? parameter is specified*. For details about the JDBC SQL types supported by the connected database, see *17.12 Data types and character codes*.

(b) Multiple result sets

The function for returning multiple result sets is not available. Therefore, the `getMoreResults` method unconditionally returns `false` and closes any currently open result set.

17.2.5 CallableStatement class

(1) Overview

The `CallableStatement` class provides the following functions:

- Execution of Java stored routines

- Specification of IN and INOUT parameters (a set*XXX* method of the PreparedStatement class is used)
- Registration of OUT and INOUT parameters
- Acquisition of OUT and INOUT parameters
- Acquisition of a result set

Because the CallableStatement class is a subclass of the PreparedStatement class, it inherits all of the PreparedStatement class functions and the Statement class functions. Note that the result set obtained with the DatabaseMetaData class within a Java stored routine can be used only within the Java stored routine. The CallableStatement class's getResultSet cannot acquire it as a dynamic result set.

For details about and usage of each method provided with the CallableStatement class, see the applicable JDBC manual.

(2) Notes

1. All of the notes about the PreparedStatement and Statement classes applicable to the CallableStatement class, because the CallableStatement class is a subclass of the PreparedStatement class.
2. Parameter information is erased when the clearParameters method is executed. If the clearParameters method is executed after execution of the execute method but before execution of the get*XXX* method, the KFPJ20506-E message is output when the get*XXX* method is executed.
3. If you use the INOUT parameter of a Java stored routine, the java.sql.Types specified using the registerOutParameter method and the data type set using the set*XXX* method must be the same.

17.2.6 ResultSet class

(1) Overview

The ResultSet class provides the following functions:

- Moving within a result set in units of rows
- Returning resulting data
- Issuing a message indicating whether or not the retrieval result data is the NULL value

For details about and usage of each method provided with the ResultSet class, see the applicable JDBC manual.

(2) Notes

(a) Multi-thread

If a single ResultSet object is used with multiple threads in parallel, operation cannot be guaranteed. Therefore, you should process a single ResultSet object by a single thread.

(b) Data mapping (conversion)

For details about the get*XXX* method used during the acquisition of results, see *17.3.3(1) Data mapping during retrieval data acquisition*. For details about the JDBC SQL types supported by the connected database, see *17.12 Data types and character codes*.

17.2.7 ResultSetMetaData class

(1) Overview

The ResultSetMetaData class provides the following functions:

- Returning meta-information, such as data type and length of each column, in ResultSet (result set)

(2) Details of method

(a) isSearchable (int column) method

true is returned if the column specified by the column parameter can be used for the WHERE clause; otherwise, false is returned as the return value. For the WHERE clause, true is always returned to enable use of all data type columns. However, for the first column of ResultSet, which is the return value of the Array.getResultSet method, false is returned. For details about getResultSet, see *17.6 Array class*.

Example:

Column C1 is in table T1. Regardless of its data type, C1 can be used in the WHERE clause as shown below:

```
SELECT * FROM T1 WHERE LENGTH (C1) > 5
```

(b) getColumnDisplaySize (int column) method

The return value is the maximum number of characters when the column specified by the column parameter is expressed in a character string. However, for the first column of ResultSet, which is the return value of the Array.getResultSet method, 10 is returned. The following table lists the return values for this method for each SQL data type in HiRDB.

Table 17-4: Return values of the getColumnDisplaySize method for each SQL data type in HiRDB

SQL data type in HiRDB	Return value (int)	Return value format
`INTEGER`	11	1 sign character + 10 digits, which is the maximum number of digits
`SMALLINT`	6	1 sign character + 5 digits, which is the maximum number of digits

`DECIMAL(`*m*`,`*n*`)`
`NUMERIC(`*m*`,`*n*`)`
• *m*: Accuracy (total number of digits)
• *n*: Decimal scaling position (number of digits after decimal point)
	m + 2	1 sign character + accuracy m + 1 decimal point digit

`FLOAT`
`DOUBLE PRECISION`
	23	1 sign character + 17 digits, which is the maximum number of significant digits + 1 decimal point character + 4, which is the maximum number of characters in the index area

`SMALLFLT` `REAL`	13	1 sign character + 8 digits, which is the maximum number of significant digits + 1 decimal point character + 3, which is the maximum number of characters in the index area
`CHAR(`*n*`)` • *n*: Number of bytes of the definition length	*n*	NA

`VARCHAR(`*n*`)`
`CHAR VARYING(`*n*`)`
• *n*: Number of bytes of the maximum length
	n	NA

`NCHAR(`*n*`)`
`NATIONAL CHAR(`*n*`)`
• *n*: Number of characters of the definition length
	n	NA

SQL data type in HiRDB	Return value (int)	Return value format
NVARCHAR(*n*) NATIONAL CHAR VARYING(*n*) NCHAR VARYING(*n*) • *n*: Number of characters of the maximum length	*n*	NA
MCHAR(*n*) • *n*: Number of bytes of the maximum length	*n*	NA
MVARCHAR(*n*) • *n*: Number of bytes of the maximum length	*n*	NA
DATE	10	*yyyy*-*mm*-*dd*, which is 10 characters
TIME	8	*hh*:*mm*:*ss*, which is 8 characters
TIMESTAMP(*p*) • *p*: Number of digits of the fractional part of the second	(1) When *p* is 0: 19 (2) When *p* is 2, 4, or 6: 20 + *p*	(1) *yyyy*-*mm*-*dd* *hh*:*mm*:*ss*, which is 19 characters (2) 19 characters shown above + 1 decimal point character + *p*, which is the number of digits of the decimal part
BLOB(*n*[*K*|*M*|*G*]) • *n*: Maximum length • *K*: Kilobytes • *M*: Megabytes • *G*: Gigabytes If the unit is omitted, bytes is assumed.	When the unit specification is omitted: *n* When *K* is specified for the unit: *n* x 1024# When *M* is specified for the unit: *n* x 1024 x 1024# When *M* is specified for the unit: *n* x 1024 x 1024 x 1024#	NA
BINARY(*n*) • *n*: Number of bytes of the maximum length.	N	NA

Legend:

NA: Not applicable.

#: 2147483648 is assumed for any calculation result greater than 2147483648.

17.2.8 DatabaseMetaData class

The `DatabaseMetaData` class provides the following functions:

- Returning various information about a connected database
- Storing or returning listing information (such as a list of tables or columns) in `ResultSet` (result set)

Note that the result set obtained with the `DatabaseMetaData` class within a Java stored routine can be used only within the Java stored routine.

For details about the methods provided by the `DatabaseMetaData` class and how to use them, see the applicable JDBC documentation. For details about the values that are actually returned, see *17.13 Classes and methods with limitations*. Note that the value returned by each method is information related to the HiRDB server, whose version has to be the same as that of the JDBC driver being used.

17.2.9 SQLWarning class

(1) Overview

The `SQLWarning` class provides the following function:

- Providing information about database access warnings

The `SQLWarning` object is accumulated without an issuance of exception in the method object that caused the warning.

(2) Notes

(a) Releasing the accumulated SQLWarning object

The `SQLWarning` object is accumulated by chain from the method object that caused the warning (`Connection`, `Statement`, `PreparedStatement`, `CallableStatement`, or `ResultSet`).

To explicitly release the accumulated `SQLWarning` object, you must execute `clearWarnings` from the object connecting the chain.

(b) SQLWarning object generation conditions

If the warnings caused by execution of SQL statements are to be retained in the JDBC driver according to the warning retention level specification, the `SQLWarning` objects are generated and the warning information is retained. The following table describes the `SQLWarning` generation conditions:

Execution result of SQL statement	**Warning retention level**		
	IGNORE	**SQLWARN**	**ALLWARN**
`SQLCODE` is greater than 0 and is not 100, nor 110, nor 120	N	N	Y

Execution result of SQL statement	Warning retention level		
	IGNORE	SQLWARN	ALLWARN
SQLWARN0 in the SQL Communications Area is W (except when SQLWARN6 is W)	N	Y	Y
Warning occurred in the JDBC driver	N	Y	Y

Legend:

Y: Generated

N: Not generated

Note

You can specify the warning retention level using the HiRDB_for_Java_SQLWARNING_LEVEL property or the setSQLWarningLevel method. The default is SQLWARN.

(c) Warning message

The following table presents the messages that can be acquired from SQLWarning:

Condition	Message acquired by getMessage
SQLWARN0 is W	KFPJ01074-W
SQLWARN0 is 'Δ' and SQLCODE is greater than 0 (except when SQLCODE=100, 110, or 120)	KFPA*XXXXX*-X
Warning occurred in the JDBC driver	KFPJ*XXXXX*-W

(d) Batch updating

When warning occurs during updating of multiple rows during batch updating, only one SQLWarning is generated.

17.3 JDBC2.0 basic facility

17.3.1 Result set enhancements

The JDBC2.0 basic standard has added scroll and parallel processing as the extended features of result sets (`ResultSet` class).

(1) Scroll types

There are three different scroll types for result sets:

(a) Forward-only type

This is the standard scroll type from JDBC1.0. It allows a result set to be scrolled in the forward direction only (from top to bottom).

(b) Scroll-insensitive type

This is a new scroll type added with JDBC2.0. It allows a result set to be scrolled in a forward or backward direction. It also allows a movement specifying a location relative to the current location or a movement to an absolute location.

Scroll-insensitive means that a change made while a result set is open does not take effect on the result set. In other words, the scroll-insensitive type provides a static view of base data. The rows contained in a result set, their order, and column values are all fixed when the result set is created.

(c) Scroll-sensitive type

This is a new scroll type added with JDBC2.0. While a result set is open, any change made takes effect on the result set.

Changes that take effect may be made directly to the current result set, or made by another result set within the same transaction, or made by another transaction. The number of changes applied depends on the driver's implementation level and DBMS transaction cut-off level.

(2) Parallel processing type

There are two different parallel processing types for result sets:

(a) Read-only type

This is the standard parallel processing type from JDBC1.0. It does not allow data to be updated from its result set.

(b) Updatable type

This is a new parallel processing type added with JDBC2.0. It allows data to be updated (`UPDATE`, `INSERT`, and `DELETE`) from its result set.

(3) Types of result set

When the scroll type and parallel processing type are combined, there are six result set types. Specify the result set type to acquire an instance of the `Statement` class (or its subclass) using the `createStatement` method, `prepareStatement` method, or `prepareCall` method of the `Connection` class.

The following table shows the availability of the result set type when you use the JDBC driver.

Table 17-5: Availability of result set types with JDBC driver

Result set type		Availability with JDBC driver
Scroll type	**Parallel processing type**	
Forward-only	Read-only	A
	Updatable	NA
Scroll-insensitive	Read-only	A
	Updatable	NA
Scroll-sensitive	Read-only	NA
	Updatable	NA

Legend:

A: Available.

NA: Not available.

Notes

1. An error occurs if an unavailable result set is specified. In this case, the JDBC driver creates an instance of the `Statement` class (or its subclass) using the result set that is closest to the specified type, then stores a warning message in `SQLWarning` of the `Connection` class.
2. Some of the methods in the `ResultSet` class are not available because the JDBC driver does not provide the updatable parallel processing type. If such an unavailable method is called, the JDBC driver unconditionally throws an `SQLException`. For details about the unavailable methods, see *17.13 Classes and methods with limitations*.

(4) Notes about using scroll-type result sets

A scroll-type result set caches all retrieval data in the JDBC driver. If there is a large amount of data, a memory shortage or performance reduction may occur. Therefore, to use a scroll-type result set, you must suppress the retrieval data volume in advance by adding a condition to an SQL statement, for example.

17.3.2 Batch updating

The JDBC2.0 basic standard adds the batch updating feature to the `Statement`, `PreparedStatement`, and `CallableStatement` classes. The batch update facility enables multiple SQL statements or multiple parameter values to be registered for batch execution.

To use the batch update facility, you need to set the `Connection` class's `AUTO` commit mode to off. This is because, if an error occurs during the batch updating, the application needs to control the transaction's validity. If the `AUTO` commit mode is on (initial status) and an error occurs during the batch updating, the SQL execution immediately preceding the error takes effect.

When you execute batch updating, you can use HiRDB facilities using arrays.

The facilities using arrays are useful for updating a large amount of HiRDB data at high speed. For details about the facilities using arrays, see *4.8 Facilities using arrays*.

Notes about using the facilities using arrays

1. The facilities using arrays are supported by HiRDB version 07-01 or later.
2. During `Connect`, you must specify the `BLOCK_UPDATE=TRUE` property (if `DataSource` is used, specify `setBlockUpdate(true)`) or `setBlockUpdate(true)` in `JdbcDbpsvPreparedStatement`.
3. If you specify the `HiRDB_for_Java_BLOCK_UPDATE=TRUE` system property, you can enable the array facilities. For details about `HiRDB_for_Java_BLOCK_UPDATE`, see *BLOCK_UPDATE* in *Table 17-3*.
4. The SQL statement to be executed must contain at least one `?` parameter (this does not apply to stored procedures). Additionally, you must use the `addBatch()` method of the `CallableStatement` class or the `PreparedStatement` class (using the `addBatch(String sql)` method of the `Statement` class results in a HiRDB error).

   Executable SQL statements include `INSERT`, `UPDATE`, and `DELETE`. All other SQL statements are executed sequentially, not in batch mode.
5. There must be two or more parameter sets that have been registered by the `addBatch()` method. If there is only one parameter set, it is processed normally, not in batch mode. If there are more than 30,000 parameter sets, each group of 30,000 parameter sets is executed at one time.
6. If the length of `BINARY` data specified in the `?` parameter is 32,001 bytes or greater, sequential execution takes place because facilities using arrays are not applied.
7. If the length of data specified for HiRDB `BLOB`-type columns is 32,001 bytes

or greater, sequential execution takes place because facilities using arrays are not applied.#2

8. Make sure that the same data type is specified for each and every column.#1
9. When DECIMAL-type data is inserted, the precision and scaling of the DECIMAL-type data specified for array are replaced by HiRDB's table definition attributes. If the length of integer part of the DECIMAL-type data specified for array is greater than that of the HiRDB table definition attribute, an overflow occurs, resulting in an error.
10. If you specify HiRDB's repetition column in the ? parameter, you cannot use the facilities using arrays.
11. If an error occurs during batch updating with facilities using arrays, the execution results immediately preceding the error are ignored.
12. Facilities using arrays cannot be used from the basic Cosminexus J2EE server mode.
13. When facilities using arrays are used from Cosminexus, the setBlockUpdate method of PreparedStatement is not available.
14. When a large amount of data is updated using the addBatch function, a large amount of Java memory is used. Depending on the performance of Java memory, the advantages of batch updating may not be obtained. When you use a large amount of data, specify a heap size at the start of Java (java -Xms32m JavaUP: set the Java heap at the start of Java to 32 megabytes).

#1: For example, if you use setInt() to specify the first addBatch for the column 1 data, you must also use setInt() for the subsequent addBatch.

#2: If you use facilities using arrays and specify the ? parameter for HiRDB's BLOB-type columns, note the following:

- If the length of data specified in the ? parameter is less than 32,001 bytes, the data is treated as BINARY-type data in the JDBC driver, thereby executing facilities using arrays. If the length is 32,001 bytes or greater, facilities using arrays are not executed.

(1) Batch updating with the Statement class

Following are notes about batch updating with the Statement class:

- Use the addBatch method to register multiple updating SQL statements.
- Use the executeBatch method to execute the registered updating SQL statements in batch mode.
- An array of the number of rows updated by each updating SQL statement is returned as the batch execution result.

- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If a retrieval SQL statement is registered, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.

The JDBC driver executes registered SQL statements sequentially because it cannot execute them in batch mode.

(2) Batch updating with the PreparedStatement class

Following are notes about batch updating with the `PreparedStatement` class:

- Use a normal procedure (`set`*XXX* method) to specify the `?` parameter for an updating SQL statement that is specified during the creation of a `PreparedStatement` instance.
- Use the `addBatch` method to register `?` parameter sets.
- Use the `executeBatch` method to execute the registered multiple`?` parameter sets in batch mode.
- An array of the number of rows updated by each `?` parameter set is returned as the batch execution result.
- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If a retrieval SQL statement is specified during the creation of a `PreparedStatement` instance, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.

When facilities using arrays are used, the JDBC driver can execute multiple lines of `?` parameters in batch mode. When facilities using arrays are not used, multiple lines of `?` parameters are executed sequentially.

Notes

- If you use HiRDB facilities using arrays, see the notes in *17.3.2 Batch updating*.
- In the second or subsequent `addBatch`, if there are not enough parameters to be specified in the `set`*XXX* method, the previous values are inherited. The following shows an example.

Example: When there are 2 `INTEGER`-type columns (columns 1 and 2)

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
prepstmt.executeBatch();
```

Explanation

- The values that are set in the first `addBatch` are `100` for both columns 1 and 2.

  If there are not enough parameters in the first `addBatch`, an error occurs.

- The values that are set in the second `addBatch` are `200` for column 1 and `100` for column 2.

  Because information for column 2 has not been updated by the second `addBatch`, information for the first `addBatch` is inherited.

(3) Batch updating with the CallableStatement class

Following are notes about batch updating with the `CallableStatement` class:

- Use a normal procedure (`set`*XXX* method) to specify input parameters for the Java stored routine that is specified during the creation of a `CallableStatement` instance.
- Use the `addBatch` method to register input parameter sets.
- Use the `executeBatch` method to execute the registered multiple input parameter sets in batch mode.
- An array of the values (number of updated rows) that are returned by the Java stored routine executed by each input parameter set is returned as the batch execution result.
- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If the Java stored routine specified during the creation of a `CallableStatement` instance does not return the number of updated rows, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.
- If the Java stored routine specified during the creation of a `CallableStatement` instance has an output parameter or input/output parameter, the JDBC driver throws a `BatchUpdateException` when calling the `addBatch` method.

The JDBC driver cannot execute multiple lines of `?` parameters in stored procedures; therefore, multiple lines of `?` parameters in stored procedures are executed sequentially.

Notes

- Batch updating of stored procedures is supported only for the `IN` parameter. If an `OUT` parameter, `INOUT` parameter, or result set (`ResultSet`) is used, an error results.

- For a stored procedure that returns a result set (`ResultSet`), whether or not it returns a result set is unknown until the stored procedure is executed during batch updating. Therefore, if data is updated within the stored procedure, updated information may be applied.[#1]
- Facilities using arrays are not supported in stored procedures. They are supported only in the SQL statements with `?` parameters.
- In the second or subsequent `addBatch`, if there are not enough parameters to be specified in the `set`*XXX* method, the previous values are inherited.[#2]
- If you use the facilities using arrays, see the notes in *17.3.2 Batch updating*.

#1: For example, if a stored procedure that searches and acquires the result of updating is executed during batch updating, `BatchUpdateException` occurs, but updated information may still be applied.

#2: Example: When there are 2 `INTEGER`-type columns (columns 1 and 2)

```
callstmt.setInt(1,100);
callstmt.setInt(2,100);
callstmt.addBatch();
callstmt.setInt(1,200);
callstmt.addBatch();
callstmt.executeBatch();
```

Explanation

- The values that are set in the first `addBatch` are `100` for both columns 1 and 2.

  If there are not enough parameters in the first `addBatch`, an error occurs.
- The values that are set in the second `addBatch` are `200` for column 1 and `100` for column 2.

  Because information for column 2 has not been updated by the second `addBatch`, information for the first `addBatch` is inherited.

17.3.3 Added data types

Several new JDBC SQL types have been added to JDBC2.0 basic standard. They are as follows:

- `BLOB`
- `CLOB`
- `ARRAY`

- REF
- DISTINCT
- STRUCT
- JAVA OBJECT

Note that the JDBC driver can use only the ARRAY JDBC SQL type.

(1) Data mapping during retrieval data acquisition

Tables 17-6 and *17-7* show the mapping between the get*XXX* methods and JDBC SQL types of ResultSet and CallableStatement.

If a get*XXX* method is called for an unsupported JDBC SQL type, the JDBC driver throws an SQLException. For details about the JDBC SQL types supported by the connected database, see *17.12 Data types and character codes*.

Note that the getCharacterStream method has been added because the getUnicodeStream method is no longer recommended in the JDBC2.0 basic standard.

Table 17-6: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (1/2)

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getByte	M	M	M	M	M	M#2
getShort	R	M	M	M	M	M#2
getInt	M	R	M	M	M	M#2
getLong	M	M	M	M	M	M#2
getFloat	M	M	M	R	M	M#2
getDouble	M	M	R	M	M	M#2
getBigDecimal	M	M	M	M	R	M#2
getBoolean	M	M	M	M	M	M
getString	M	M	M	M	M	R
getBytes	--	--	--	--	--	--
getDate	--	--	--	--	--	M#2

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getTime	--	--	--	--	--	M[#2]
getTimestamp	--	--	--	--	--	M[#2]
getAsciiStream	--	--	--	--	--	M
getUnicodeStream	--	--	--	--	--	M
getBinaryStream	--	--	--	--	--	--
getObject	M	M	M	M	M	M
getCharacterStream	--	--	--	--	--	M
getArray	--	--	--	--	--	--
getBlob	--	--	--	--	--	--
getClob[#1]	--	--	--	--	--	--
getRef[#1]	--	--	--	--	--	--

Legend:

R: Mapping is recommended.

M: Can be mapped.

--: Cannot be mapped.

#1: Not supported by the JDBC driver.

#2: When the data from the character string is converted, any single-byte spaces that precede or follow the character string data acquired from the database are stripped, before the data is converted to the Java data type that is to be returned by the get*XXX* method.

Note the following when you convert data to a Java data type:

- If there is an expression following the decimal point in text string data and the getByte, getInt, getShort, or getLong method is executed, the data following the decimal point is truncated and only the integer digits are converted and returned.
- If character string data contains double-byte characters, the method throws an SQLException. Double-byte characters include double-byte spaces filled when a character string shorter than the definition length of a column is stored in an NCHAR column.

- If overflow occurs when character string data is converted to the Java data type, the method throws an `SQLException`.
- If the execution environment of the UAP is JDK or JRE 1.2 and the character string data uses exponential notation (such as `1.23E-23`) and the `getLong` method or `getBigDecimal` method is executed, the method throws an `SQLException`.

Table 17-7: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (2/2)

getXXX method	JDBC SQL type					
	VARCHA R	DATE	TIME	TIMESTAMP	LONGVARBI NARY	ARRAY
`getByte`	M[#2]	--	--	--	--	--
`getShort`	M[#2]	--	--	--	--	--
`getInt`	M[#2]	--	--	--	--	--
`getLong`	M[#2]	--	--	--	--	--
`getFloat`	M[#2]	--	--	--	--	--
`getDouble`	M[#2]	--	--	--	--	--
`getBigDecimal`	M[#2]	--	--	--	--	--
`getBoolean`	M[#2]	--	--	--	--	--
`getString`	R	M	M	M	M	--
`getBytes`	--	--	--	--	M	--
`getDate`	M[#2]	R[#3]	--	M	--	--
`getTime`	M[#2]	--	R	M	--	--
`getTimestamp`	M[#2]	M	--	R	--	--
`getAsciiStream`	M	--	--	--	M	--
`getUnicodeStream`	M	--	--	--	M	--
`getBinaryStream`	--	--	--	--	R	--
`getObject`	M	M	M	M	M	M
`getCharacterStream`	M	--	--	--	M	--

getXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
getArray	--	--	--	--	--	R
getBlob	--	--	--	--	M	--
getClob#1	--	--	--	--	--	--
getRef#1	--	--	--	--	--	--

Legend:

R: Mapping is recommended.

M: Can be mapped.

--: Cannot be mapped.

#1: Not supported by the JDBC driver.

#2: In data conversion from character string data, any single-byte spaces that exist before and after the character string data is acquired from the database are removed, and then the data is converted to the Java data type returned by the `get`*XXX* method.

Note the following when you convert data to a Java data type:

- If there is an expression following the decimal point in text string data and the `getByte`, `getInt`, `getShort`, or `getLong` method is executed, the data following the decimal point is truncated and only the integer digits are converted and returned.
- If character string data contains double-byte characters, the method throws an `SQLException`. Double-byte characters include double-byte spaces filled when a character string shorter than the definition length of a column is stored in an NCHAR column.
- If overflow occurs when character string data is converted to the Java data type, the method throws an `SQLException`.
- If the execution environment of the UAP is JDK or JRE 1.2 and the character string data uses exponential notation (such as `1.23E-23`) and the `getLong` method or `getBigDecimal` method is executed, the method throws an `SQLException`.

#3: When the JDBC SQL type is the `DATA` type and conversion is executed by specifying a `java.util.Calendar` object for the `setDate` method, the specified `java.util.Calendar` object is used for data conversion, time data is truncated, and only date data is stored in the database. In such a case, even if you specify the java.util.Calendar object for the getDate method to acquire the data stored using the setDate method, a different date than the date specified for the setDate method may be acquired.

Example:

The following is an example of when a java.util.Calendar object using Universal Time (UTC) is specified for the setDate and getDate methods in a UAP that uses Japanese standard time as the default time zone.

When you specify a java.sql.Date object that represents 2005-10-03 for the setDate method and then execute it, the JDBC driver adds 00:00:00 in the time part, and then stores the date part as 2005-10-02 in the database by delaying 9 hours because of the time zone difference. If this data is acquired using the getDate method, the date part 2005-10-02 is acquired from the database and 00:00:00 is added for the time part, and then 2005-10-02 09:00:00 is set by advancing 9 hours because of the time zone difference. Because of this, 2005-10-02 is set in the java.sql.Date object return value of the getDate method, which is different from the 2005-10-03 date specified for the setDate method.

(2) Data mapping when the ? parameter is specified

The table below shows the set*XXX* methods of the PreparedStatement and CallableStatement classes and the JDBC SQL types to be mapped. For an unsupported JDBC SQL type, the set*XXX* method throws an SQLException. For details about the JDBC SQL types supported by the connected database, see *17.12 Data types and character codes*.

Note that the setCharacterStream method has been added because the setUnicodeStream method is no longer recommended in the JDBC2.0 basic standard.

Table 17-8: setXXX methods and JDBC SQL types to be mapped for PreparedStatement class

PreparedStatement class's setXXX method	JDBC SQL type to be mapped
setCharacterStream	CHAR, VARCHAR, or LONGVARCHAR
SetRef#	REF
setBlob	LONGVARBINARY
setClob#	CLOB
setArray	ARRAY

#: Not supported by the JDBC driver.

Tables 17-9 and *17-10* show the mapping between the set*XXX* methods and JDBC SQL types of `PreparedStatement` and `CallableStatement`.

Table 17-9: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (1/2)

setXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
`setByte`	M	M	M	M	M	M
`setShort`	R	M	M	M	M	M
`setInt`	M	R	M	M	M	M
`setLong`	M	M	M	M	M	M
`setFloat`	M	M	M	R	M	M
`setDouble`	M	M	R	M	M	M
`setBigDecimal`	M	M	M	M	R	M
`setBoolean`	M	M	M	M	M	M
`setString`	M	M	M	M	M	R
`setBytes`	--	--	--	--	--	--
`setDate`	--	--	--	--	--	M
`setTime`	--	--	--	--	--	M
`setTimestamp`	--	--	--	--	--	M
`setAsciiStream`	--	--	--	--	--	M
`setUnicodeStream`	--	--	--	--	--	M
`setBinaryStream`	--	--	--	--	--	--
`setObject`	M	M	M	M	M	M
`setCharacterStream`	--	--	--	--	--	M
`setArray`	--	--	--	--	--	--
`setBlob`	--	--	--	--	--	--
`setClob`#	--	--	--	--	--	--
`setRef`#	--	--	--	--	--	--

Legend:

R: Mapping is recommended.

M: Can be mapped. Note that data may be missing or a conversion error may occur depending on the format of source data.

--: Cannot be mapped.

#: Not supported by the JDBC driver.

Table 17-10: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (2/2)

setXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`setByte`	M	--	--	--	--	--
`setShort`	M	--	--	--	--	--
`setInt`	M	--	--	--	--	--
`setLong`	M	--	--	--	--	--
`setFloat`	M	--	--	--	--	--
`setDouble`	M	--	--	--	--	--
`setBigDecimal`	M	--	--	--	--	--
`setBoolean`	M	--	--	--	--	--
`setString`	R	M	M	M	M	--
`setBytes`	--	--	--	--	M	--
`setDate`	M	R#2	--	M	--	--
`setTime`	M	--	R	M	--	--
`setTimestamp`	M	M	--	R	--	--
`setAsciiStream`	M	--	--	--	M	--
`setUnicodeStream`	M	--	--	--	M	--
`setBinaryStream`	--	--	--	--	R	--
`setObject`	M	M	M	M	M	M

setXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
setCharacterStream	M	--	--	--	M	--
setArray	--	--	--	--	--	R
setBlob	--	--	--	--	M	--
setClob[#1]	--	--	--	--	--	--
setRef[#1]	--	--	--	--	--	--

Legend:

R: Mapping is recommended.

M: Can be mapped. Note that data may be missing or a conversion error may occur depending on the format of source data.

--: Cannot be mapped.

#1: Not supported by the JDBC driver.

#2: When the JDBC SQL type is the `DATA` type and conversion is executed by specifying a `java.util.Calendar` object for the `setDate` method, the specified `java.util.Calendar` object is used for data conversion, time data is truncated, and only date data is stored in the database. In such a case, even if you specify the `java.util.Calendar` object for the `getDate` method to acquire the data stored using the `setDate` method, a different date than the date specified for the `setDate` method may be acquired.

Example:

The following is an example of what happens when a `java.util.Calendar` object using Universal Time (UTC) is specified for the `setDate` and `getDate` methods in a UAP that uses Japanese standard time as the default time zone.

When you specify a `java.sql.Date` object that represents `2005-10-03` for the `setDate` method, and then execute it, the JDBC driver adds `00:00:00` in the time part and then stores the date part as `2005-10-02` in the database by delaying 9 hours because of the time zone difference. If this data is acquired using the `getDate` method, the date part `2005-10-02` is acquired from the database and `00:00:00` is added for the time part, and then `2005-10-02 09:00:00` is set by advancing 9 hours because of the time zone difference. Because of this, `2005-10-02` is set in the `java.sql.Date` object return value of the `getDate` method, which is different from the `2005-10-03` date specified for the `setDate` method.

17.4 JDBC2.0 Optional Package

17.4.1 Database connection using DataSource and JNDI

Database connection using DataSource and JNDI can now be used by the JDBC2.0 Optional Package.

Although it is not essential to use JNDI, using it offers a benefit in that you need to specify the connection information only once. Because DataSource class interface definition and JNDI are not included in JDK as standard features, you need to obtain them from the JavaSoft web site when developing application programs.

To connect to a database using DataSource and JNDI:

1. Create a DataSource object.
2. Set up connection information.
3. Register DataSource in JNDI.
4. Obtain DataSource from JNDI.
5. Connect to the database.

If you do not use JNDI, the operations in Steps 3 and 4 are unnecessary.

If you use JNDI, execute the operations in Steps 1 through 3 only once. Afterwards, you can connect to the database by performing the operations in Steps 4 and 5 only. Furthermore, after the operation in Step 4, you can modify the connection information as needed.

(1) Creating a DataSource object

Generate the DataSource class objects provided by the JDBC driver.

The DataSource class name of the JDBC driver required to generate the DataSource class objects is JdbhDataSource.

A DataSource class object creation example follows:

```
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource() ;
```

(2) Setting up connection information

Call up a connection information setup method for the DataSource object and set up connection information. Because a connection information acquisition method can also be used, you can also check the current connection information. For details on the connection information setup/acquisition method, see *17.11 Connection information setup/acquisition interface*.

(3) Registering DataSource in JNDI

Register the `DataSource` object in JNDI.

In JNDI, you can select a service provider from several that are available.

An example of obtaining a `DataSource` object in JNDI is shown as follows (for Windows). Note that this obtaining example uses File System, which is one of the service providers. For information on other service providers, see the JNDI documentation.

```
 // Generate a DataSource class object provided by the JDBC driver.
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource();

 // Specify connection information.
         ...


 // Obtain system properties.
 Properties sys_prop = System.getProperties() ;

 // Set up properties for the File System service provider.
 sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
              "com.sun.jndi.fscontext.RefFSContextFactory");


 // Set up the directory to be used by the File System service provider.
 // (In this case, the directory is registered under  c:\JNDI_DIR.)
 sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");

 // Update the system properties.
 System.setProperties(sys_prop) ;


 // Initialize JNDI.
 Context ctx = new InitialContext();

 // Register a DataSource class object provided by the HiRDB driver in JNDI
 // under a logical name called jdbc/TestDataSource.
 ctx.bind("jdbc" + "\\" + "TestDataSource", ds);
                       ...
```

Note that the JDBC2.0 specification recommends that the logical name to be registered in JNDI be registered under a subcontext called `jdbc` (`jdbc/TestDataSource` in the registration example).

(4) Obtaining DataSource from JNDI

Obtain the `DataSource` object from JNDI.

An example of obtaining a `DataSource` object from JNDI is shown as follows (for Windows). Note that this obtaining example uses File System, which is one of service providers. For information on other service providers, see the JNDI documentation.

```
// Obtain system properties.
Properties sys_prop = System.getProperties() ;

// Set up properties for the File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set up the directory to be used by the File System service provider.
// (In this case, the directory is registered under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");


// Update the system properties.
System.setProperties(sys_prop) ;

// Initialize JNDI.
Context ctx = new InitialContext();


// Obtain an object with a logical name jdbc/TestDataSource from JNDI.
Object obj = ctx.lookup("jdbc" + "\\" + "TestDataSource") ;

// Cast the extracted object into the DataSource class type.
DataSource ds = (DataSource)obj;
              ...
```

(5) Connecting to the database

Invoke the `getConnection` method for the `DataSource` object.

An example of calling the `getConnection` method follows:

```
DataSource ds

// Obtain a DataSource object from JNDI.
                     ...


// Issue the getConnection method.
Connection con = ds.getConnection();
  or
Connection con = ds.getConnection("USERID", "PASSWORD");#
```

#: The method's arguments (authorization identifier and password) take precedence over the connection information set for the `DataSource` objects. If the necessary connection information is not set for the `DataSource` object and the connection information is invalid or connection with the HiRDB server fails, the `getConnection` method throws an `SQLException`.

You can set connection information again as necessary after the `Datasource` object is obtained from JNDI. In such a case, you must cast the `Datasource` object to the `DataSource` class type provided by the JDBC driver and then set the connection information.

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource hirdb_ds;

// Obtain a DataSource object from JNDI.
                          ...


// Cast the DataSource object to the DataSource class type provided by the JDBC driver.
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource)ds;

// Reset the connection information.
           ...
```

17.4.2 Connection pooling

A function is provided in the JDBC2.0 Optional Package for pooling connections to a database. An overview of connection pooling is provided below:

- Connection pooling has no effect on existing applications. This means that applications do not need to be aware of connection pooling. However, this assumes that the database is not connected by `DriverManager`, but rather by `DataSource` and JNDI provided by the JDBC2.0 Optional Package.
- The connection pooling function itself is outside the functional scope of the JDBC specifications. This is intended to allow the user to select a desired connection pooling function when building a system (the user can create one, use one provided by an APServer vendor, or use one provided by a JDBC vendor).
- With the connection pooling function, the `Datasource` class can be used as an interface with applications. This `DataSource` class is different from the `DataSource` class provided by the JDBC driver.
- With the JDBC driver, the `ConnectionPoolDataSource` class and `PooledConnection` class can be used as an interface with the connection pooling function.
- The `ConnectionPoolDataSource` class provided by the JDBC driver can use the connection information setting and acquisition methods in the same way as the `Datasource` class provided by the JDBC driver.

The following table lists and describes the classes related to connection pools.

Table 17-11: Classes related to connection pools

Class	Overview
`javax.sql.DataSource`	• Provided by a connection pooling function. • Used as the interface to applications during database connection. • Normally, connection pools are controlled in this class. • Normally, registered in JNDI for use. • Different from the `DataSource` class provided by the JDBC driver.
`javax.sql.ConnectionPoolDataSource`	• Provided by the JDBC driver. • Can use a method for setting/acquiring connection information necessary for database connection. • Normally not used directly from an application, and is used by a connection pooling function. • Normally, registered in JNDI for use. • A connection pooling function acquires a `PooledConnection` object from this class of objects.
`javax.sql.PooledConnection`	• Provided by the JDBC driver. • Normally not used directly from an application, and is used by a connection pooling function. • A connection pooling function targets this class of objects for pooling. • A connection pooling function acquires a `Connection` object to be used by an application from this class of objects.
`javax.sql.ConnectionEventListener`	• Provided by a connection pooling function. • A connection pooling function senses a connection pooling trigger by detecting a disconnection or SQL error through this class of objects.

Depending on the JDK version, the interface definition of the classes shown in *Table 17-11* might not be included in the JDK standard; you will need to check the JavaSoft website if you intend to use the connection pooling function.

The following are the package name and class names of the classes provided by the JDBC driver and shown in *Table 17-11*.

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`ConnectionPoolDataSource` class name: `JdbhConnectionPoolDataSource`

`PooledConnection` class name: `JdbhPooledConnection`

Note that the setting of the connection information of the `ConnectionPoolDataSource` class provided by the JDBC driver is the same as the setting of the connection information of the `DataSource` class provided by the JDBC driver.

17.4.3 Distributed transactions

In the JDBC2.0 Optional Package, distributed transactions in cooperation with the transaction manager (TM) based on the XA standard of X/Open are defined as an extension of the connection pooling function. The following provides an overview of distributed transactions:

- Connection pooling has almost no effect on existing applications. However, there are certain restrictions, such as that direct commit is not allowed. Also, as with connection pooling, it is a precondition that database connection is not performed by using `DriverManager`, but rather by using `DataSource` and JNDI introduced by the JDBC2.0 Optional Package.
- As with connection pooling, the transaction linkage function for linking with a TM is outside the functional scope of the JDBC specifications.
- Normally, a transaction linkage function is installed as an extension of a connection pooling function, and uses TM-provided JTA and JTS as the interface with the TM. Note that operations complying with JTA standard 1.0 are not guaranteed.
- In the transaction linkage facility, as with connection pooling, the `DataSource` class can be used as the interface with applications. This `DataSource` class is different from the one provided by the JDBC driver.
- The JDBC driver can use the `XADataSource` class and `XAConnection` class as an interface with the transaction linkage facility. Also, the JDBC driver can use `XAResource` class as an interface with TM.
- As with the `DataSource` class provided by the JDBC driver, the `XADataSource` class provided by the JDBC driver can use the connection information setting/ acquisition methods.

As with connection pooling, `Connection` objects used by applications are generated by the `XAConnection` class. However, there are certain differences compared with the `Connection` objects generated by the `DataSource` class provided by the `PooledConnection` class or JDBC driver.

- The method of invoking a `commit` method or `rollback` method for the `Connection` class is based on an `SQLException`. That is, an application cannot directly complete a transaction.
- The default mode for `AutoCommit` is `OFF`.
- Issuance of a `Connection` class's `setAutoCommit (true)` method that turns on the `AutoCommit` mode results in an `SQLException`.

The following table lists and describes the classes related to distributed transactions.

Table 17-12: Classes related to distributed transactions

Class	Overview
`javax.sql.DataSource`	• Provided by a transaction linkage function. • Used as the interface to applications during database connection. • Normally, linkage to a TM and connection pools are controlled in this class. • Normally, registered in JNDI for use. • Different from the `DataSource` class provided by the JDBC driver.
`javax.sql.XADataSource`	• Provided by the JDBC driver. • Can use a method for setting/acquiring connection information necessary for database connection. • Normally not used directly from an application, and is used by a transaction linkage function. • Normally, registered in JNDI for use. • A transaction linkage function acquires an `XAConnection` object from this class of objects.
`javax.sql.XAConnection`	• Provided by the JDBC driver. • This is a subclass of the `PooledConnection` class. That is, it inherits all methods related to connection pooling. • Normally not used directly from an application, and is used by a transaction linkage function. • A transaction linkage function targets this class of objects for pooling. • A transaction linkage function acquires a `Connection` object to be used by an application from this class of objects.
`javax.sql.ConnectionEventListener`	• Provided by a transaction linkage function. • A transaction linkage function senses a connection pooling trigger by detecting a disconnection or SQL error through this class of objects.
`javax.transaction.xa.XAResource`	• Provided by the JDBC driver. • Can use the XA-related methods used by a TM.
`javax.transaction.xa.Xid`	• Provided by the JDBC driver and TM. • Used as the argument/return value of an `XAResource` class method.

Because the interface definition of the classes listed in *Table 17-12* is not included in JDK as a standard feature, you must acquire them from the JavaSoft website when you develop a transaction linkage facility.

The following are the package names and class names of the classes provided by the JDBC driver and shown in *Table 17-12*.

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`XADataSource` class name: `JdbhXADataSource`

`XAConnection` class name: `JdbhXAConnection`

`XAResource` class name: `JdbhXAResource`

`Xid` class name: `JdbhXid`

Note that the setting of the connection information of the `XADataSource` class provided by the JDBC driver is the same as the setting of the connection information of the `DataSource` class provided by the JDBC driver.

17.5 JAR file access facility

To use a Java stored routine, you need to register the JAR file in HiRDB. This processing takes place via the JDBC driver.

This section explains the class and method names used to register, delete, and re-register JAR files.

17.5.1 Class name

The class name follows:
`JP.co.Hitachi.soft.HiRDB.JDBC.Jdbh_JARAccess`

17.5.2 Method name

(1) Registering a JAR file in HiRDB

(a) Format

```
public void Jdbh_JARInstall(java.sql.Connection    con,
                            String                 JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to register the JAR file.

JarName

Specifies the name of the JAR file.

Specify either the absolute path name or relative path name. You cannot specify a file located in another server machine, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method registers the specified JAR file in HiRDB using the `Java.sql.Connection` object. If HiRDB already contains a file with the same name, an error occurs.

(2) Deleting a JAR file from HiRDB

(a) Format

```
public void Jdbh_JARUnInstall(java.sql.Connection   con,
                              String                JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to delete the JAR file.

JarName

Specifies the name of the JAR file.

You cannot specify an absolute path name, nor a relative path name, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method deletes the specified JAR file from HiRDB using the `Java.sql.Connection` object.

(3) Re-registering a JAR file in HiRDB

(a) Format

```
public void Jdbh_JARReInstall(java.sql.Connection   con,
                              String                JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to re-register the JAR file.

JarName

Specifies the name of the JAR file.

You cannot specify an absolute path name, nor a relative path name, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method re-registers the specified JAR file in HiRDB using the `Java.sql.Connection` object. If HiRDB already contains a file with the same name, it is overwritten (an error does not occur).

17.6 Array class

The JDBC driver can access repetition columns using the Array class. Note the following when using each method:

(1) getArray

- MAP cannot be used.
- The following table lists the object types returned by this method.

Table 17-13: Object types returned by getArray

HiRDB data type	Object type
INTEGER	java.lang.Integer[]
SMALLINT	java.lang.Short[]
DECIMAL	java.math.BigDecimal[]
FLOAT, DOUBLE PRECISION	java.lang.Double[]
SMALLFLT, REAL	java.lang.Float[]
CHAR	java.lang.String[]
VARCHAR	java.lang.String[]
NCHAR	java.lang.String[]
NVARCHAR	java.lang.String[]
MCHAR	java.lang.String[]
MVARCHAR	java.lang.String[]
DATE	java.sql.Date[]
TIME	java.sql.Time[]
TIMESTAMP	java.sql.Timestamp[]

(2) getResultSet

- MAP cannot be used.
- The result set returned by this method includes one row in each array element, and each row has two columns. The second column stores the value of the element, while the first column stores the index of the corresponding element inside the array (the index of the first array element is 1). Rows are arranged in ascending order based on the indexes.

- Closing a statement also closes the result returned by this method.
- The following table shows the attribute values of the result sets returned by this method.

Table 17-14: Attribute values of the result sets returned by getResultSet

ResultSetMetaData class method name	Values returned by the method	
	First column	Second column
`getCatalogName`	`null`	`null`
`getColumnClassName`	`java.lang.Integer`	Depends on the database column attribute.
`getColumnDisplaySize`	`10`	Depends on the database column length.
`getColumnLabel`	`JDBC_Array_Index`	Depends on the database column name.
`getColumnName`		
`getColumnType`	`java.sql.Types.INTEGER`	Depends on the database column attribute.
`getColumnTypeName`	`INTEGER`	
`getPrecision`	`10`	Depends on the database column attribute and column length.
`getScale`	`0`	
`getSchemaName`	`null`	`null`
`getTableName`		
`isAutoIncrement`	`true`	`false`
`isCaseSensitive`	`false`	Depends on the database column attribute.
`isCurrency`		`false`
`isDefinitelyWritable`		
`isNullable`	`java.sql.ResultSetMetaData.` `columnNoNulls`	Depends on the database column attribute.
`isReadOnly`	`true`	`false`
`isSearchable`	`false`	`true`
`isSigned`		Depends on the database column attribute.
`isWritable`		`false`

17.7 Specifying a value when using a repetition column as the ? parameter

This section explains how to specify a value when using a repetition column as the `?` parameter.

To specify a value for the `?` parameter, use the `setObject` method to specify an object in the class in which the Array interface was installed or a column object.

(1) Specifying an object in the class in which the Array interface was installed

- Create an object in the class in which the Array interface was installed, and use the `setArray` or `setObject` method to specify that object.
- The JDBC driver uses the `Array.getBaseType` method to check the data type of that object. If the data type of the database turns out to be different from the data type of the object, the JDBC driver throws an `SQLException`. For details on database and object data types, see *17.12.1 Data types*.
- The actual data is acquired using the `Array.getArray()` method without any argument. The table below shows the object types that must be returned during this data acquisition. If an object type that is returned is not one that is shown in the table, the JDBC driver throws an `SQLException`.

Table 17-15: Object types returned during data acquisition using the Array.getArray() method without any argument

Data type returned by the Array.getBaseType method	Object types returned during data acquisition using the Array.getArray() method without any argument
`java.sql.Types.INTEGER`	`int[]` or `java.lang.Integer[]`
`java.sql.Types.SMALLINT`	`short[]` or `java.lang.Short[]`
`java.sql.Types.DECIMAL`	`java.math.BigDecimal[]`
`java.sql.Types.FLOAT`	`double[]` or `java.lang.Double[]`
`java.sql.Types.REAL`	`float[]` or `java.lang.Float[]`
`java.sql.Types.CHAR`	`java.lang.String[]`
`java.sql.Types.VARCHAR`	`java.lang.String[]`
`java.sql.Types.DATE`	`java.sql.Date[]`
`java.sql.Types.TIME`	`java.sql.Time[]`
`java.sql.Types.LONGVARBINARY`	`java.io.DataInputStream[]`

Data type returned by the Array.getBaseType method	Object types returned during data acquisition using the Array.getArray() method without any argument
`java.sql.Types.TIMESTAMP`	`java.sql.Timestamp[]`

(2) Using the setObject method to specify an array object

- If the database data type is different from the array object data type, the JDBC driver throws an `SQLException`.
- If the data type of the SQL statement specified by the `setObject` method and the data type of the array object differ from those shown in the following table, the JDBC driver throws an `SQLException`.

Table 17-16: Data type of the SQL statement specified by the setObject method and the data type of the array object

Data type of the SQL statement specified by the setObject method	Data type of the array object
`java.sql.Types.INTEGER`	`int[]` or `java.lang.Integer[]`
`java.sql.Types.SMALLINT`	`short[]` or `java.lang.Short[]`
`java.sql.Types.DECIMAL`	`java.math.BigDecimal[]`
`java.sql.Types.FLOAT`	`double[]` or `java.lang.Double[]`
`java.sql.Types.REAL`	`float[]` or `java.lang.Float[]`
`java.sql.Types.CHAR`	`java.lang.String[]`
`java.sql.Types.VARCHAR`	`java.lang.String[]`
`java.sql.Types.DATE`	`java.sql.Date[]`
`java.sql.Types.TIME`	`java.sql.Time[]`
`java.sql.Types.LONGVARBINARY`	`java.io.DataInputStream[]`
`java.sql.Types.TIMESTAMP`	`java.sql.Timestamp[]`

(3) Relationship between repetition column elements and the object specified as the ? parameter

The sequence of the array objects obtained by the `Array.getArray()` method from the objects in the class in which the Array interface was installed is the same as the sequence of the repetition columns. Consequently, the first element of the array object becomes the first element of the repetition column, and the second element of the array object becomes the second element of the repetition column.

The same also holds true for the array objects specified by the `setObject` method. You can also specify an array object consisting of only one element.

(4) Specifying a null value for an element in the middle of a repetition column

Regardless of whether an object is in the class in which the Array interface was installed or an array object, if you specify a null value for an element in the middle of an element, the element of the applicable array becomes null. Therefore, to set a null value for the second element of a repetition column, specify a null value for the second element of the array object obtained by the `Array.getArray()` method from the objects in the class in which the Array interface was installed.

The same also holds true for the array objects specified by the `setObject` method.

17.8 Functions provided by the HiRDB JDBC driver

This section describes the HiRDB JDBC driver functions that are not standardized by JDBC2.0.

17.8.1 Provided class

To use the functions provided only by the HiRDB JDBC driver, you must use the following class:

Interface name	Main function	Class name
`PreparedStatement`	• Executing SQL statements with the `?` parameter • Setting values for the `?` parameter • `Statement` functions (all functions are inherited because this is `Statement`'s subclass)	`JdbcDbpsvPreparedStatement`

17.8.2 setBlockUpdate

(a) Function

`setBlockUpdate` specifies whether or not multiple parameters are to be processed at one time when the `?` parameter is used to update databases.

(b) Format

```
public void setBlockUpdate(boolean Mode)
```

(c) Arguments

`boolean Mode`

Specifies whether or not multiple parameter sets are to be processed at one time. When this argument is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.#

#: During database connection, if `BLOCK_UPDATE=TRUE` is specified in the argument of the `getConnection` method of the `DriverManager` class, the default for this function is `true`. Also, when `HiRDB_for_Java_BLOCK_UPDATE=TRUE` is specified in the system property, the default for this function is `true`.

(d) Return value

None.

(e) Functional detail

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(f) Exception

None.

(g) Notes

For details about how to process multiple lines of `?` parameters in batch mode, see *Table 17-3 Information to be specified for Properties info* and *17.3.2 Batch updating*.

17.8.3 getBlockUpdate

(a) Function

This function acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

(b) Format

```
public boolean getBlockUpdate()
```

(c) Arguments

None.

(d) Return value

`boolean`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.#

#: During database connection, if BLOCK_UPDATE=TRUE is specified in the argument of the getConnection method of the DriverManager class, the default for this function is true.

(e) Functional detail

This function acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the ? parameter (INSERT, UPDATE, or DELETE).

(f) Exception

None.

17.9 Notes on using the BLOB type

This section provides notes about the processing of methods when the BLOB type is used.

(1) Method processing and notes

The following table describes each method's processing.

Table 17-17: Method processing and notes

Method name of Blob interface class	Processing and notes
getBinaryStream	Returns the InputStream class equipped with JdbhInputStream. The maximum length of data that can be acquired is 2,147,483,639.
getBytes(long pos, int length)	Returns the maximum length of data from the specified pos location using the byte[] object. If the database contents are the null value, no data can be acquired from the specified location; or if the data length is 0 bytes, the method returns null. The maximum value is 2,147,483,639. If this length is exceeded, the method throws an SQLException.
length()	Returns the actual data length.
position(Blob pattern,long start)	Executes position(pattern.getBytes(1, (int)(pattern.length())), start). If null is specified in pattern, the method throws a NullPointerException.
position(byte[] pattern,long start)	Returns the position corresponding to pattern from the specified start location. The return value is >=start. If there is no location that corresponds to pattern, the method returns -1. The maximum value of pattern.length is 2,147,483,639. If this value is exceeded, the method throws an SQLException. If null is specified in pattern, the method throws a NullPointerException.
setBinaryStream(long pos)	Unconditionally throws an SQLException.
setBytes(long pos,byte[] bytes)	
setBytes(long pos,byte[] bytes,int offset,int len)	
truncate(long len)	

Note

If you have acquired data using the locator facility and execute ResultSet.close() or Statement.close(), you can no longer acquire data.

(2) Specification method using the ? parameter

To specify a value in the `?` parameter, you can use the `PreparedStatement.setBlob()` and `CallableStatement.setBlob()` methods. This subsection provides notes about using these methods.

(a) When using objects equipped with the Blob interface

When using the `setBlob()` method, you must specify an object equipped with the Blob interface. Additionally, the UAP must create the object equipped with the Blob interface.

JDBC uses the `Blob.getBytes()` method to acquire the value to be set in the `byte[]` format. The following method is used to acquire the value to be used:

```
Blob.getBytes(1, (int)(Blob.length()))
```

In the UAP, the `getBytes()` and `length()` methods must return normal values. JDBC assumes that the values returned by these methods are correct.

(b) When using the Blob object acquired by the ResultSet.getBlob() or CallableStatement.getBlob() method

When the Blob object acquired by the `ResultSet.getBlob()` or `CallableStatement.getBlob()` method as the execution result from JDBC is to be used as is, operation depends on whether or not the object was acquired by using the locator facility for access.

- When the locator facility was not used for access

  The data acquired by the `ResultSet.getBlob()` or `CallableStatement.getBlob()` method is used as the value of the `?` parameter.

- When the locator facility was used for access

  When the `setBlob()` method is called, `Blob.getBytes(1, (int)(Blob.length()))` is executed internally. The data acquired by `Blob.getBytes(1, (int)(Blob.length()))` is used as the value of the `?` parameter.

17.10 Setting system properties

17.10.1 Setting the array facility

(1) Overview

If you set the `HiRDB_for_Java_BLOCK_UPDATE` system property during program execution, you can specify whether or not to process multiple parameter sets at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

(2) Setting method

During program execution, use the `-D` option of the `java` command to set the `HiRDB_for_Java_BLOCK_UPDATE` system property.

(a) Function

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

(b) Format

```
java -D<name>=<value> class-name
```

(c) Description

`name`

`HiRDB_for_Java_BLOCK_UPDATE`

`value`

`TRUE`: Processes multiple parameter sets at one time.

`FALSE`: Processes one parameter set at a time.

Other: Processes one parameter set at a time.

(d) Functional detail

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(e) Notes

- When you specify `-D<`*name*`>=<`*value*`>`, make sure that there is no space in the specified information. The specified information cannot be set correctly if it has any of the following formats, where Δ indicates a space:

  - `-D`Δ*<name>*=*<value>*
  - `-D`*<name>*Δ=*<value>*
  - `-D`*<name>*=Δ*<value>*

- If `BLOCK_UPDATE` is set during database connection (`setBlockUpdate` method during data source connection), `BLOCK_UPDATE` or the value set in the `setBlockUpdate` method takes effect.
- If you used the `PreparedStatement` class's `setBlockUpdate` method, you can change the setting as to whether or not multiple parameter sets are to be processed at one time.
- For details about how to process multiple lines of `?` parameters in batch mode, see *Table 17-3 Information to be specified for Properties info* and *17.3.2 Batch updating*.

(f) Example

The following shows an example of setting the `HiRDB_for_Java_BLOCK_UPDATE` system property:

```
java -DHiRDB_for_Java_BLOCK_UPDATE=TRUE TestUP
```

17.10.2 Setting the maximum number of SQL search items or ? parameters

(1) Overview

If you set the `HiRDB_for_Java_SQL_IN_NUM` or `HiRDB_for_Java_SQL_OUT_NUM` system property during program execution, you can specify the maximum number of search items, output `?` parameters, input `?` parameters, or input/output `?` parameters that are to be acquired during SQL preprocessing.

(2) Setting method

During program execution, set the system property `HiRDB_for_Java_SQL_OUT_NUM` or `HiRDB_for_Java_SQL_IN_NUM` or both in the `-D` option of the `java` command.

(a) Function

This function specifies the maximum number of search items, output `?` parameters, input `?` parameters, or input/output `?` parameters that are to be acquired during SQL preprocessing.

(b) Format

```
java -D<name>=<value> class-name
```

(c) Description

The following table describes the information that can be specified in *<name>* and *<value>*:

<name>	<value>
HiRDB_for_Java_SQL_IN_NUM	Specifies the maximum number of input or input/output ? parameters in the SQL statements to be executed. This is the number of input or input/output ? parameter information items acquired during SQL preprocessing. If the actual number of input or input/output ? parameters is greater than this property value, the input or input/output ? parameter information is acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or a non-numeric value results in an error during database connection.
HiRDB_for_Java_SQL_OUT_NUM	Specifies the maximum number of output items for the SQL statement to be executed. This is the number of output items acquired during SQL preprocessing. If the actual number of output items is greater than this property value, the output items are acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or a non-numeric value results in an error during database connection.

(d) Functional detail

This function specifies the maximum number of search items, output ? parameters, input ? parameters, or input/output ? parameters that are to be acquired during SQL preprocessing. A sufficient value enables you to acquire search item, output ? parameter, input ? parameter, or input/output parameter information during SQL preprocessing, thereby improving performance compared to when this information is acquired after preprocessing.

(e) Notes

- When you specify -D*<name>*=*<value>*, make sure that there is no space in the specified information. The specified information cannot be set correctly if it has any of the following formats, where Δ indicates a space:
  - -DΔ*<name>*=*<value>*
  - -D*<name>*Δ=*<value>*
  - -D*<name>*=Δ*<value>*

- If HiRDB_for_Java_SQL_IN_NUM is set during database connection (setSQLInNum method during data source connection), HiRDB_for_Java_SQL_IN_NUM or the value set in the setSQLInNum method takes effect.
- If HiRDB_for_Java_SQL_OUT_NUM is set during database connection (setSQLOutNum method during data source connection), HiRDB_for_Java_SQL_OUT_NUM or the value set in the setSQLOutNum method takes effect.
- To acquire search item, output ? parameter, input ? parameter, or input/output parameter information during SQL preprocessing, the version of the connected HiRDB server must be 07-02 or later.

(f) Example

The following shows an example of setting the HiRDB_for_Java_SQL_IN_NUM and HiRDB_for_Java_SQL_OUT_NUM system properties:

```
java -DHiRDB_for_Java_SQL_IN_NUM=128
-DHiRDB_for_Java_SQL_OUT_NUM=128 TestUP
```

17.11 Connection information setup/acquisition interface

The JdbhDataSource, JdbhConnectionPoolDataSource, and JdbhXADataSource classes, which are provided by the JDBC driver, provide methods of setting/acquiring the connection information necessary for database connection, besides the methods specified by the JDBC2.0 Optional Package specification.

The following table lists and describes the methods that set and acquire connection information.

Table 17-18: Methods of setting/acquiring connection information

Subsection	Method	Function
17.11.1	setDescription	Sets the additional connection information needed by the database to be connected.
17.11.2	getDescription	Acquires the additional connection information needed by the database to be connected.
17.11.3	setDBHostName	Sets the host name of the HiRDB to be connected.
17.11.4	getDBHostName	Acquires the host name of the HiRDB to be connected.
17.11.5	setEncodeLang	Uses the specified encoding character code to convert data.
17.11.6	getEncodeLang	Returns the encoding characters to be used for data conversion.
17.11.7	setUser	Sets the authorization identifier.
17.11.8	getUser	Acquires the authorization identifier.
17.11.9	setPassword	Sets a password.
17.11.10	getPassword	Acquires a password.
17.11.11	setXAOpenString#	Sets an XA_OPEN character string.
17.11.12	getXAOpenString#	Acquires an XA_OPEN character string.
17.11.13	setXACloseString#	Sets an XA_CLOSE character string.
17.11.14	getXACloseString#	Acquires an XA_CLOSE character string.
17.11.15	setRMID#	Sets a resource manager identifier.
17.11.16	getRMID#	Acquires a resource manager identifier.

Subsection	Method	Function
17.11.17	setXAThreadMode#	Sets a thread mode for using XA.
17.11.18	getXAThreadMode#	Acquires a thread mode for using XA.
17.11.19	setCommit_Behavior	Sets whether a cursor remains valid following COMMIT.
17.11.20	getCommit_Behavior	Acquires whether a cursor remains valid following COMMIT.
17.11.21	setBlockUpdate	Specifies whether or not multiple parameter sets are to be processed at one time.
17.11.22	getBlockUpdate	Acquires a value indicating whether or not multiple parameter sets are to be processed at one time.
17.11.23	setLONGVARBINARY_Access	Specifies the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY).
17.11.24	getLONGVARBINARY_Access	Acquires the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY).
17.11.25	setSQLInNum	Specifies the maximum number of input or input/output ? parameters in the SQL statements to be executed.
17.11.26	getSQLInNum	Acquires the maximum number of input or input/output ? parameters in the SQL statements to be executed that has been set by setSQLInNum.
17.11.27	setSQLOutNum	Specifies the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed.
17.11.28	getSQLOutNum	Acquires the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed that has been set by setSQLOutNum.
17.11.29	setSQLWarningLevel	Specifies the warning retention level that occurred during execution of SQL statements.
17.11.30	getSQLWarningLevel	Acquires the warning retention level specified in setSQLWarningLevel.
17.11.31	setClear_Env	Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection.
17.11.32	getClear_Env	Acquires the environment variable invalidation setting specified by setClear_Env.

#: These methods are provided by the JdbhXADataSource class only.

17.11.1 setDescription

(a) Function

Sets the additional connection information needed by the database to be connected.

(b) Format

```
public void setDescription (String description)
```

(c) Argument

String `description`

Specifies additional connection information.

(d) Return value

None.

(e) Functional detail

Sets the additional connection information needed by the database to be connected. Setting details and whether setting is required are shown as follows.

Setting	Setting details	Setting required?
HiRDB port number	Sets the HiRDB port number expressed as a character string.	Optional
HiRDB environment variable group name	Sets the HiRDB environment variable group name following `@HIRDBENVGRP=`, expressed as a character string. If the environment variable name contains single-byte spaces or single-byte `@` characters, enclose the name in single-byte quotation marks (`"`). When an environment variable group name is enclosed in single-byte quotation marks, all characters following the last single-byte quotation mark through the end of the character string are ignored. An environment variable group name containing single-byte quotation marks or single-byte commas cannot be specified.	Optional
HiRDB environment variable group identifier	Sets the HiRDB environment variable group identifier expressed as four alphanumeric characters.	Required during XA connection

Note 1

The environment variables registered in an environment variable group have precedence over the user environment variables and the environment variables registered by `HiRDB.INI`.

Note 2

Specification examples are shown below. In these examples, `ds` represents the name of a variable that has reference to the `JdbhDataSource` class's instance.

In UNIX:

Example 1: When the path of the HiRDB environment variable group name is `/HiRDB_P/Client/HiRDB.ini`

```
ds.setDescription("@HIRDBENVGRP=/HiRDB_P/Client/
HiRDB.ini");
```

In Windows

Example 1: When specifying the HiRDB port number

```
ds.setDescription("22200");
```

Example 2: When specifying the environment variable group name HiRDB_ENV_GROUP that has been registered using the tool for registering HiRDB client environment variables

```
ds.setDescription("@HIRDBENVGRP=HiRDB_ENV_GROUP");
```

Example 3: When the path of the HiRDB environment variable group name is C:\HiRDB_P\Client\HiRDB.ini

```
ds.setDescription("@HIRDBENVGRP=C:\\HiRDB_P\\Client\\Hi
RDB.ini");
```

Example 4: When the path of the HiRDB environment variable group name is C:\Program△Files\HITACHI\HiRDB\HiRDB.ini (△: single-byte space)

```
ds.setDescription("@HIRDBENVGRP=\"C:\\Program△Files\\H
ITACHI\\HiRDB\HiRDB.ini\"");
```

Example 5: When the HiRDB environment variable group identifier is HDB1

```
ds.setDescription("HDB1");
```

(f) Exception that occurs

If an environment variable group name beginning with `@` is specified during a connection other than the XA connection, and the specified information following `@`

contains a single-byte space, this method throws an `SQLException`.

17.11.2 getDescription

(a) Function

Acquires the additional connection information that was specified by the `setDescription` method and that is needed for the database to be connected.

(b) Format

```
public String getDescription()
```

(c) Argument

None.

(d) Return value

String

This is the additional connection information. If none is set, `null` is returned.

(e) Exception that occurs

None.

17.11.3 setDBHostName

(a) Function

Sets the host name of the HiRDB to be connected (host name set in the `PDHOST` client environment definition).

If the connection is not XA and the environment variable group name of a HiRDB client is specified in the additional connection information, the value specified by this method will be ignored.

(b) Format

```
public void setDBHostName (String db_host_name)
```

(c) Argument

String `db_host_name`

Sets a HiRDB host name.

(d) Return value

None.

(e) Exception that occurs

None.

17.11.4 getDBHostName

(a) Function

Acquires the host name of the HiRDB to be connected that was specified by the `setDBHostName` method.

(b) Format

```
public String getDBHostName()
```

(c) Argument

None.

(d) Return value

String

This is the HiRDB host name. If none is set, `null` is returned.

(e) Exception that occurs

None.

17.11.5 setEncodeLang

(a) Function

Specifies the character set used for character code conversion in the JDBC driver.

(b) Format

```
public void setEncodeLang (String encode_lang)
```

(c) Argument

String `encode_lang`

Specifies a character set supported by Java (such as MS932).

If `OFF` is specified with this method or if nothing is specified (including in the `ENCODELANG` settings of `Properties info` and the URL), the following operation takes place.

`OFF`:

The JDBC driver determines the character set that corresponds to the character codes type of the connected HiRDB. The following table shows the correspondence between the connected HiRDB character codes type and the character encoding used by the JDBC driver:

HiRDB character codes type#	Character encoding used
`lang-c`	`8859_1`
`sjis`	Java Virtual Machine standard encoding
`ujis`	EUCJIS
`utf-8`	UTF-8
`chinese`	GB2312
`chinese-gb18030`	GB18030

#: The specification value is in the `-c` option of the `pdsetup` command for UNIX and the `-c` option of the `pdntenv` command for Windows. For the character codes types when the `pdntenv` command is not executed, see the *HiRDB Version 9 Installation and Design Guide*.

`None`:

For UNIX:

The JDBC driver determines the character set that corresponds to the HiRDB character codes type.

For Windows:

The JDBC driver uses the following rules to determine the character set:

Java Virtual Machine standard encoding	HiRDB character codes type	
	SJIS	Other than SJIS
MS932	MS932	Character set corresponding to the HiRDB character codes type
Other than MS932	SJIS	

(d) Return value

None.

(e) Functional detail

In a Java program, Unicode is used for the character codes. Therefore, during character data processing with HiRDB, the JDBC driver performs mutual character code conversion between the HiRDB character data and Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the

Java Virtual Machine. This method specifies the character set names specified by the JDBC driver for the encoder and decoder that are provided by the Java Virtual Machine.

(f) Exception that occurs

None.

17.11.6 getEncodeLang

(a) Function

Acquires the character set that was specified by the `setEncodeLang` method.

(b) Format

```
public String getEncodeLang()
```

(c) Argument

None.

(d) Return value

String

Returns the character set. If no character set is specified, null is returned.

(e) Exception that occurs

None.

17.11.7 setUser

(a) Function

Sets the authorization identifier.

(b) Format

```
public void setUser (String user)
```

(c) Argument

String `user`

Sets the authorization identifier.

(d) Return value

None.

(e) Functional detail

Sets the authorization identifier.

You can specify the authorization identifier using an argument of the `DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method (which are referred to generically as the *DB connection methods*).

If this method is used to set an authorization identifier, and if a DB connection method that has an authorization identifier and a password set as arguments is also called, the authorization identifier setting specified by the DB connection method takes precedence.

For details about specifying an authorization identifier, see *Table 17-2 Arguments of the getConnection method*.

(f) Exception that occurs

None.

17.11.8 getUser

(a) Function

Acquires the authorization identifier.

(b) Format

```
public String getUser()
```

(c) Argument

None.

(d) Return value

String

Sets the authorization identifier. If no authorization identifier has been set, `null` is returned.

(e) Functional detail

Returns the authorization identifier specified by the `setUser` method.

If the `setUser` method is used to set a password, and if a DB connection method (`DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method) that has an authorization identifier and a password set as arguments is also called, the authorization identifier setting specified by the DB connection method is returned.

(f) Exception that occurs

None.

17.11.9 setPassword

(a) Function

Sets a password.

(b) Format

```
public void setPassword (String password)
```

(c) Argument

String `password`

Specifies a password.

(d) Return value

None.

(e) Functional detail

Sets a password.

You can specify the password using an argument of the `DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method (which are referred to generically as the DB connection methods).

If this method is used to set a password, and if a DB connection method that has an authorization identifier and a password set as arguments is also called, the password setting specified by the DB connection method takes precedence.

(f) Exception that occurs

None.

17.11.10 getPassword

(a) Function

Acquires a password.

(b) Format

```
public String getPassword()
```

(c) Argument

None.

(d) Return value

String

This is a password. If none is set, `null` is returned.

(e) Functional detail

Returns the password specified by the `setPassword` method.

If the `setPassword` method is used to set a password, and if a DB connection method (`DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method) that has an authorization identifier and a password set as arguments is also called, the password setting specified by the DB connection method is returned.

(f) Exception that occurs

None.

17.11.11 setXAOpenString

(a) Function

Sets an `XA` open character string.

(b) Format

```
public void setXAOpenString (String xa_string)
```

(c) Argument

String `xa_string`

Specifies an `XA` open character string.

(d) Return value

None.

(e) Functional detail

Sets an `XA` open character string.

This method is provided by the `JdbhDbpsvXADataSource` class only.

Specify the `XA` open character string in the format

HiRDB-environment-variable-group-identifier+HiRDB-environment-variable-group-name. This HiRDB environment variable group identifier must be the one set in the `setDescription` method. The following shows examples.

Example 1

When setting the environment variable group name `HiRDB_ENV_GROUP` that has been registered by the tool for registering HiRDB client environment variables

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+HiRDB_ENV_GROUP");
```

Example 2

When the path of the HiRDB environment variable group name is `C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini` (∆: single-byte space)

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+C:\\Program∆Files\\HITACHI\\HiRDB
\\HiRDB.ini");
```

(f) Exception that occurs

None.

17.11.12 getXAOpenString

(a) Function

Acquires the XA open character string that was specified by the `setXAOpenString` method. This method is provided by the `JdbhDbpsvXADataSource` class only.

(b) Format

```
public String getXAOpenString()
```

(c) Argument

None.

(d) Return value

String

This is an `XA` open character string. If none is set, `null` is returned.

(e) Exception that occurs

None.

17.11.13 setXACloseString

(a) Function

Sets an `XA` close character string. This method is provided by the `JdbhDbpsvXADataSource` class only.

(b) Format

```
public void setXACloseString (String xa_string)
```

(c) Argument

String `xa_string`

Sets an `XA` close character string.

(d) Return value

None.

(e) Exception that occurs

None.

17.11.14 getXACloseString

(a) Function

Acquires the `XA` close character string that was specified by the `setXACloseString` method. This method is provided by the `JdbhDbpsvXADataSource` class only.

(b) Format

```
public String getXACloseString()
```

(c) Argument

None.

(d) Return value

String

This is an `XA` close character string. If none is set, `null` is returned.

(e) Exception that occurs

None.

17.11.15 setRMID

(a) Function

Sets an identifier for the Resource Manager.

(b) Format

```
public void setRMID (int rmid)
```

(c) Argument

`int rmid`

Specifies an identifier for the Resource Manager.

(d) Return value

None.

(e) Functional detail

Sets a positive numeric value of 1 or greater as the identifier for the Resource Manager.

If multiple Resource Managers are used, a unique identifier must be set for each Resource Manager.

If this method is not invoked, the default identifier of `1` is used.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

If the argument value is smaller than 1, the method throws an `SQLException`.

17.11.16 getRMID

(a) Function

Acquires the identifier for the Resource Manager that was specified by the `setRMID` method. This method is provided by the `JdbhDbpsvXADataSource` class only.

(b) Format

```
public int getRMID()
```

(c) Argument

None.

(d) Return value

`int`

This is an identifier for the Resource Manager. If none is set, `1` is returned.

(e) Exception that occurs

None.

17.11.17 setXAThreadMode

(a) Function

Sets a thread mode for using XA.

(b) Format

```
public void setXAThreadMode (boolean mode)
```

(c) Argument

`boolean mode`

Specifies a thread mode for using XA.

`true`: Multi-thread mode

`false`: Single-thread mode

(d) Return value

None.

(e) Functional detail

Sets a thread mode for using XA. If this method is not invoked, the default value is `false` (single-thread mode).

This method is provided by the `JdbhDbpsvXADataSource` class only.

If the XA library provided by the RM (Resource Manager) supports multi-thread and if the application operates in the multi-thread mode, this method must be invoked in the `true` setting (multi-thread mode).

(f) Exception that occurs

None.

17.11.18 getXAThreadMode

(a) Function

Acquires the thread mode for using XA that was specified by the `setXAThreadMode` method. This method is provided by the `JdbhDbpsvXADataSource` class only.

(b) Format

```
public boolean getXAThreadMode()
```

(c) Argument

None.

(d) Return value

boolean

Specifies a thread mode for using XA.

`true`: Multi-thread mode

`false`: Single-thread mode

(e) Exception that occurs

None.

17.11.19 setCommit_Behavior

(a) Function

Sets whether or not the following classes are to be valid after commit execution when HiRDB commits:

- `ResultSet` class
- `Statement` class, `PreparedStatement` class, and `CallableStatement` class

(b) Format

```
public void setCommit_Behavior (String type)
```

(c) Argument

`String type`

Sets whether or not the objects of the `Statement` class, `PreparedStatement` class, `CallableStatement` class, and `ResultSet` class remain valid even after a transaction terminates.

Specification value	ResultSet class	Statement class, PreparedStatement class, CallableStatement class
`DELETE` (default value)	Invalid#1	Invalid#2
`CLOSE`	Invalid#1	Valid

Specification value	ResultSet class	Statement class, PreparedStatement class, CallableStatement class
PRESERVE	Valid[#3]	Valid[#3]

#1: The condition that invalidates objects of the ResultSet class after commit execution is that the get*XXX* method of the ResultSet class can be executed by executing the following methods of the ResultSet class:

- next method
- first method
- last method
- absolute method
- relative method

Correct execution of a method using objects of a ResultSet class that was invalidated is not guaranteed.

#2: Objects that are invalid after commit execution include the following:

- SQL statements precompiled by the Connection.prepareStatement method
- SQL statements precompiled by the Connection.prepareCall method
- ResultSet class objects acquired by the executeQuery method of the Statement class, PreparedStatement class. or CallableStatement class.

#3: If the version of the connected HiRDB is earlier than 07-01, using LOCK TABLE to lock the table is required.

(d) Return value

None.

(e) Functional detail

Sets whether or not the objects of the Statement class, PreparedStatement class, CallableStatement class, and ResultSet class remain valid even after the transaction terminates. If this method is not called, the default is DELETE.

Executing this method is equivalent to setting the COMMIT_BEHAVIOR property that is performed when a database is connected using DriverManager.

(f) Exception that occurs

When XADataSource is used for the connection, DELETE always results, regardless of the specified value. However, getCommit_Behavior returns the value specified in the type argument.

(g) Notes

For notes, see *Notes on COMMIT_BEHAVIOR* following *Table 17-3*.

17.11.20 getCommit_Behavior

(a) Function

Sets whether or not objects of the Statement class, PreparedStatement class, CallableStatement class, and ResultSet class are to be valid even after the transaction terminates.

(b) Format

```
public String getCommit_Behavior()
```

(c) Argument

None.

(d) Return value

String

Returns Delete if there is no setting of the type that determines whether or not objects of the Statement class, PreparedStatement class, CallableStatement class and ResultSet class remain valid even after the transaction ends.

(e) Functional detail

The information specified by the setCommit_Behavior method is returned.

(f) Exception that occurs

None.

17.11.21 setBlockUpdate

(a) Function

Sets whether or not multiple parameter sets are to be processed at one time during database updating using the ? parameter (INSERT, UPDATE, and DELETE).

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(b) Format

```
public void setBlockUpdate(boolean Mode)
```

(c) Argument

`boolean Mode`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.

(d) Return value

None.

(e) Exception that occurs

None.

(f) Notes

For details about how to process multiple lines of `?` parameters in batch mode, see *Table 17-3 Information to be specified for Properties info* and *17.3.2 Batch updating*.

This function can also be specified by the `HiRDB_for_Java_BLOCK_UPDATE` system property. If the `setBlockUpdate` method has been set, the `HiRDB_for_Java_BLOCK_UPDATE` system property setting is ignored.

17.11.22 getBlockUpdate

(a) Function

Acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, and `DELETE`).

(b) Format

```
public boolean getBlockUpdate()
```

(c) Argument

None.

(d) Return value

`boolean`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.

(e) Exception that occurs

None.

(f) Notes

None.

17.11.23 setLONGVARBINARY_Access

(a) Function

Specifies the database access method for `LONGVARBINARY` (column attribute is `BLOB` or `BINARY`).

(b) Format

```
public void setLONGVARBINARY_Access(String Mode)
```

(c) Argument

`String Mode`

Specifies the database access method for `LONGVARBINARY` (column attribute is `BLOB` or `BINARY`). When this argument is omitted, `"REAL"` is assumed.

`"REAL"`

Accesses real data.

`"LOCATOR"`

Uses HiRDB's locator facility to access data.

Other:

Assumes that `"REAL"` has been specified.

(d) Return value

None.

(e) Exception that occurs

None.

17.11.24 getLONGVARBINARY_Access

(a) Function

Acquires the database access method for `LONGVARBINARY` (column attribute is `BLOB` or `BINARY`).

(b) Format

```
public String getLONGVARBINARY_Access()
```

(c) Argument

None.

(d) Return value

`String`

Indicates the information set as the database access method for `LONGVARBINARY` (column attribute is `BLOB` or `BINARY`). When no information has been set, `"REAL"` is assumed.

`"REAL"`

Accesses real data.

`"LOCATOR"`

Uses HiRDB's locator facility to access data.

(e) Functional detail

Returns the information specified by the `setLONGVARBINARY_Access` method.

(f) Exception that occurs

None.

17.11.25 setSQLInNum

(a) Function

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLInNum(int inNum)
```

(c) Argument

`int inNum`:

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed. The permitted value range is from `1` to `30,000` (default is `64`).

(d) Return value

None.

(e) Functional detail

Specifies the maximum number of input or input/output `?` parameters to be acquired during SQL preprocessing.

If the actual number of `?` parameters is greater than this property value, this method acquires information about the input or input/output `?` parameters after SQL preprocessing.

The value specified in this method is used as the value of `HiRDB_for_Java_SQL_IN_NUM` property during database connection.

(f) Exception that occurs

If the specified argument value falls beyond the permitted range, the method throws an `SQLException`.

(g) Notes

- This function can also be specified by the `HiRDB_for_Java_SQL_IN_NUM` system property. If the `setSQLInNum` method has been set, the `HiRDB_for_Java_SQL_IN_NUM` system property setting is ignored.
- If you do not execute any SQL statement that uses input or input/output `?` parameters, we recommend that you specify a value of `1`.

17.11.26 getSQLInNum

(a) Function

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLInNum`.

(b) Format

```
public int getSQLInNum()
```

(c) Argument

None.

(d) Return value

`int`

This is the maximum number of input or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLInNum`. If no value has been set, the method returns the default value (`64`).

(e) Exception that occurs

None.

17.11.27 setSQLOutNum

(a) Function

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLOutNum(int outNum)
```

(c) Argument

`int outNum`

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed. The permitted value range is from `1` to `30,000` (default is `64`).

(d) Return value

None.

(e) Functional detail

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed.

This specification is used as the number of output items that are to be acquired during SQL preprocessing.

If the number of actual output items is greater than the value of this property, the method acquires information about the output items after SQL preprocessing.

The value specified in this method is used as the value of the

HiRDB_for_Java_SQL_OUT_NUM property during database connection.

(f) Exception that occurs

If the specified argument value falls beyond the permitted range, the method throws an SQLException.

(g) Notes

- This function can also be specified by the HiRDB_for_Java_SQL_OUT_NUM system property. If the setSQLOutNum method has been set, the HiRDB_for_Java_SQL_OUT_NUM system property setting is ignored.
- If there is no search item, output ? parameter, or input/output ? parameter, we recommend that you specify a value of 1.

17.11.28 getSQLOutNum

(a) Function

Acquires the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed that has been set by setSQLOutNum.

(b) Format

```
public int getSQLOutNum()
```

(c) Argument

None.

(d) Return value

int

This is the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed that has been set by setSQLOutNum. If this value has not been set, the method returns the default value (64).

(e) Exception that occurs

None.

17.11.29 setSQLWarningLevel

(a) Function

Specifies the warning retention level that occurred during execution of SQL statements. The value specified in this method is used as the value of the HiRDB_for_Java_SQLWARNING_LEVEL property during database connection.

(b) Format

```
public void setSQLWarningLevel (String warningLevel)
```

(c) Argument

`String` `warningLevel`

Specifies the retention level of warning information that has been issued during execution of SQL statements. The permitted warning retention levels are listed below. For details about the relationship between the specified value and the retained warning, see *17.2.9 SQLWarning class*.

- `IGNORE`
- `SQLWARN` (default)
- `ALLWARN`

The value specified in the argument of this method is not case sensitive.

(d) Return value

None.

(e) Exception that occurs

If the specified argument value is invalid, the method throws an `SQLException`.

17.11.30 getSQLWarningLevel

(a) Function

Acquires the warning retention level specified in `setSQLWarningLevel`.

(b) Format

```
public String getSQLWarningLevel ()
```

(c) Argument

None.

(d) Return value

`String`

Returns the warning retention level set by `setSQLWarningLevel` (`IGNORE`, `SQLWARN`, or `ALLWARN`). If no warning retention level is specified, the method returns the default value (`SQLWARN`). For details about the relationship between the returned value and the retained warning, see *17.2.9 SQLWarning class*.

(e) Exception that occurs

None.

17.11.31 setClear_Env

(a) Function

Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection. The value specified in this method is equivalent to the `HiRDB_for_Java_CLEAR_ENV` property setting that is specified during database connection.

(b) Format

```
public void setClear_Env(boolean Mode)
```

(c) Argument

`boolean Mode`

Specifies whether or not the HiRDB client environment definition is to be ignored.

`true`: Ignores.

`false`: Does not ignore.

(d) Return value

None.

(e) Exception that occurs

None.

(f) Notes

For details, see *HiRDB_for_Java_CLEAR_ENV* in *Table 17-3 Information to be specified for Properties info*.

17.11.32 getClear_Env

(a) Function

Acquires the environment variable invalidation setting specified by `setClear_Env`.

(b) Format

```
public boolean getClear_Env()
```

(c) Argument

None.

(d) Return value

`String`

Returns the environment variable invalidation setting specified by `setClear_Env`. If no environment variable invalidation setting is specified, the method returns the default value (`false`).

`true`

Ignores the HiRDB client environment definition set as OS environment variables during database connection.

`false`

Does not ignore the HiRDB client environment definition set as OS environment variables during database connection.

(e) Exception that occurs

None.

17.12 Data types and character codes

17.12.1 Data types

JDBC's SQL data types and the SQL data types connected via a HiRDB client library do not match perfectly. The JDBC driver maps JDBC's SQL data types and HiRDB's SQL data types. If an unmappable SQL data type is used for data access, the JDBC driver throws an `SQLException`.

The SQL data types are mapped with the `get`*XXX* and `set`*XXX* methods in the `ResultSet`, `PreparedStatement`, and `CallableStatement` classes. For the SQL data types and the `get`*XXX* and `set`*XXX* method mapping rules, see the documentation for the JDBC1.0 standard.

The following table shows the correspondence of SQL data types between HiRDB and JDBC.

Table 17-19: Correspondence of SQL data types between HiRDB and JDBC

HiRDB's SQL data type	JDBC's SQL data type
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT, DOUBLE PRECISION	FLOAT
SMALLFLT, REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	CHAR
NVARCHAR	VARCHAR
MCHAR	CHAR
MVARCHAR	VARCHAR
DATE	DATE
TIME	TIME
BLOB	LONGVARBINARY
TIMESTAMP	TIMESTAMP

HiRDB's SQL data type	JDBC's SQL data type
BINARY#	LONGVARBINARY

#: Data is handled in the same way as BLOB.

17.12.2 Character code conversion facility

In a Java program, Unicode is used for the character codes. Therefore, the JDBC driver performs mutual character code conversion between the HiRDB character data and the Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the Java Virtual Machine. At this time, ENCODELANG of Properties info specifies the character set names specified by the JDBC driver for the encoder and decoder that are provided by the Java Virtual Machine.

Tables 17-20 and *17-21* show the correspondences between the HiRDB character codes and the Java character sets.

Table 17-20: Correspondence between HiRDB character codes and Java character sets (UNIX)

HiRDB character codes	Character set	Remarks
sjis (Shift JIS kanji)	"SJIS"	Double-byte characters include external characters.
ujis (EUC Japanese kanji)	"EUC_JP" (Japanese EUC)	Double-byte characters do not include external characters#
chinese (EUC Chinese kanji)	"EUC_CN" (Simplified Chinese)	Double-byte characters do not include external characters#
lang-c (8-bit codes)	"ISO-8859-1" (ISO Latin-1)	Can be used with US ASCII and 8-bit codes.
UTF-8	UTF-8	None
chinese-gb18030 (Chinese kanji codes (GB18030))	GB18030	None

Note

If ENCODELANG of Properties info is set using the following methods, this setting takes precedence for encoding.

- Set using Properties info passed as the argument of the DriverManager.getConnection method
- Set using the JdbhDataSource.setEncodLang method,

`rce.setEncodLang` method, or `JdbhXADataSource` method

For details about operation when `ENCODELANG` is not set using the above methods or when `OFF` is set, see *17.11.5 setEncodeLang*.

#: You cannot use external character codes assigned to EUC code set 3 (character codes expressed by three bytes in the range of $(8F)_{16}$ to $(XXXX)_{16}$.

Table 17-21: Correspondence between HiRDB character codes and Java Character sets (Windows)

HiRDB character codes	Character set	Remarks
`sjis` (Shift JIS kanji)	MS932 when the Java Virtual Machine standard encoding is MS932; otherwise, it is SJIS.	Double-byte characters include external characters.
UTF-8	UTF-8	None

Note

If `ENCODELANG` of `Properties info` is set using the following methods, this setting takes precedence for encoding:

- Set using `Properties info` passed as the argument of the `DriverManager.getConnection` method
- Set using the `JdbhDataSource.setEncodLang` method, `JdbhDataSource.setEncodLang` method, or `JdbhXADataSource` method.

For details about operation when `ENCODELANG` is not set using the above methods or when `OFF` is set, see *17.11.5 setEncodeLang*.

17.13 Classes and methods with limitations

This section explains the classes defined in the JDBC1.0 standard.

The JDBC driver does not support the following classes that are defined in the JDBC2.0 basic standard:

- `Clob` class
- `Struct` class
- `Ref` class
- `SQLData` class
- `SQLInput` class
- `SQLOutput` class

17.13.1 Driver class

There is no limitation to this class.

17.13.2 Connection class

Table 17-22 lists limitations to the methods in the `Connection` class that are defined in the JDBC1.0 standard, while *Table 17-23* lists limitations to the methods added in the JDBC2.0 basic standard.

Table 17-22: Limitations to the methods in the Connection class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
`setReadOnly`	Not usable.
`isReadOnly`	Unconditionally returns `false`.
`setCatalog`	Not usable.
`getCatalog`	Returns `null` unconditionally.
`setTransactionIsolation`	Not usable.
`getTransactionIsolation`	Returns `TRANSACTION_REPEATABLE_READ` unconditionally.

Table 17-23: Limitations to the methods in the Connection class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
`createStatement`	A result set reflecting updating results is not usable. Therefore, if `TYPE_SCROLL_SENSITIVE` is specified for the result set type, the method changes it to `TYPE_SCROLL_INSENSITIVE` and sets an `SQLWarning`.
`prepareStatement`	
`prepareCall`	
`getTypeMap`	Unconditionally throws `SQLException` because a user-defined type is not usable.
`setTypeMap`	

17.13.3 Statement class

Table 17-24 lists limitations to the methods in the `Connection` class that are defined in the JDBC1.0 standard, while *Table 17-25* lists limitations to the methods added in the JDBC2.0 basic standard.

Table 17-24: Limitations to the methods in the Statement class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
`setCursorName`	Not usable (because positioned updating or deletion is not available).
`getMaxFieldSize`	Returns the value specified with `setMaxFieldSize`.
`getMoreResults`	Unconditionally returns false.
`setMaxRows`	Not usable.
`setQueryTimeout`	

Table 17-25: Limitations to the methods in the Statement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
`setFetchDirection`	Throws `SQLException` if anything other than `FETCH_FORWARD` is specified.
`getFetchSize`	Returns the value specified with the `setFetchSize` method.

17.13.4 PreparedStatement class

The following table lists limitations to the methods in the PreparedStatement class that are added in the JDBC2.0 basic standard.

Table 17-26: Limitations to the methods in the PreparedStatement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
setBlob	For the JDBC driver, the method treats the JDBC SQL type as LONGVARBINARY.
setClob	Unconditionally throws SQLException because the SQL CLOB type is not available.
setRef	Unconditionally throws SQLException because the SQL structured type is not available.
setNull	If the complete name of an SQL user-defined type is specified, the method unconditionally throws SQLException because the SQL structured type or the SQL array type is not available.
setObject	Ignores the specified scale and obtains the value of scale from the actual value specified.

17.13.5 CallableStatement class

The following table lists limitations to the methods in the CallableStatement class that are added in the JDBC2.0 basic standard.

Table 17-27: Limitations to the methods in the CallableStatement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
getObject	If Map is specified, the method throws SQLException because the Map specification is not available.
getBlob	For the JDBC driver, the method treats the JDBC SQL type as LONGVARBINARY.
getClob	Unconditionally throws SQLException because the SQL CLOB type is not available.
getRef	Unconditionally throws SQLException because the SQL structured type is not available.

17.13.6 ResultSet class

The following table lists limitations to the methods in the ResultSet class that are added in the JDBC2.0 basic standard.

Table 17-28: Limitations to the methods in the ResultSet class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
setFetchDirection	Throws SQLException if anything other than FETCH_FORWARD is specified.
rowUpdated	Unconditionally throws SQLException because an updatable result set is not available.
rowInserted	
rowDeleted	
updateNull	
updateBoolean	
updateByte	
updateShort	
updateInt	
updateLong	
updateFloat	
updateDouble	
updateBigDecimal	
updateString	
updateBytes	
updateDate	
updateTime	
updateTimestamp	
updateAsciiStream	
updateBinaryStream	
updateCharacterStream	

Method added in JDBC2.0 basic standard	Limitation
`updateObject`	
`insertRow`	
`updateRow`	
`deleteRow`	
`refreshRow`	
`cancelRowUpdates`	
`moveToInsertRow`	
`moveToCurrentRow`	
`getObject`	If `Map` is specified, the method throws `SQLException` because the Map specification is not available.
`getBlob`	For the JDBC driver, the method treats the JDBC SQL type as `LONGVARBINARY`.
`getClob`	Unconditionally throws `SQLException` because the SQL `CLOB` type is not available.
`getRef`	Unconditionally throws `SQLException` because the SQL structured type is not available.

17.13.7 ResultSetMetaData class

The table below lists limitations to the methods in the `ResutlSetMetaData` class that are defined in the JDBC1.0 standard. However, for details about the return value of each method of the `MetaData` class acquired from the result set generated by the `getResultSet` method of the `Array` class, see *Table 17-14*.

Table 17-29: Limitations to the methods in the ResultSetMetaData class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
`isAutoIncrement`	Unconditionally returns `false`.
`isCaseSensitive`	Unconditionally returns `true`.
`isCurrency`	Unconditionally returns `false`.
`getColumnLabel`	Returns a column name because the column label (column header) is not available.

Method defined in JDBC1.0 standard	Limitation
getSchemaName	Unconditionally returns null.
getTableName	
getCatalogName	
isReadOnly	Unconditionally returns false.
isWritable	
isDefinitelyWritable	

17.13.8 DatabaseMetaData class

Table 17-30 lists limitations to the returned contents of methods in the DatabaseMetaData class that are defined in the JDBC1.0 standard, while *Table 17-31* lists limitations to the returned contents of the methods added by the JDBC2.0 basic standard. Note that the value returned by each method is information related to the HiRDB server, whose version has to be the same as the JDBC driver being used.

Table 17-30: Limitations to the methods in the DatabaseMetaData class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation or return value
allProceduresAreCallable	Returns false.
allTablesAreSelectable	Returns false.
getURL	Returns the JDBC URL of the connected database.
getUserName	Returns the authorization identifier used when connecting to the database.
isReadOnly	Unconditionally returns false because the access mode cannot be changed.
nullsAreSortedHigh	Returns true.
nullsAreSortedLow	Returns false.
nullsAreSortedAtStart	Returns false.
nullsAreSortedAtEnd	Unconditionally returns false.
getDatabaseProductName	Returns HiRDB.
getDatabaseProductVersion	Returns null.
getDriverName	Returns HiRDB_for_JDBC.

Method defined in JDBC1.0 standard	Limitation or return value
getDriverVersion	Returns 08.02.0000.
getDriverMajorVersion	8
getDriverMinorVersion	2
usesLocalFiles	Unconditionally returns false.
usesLocalFilePerTable	Unconditionally returns false.
supportsMixedCaseIdentifiers	Unconditionally returns false.
storesUpperCaseIdentifiers	Returns true.
storesLowerCaseIdentifiers	Unconditionally returns false.
storesMixedCaseIdentifiers	Returns false.
supportsMixedCaseQuotedIdentifiers	Returns true.
storesUpperCaseQuotedIdentifiers	Returns false.
storesLowerCaseQuotedIdentifiers	Unconditionally returns false.
storesMixedCaseQuotedIdentifiers	Returns true.
getIdentifierQuoteString	Unconditionally returns a quotation mark.
getSQLKeywords	Returns a HiRDB-specific SQL keyword.
getNumericFunctions	Returns a list of mathematical functions.
getStringFunctions	Returns a list of character string functions.
getSystemFunctions	Returns a list of system functions.
getTimeDateFunctions	Returns a list of time and date functions.
getSearchStringEscape	Returns \.
getExtraNameCharacters	Returns a special character that can be used as an SQL identification name.
supportsAlterTableWithAddColumn	Returns true.
supportsAlterTableWithDropColumn	
supportsColumnAliasing	
nullPlusNonNullIsNull	
supportsConvert (no argument)	Returns true.

Method defined in JDBC1.0 standard	Limitation or return value
`supportsConvert` (with arguments)	Returns either `true` or `false` depending on the combination of data types specified in arguments.
`supportsTableCorrelationNames`	Returns `true`.
`supportsDifferentTableCorrelationNames`	
`supportsExpressionsInOrderBy`	Returns `false`.
`supportsOrderByUnrelated`	Returns `true`.
`supportsGroupBy`	
`supportsGroupByUnrelated`	
`supportsGroupByBeyondSelect`	
`supportsLikeEscapeClause`	
`supportsMultipleResultSets`	Unconditionally returns `true`.
`supportsMultipleTransactions`	
`supportsNonNullableColumns`	Returns `true`.
`supportsMinimumSQLGrammar`	Unconditionally returns `true`.
`supportsCoreSQLGrammar`	
`supportsExtendedSQLGrammar`	Returns `false`.
`supportsANSI92EntryLevelSQL`	Unconditionally returns `true`.
`supportsANSI92IntermediateSQL`	Unconditionally returns `false`.
`supportsANSI92FullSQL`	
`supportsIntegrityEnhancementFacility`	Returns `false`.
`supportsOuterJoins`	Returns `true`.
`supportsFullOuterJoins`	Returns `false`.
`supportsLimitedOuterJoins`	Returns `true`.
`getSchemaTerm`	Returns `schema`.
`getProcedureTerm`	Returns `procedure`.
`getCatalogTerm`	Returns `null`.
`isCatalogAtStart`	Returns `false`.

Method defined in JDBC1.0 standard	Limitation or return value
getCatalogSeparator	Returns null.
supportsSchemasInDataManipulation	Unconditionally returns true.
supportsSchemasInProcedureCalls	Returns true.
supportsSchemasInTableDefinitions	
supportsSchemasInIndexDefinitions	
supportsSchemasInPrivilegeDefinitions	
supportsCatalogsInDataManipulation	Returns false.
supportsCatalogsInProcedureCalls	
supportsCatalogsInTableDefinitions	
supportsCatalogsInIndexDefinitions	Unconditionally returns false.
supportsCatalogsInPrivilegeDefinitions	
supportsPositionedDelete	
supportsPositionedUpdate	
supportsSelectForUpdate	
supportsStoredProcedures	Returns true.
supportsSubqueriesInComparisons	
supportsSubqueriesInExists	
supportsSubqueriesInIns	
supportsSubqueriesInQuantifieds	
supportsCorrelatedSubqueries	
supportsUnion	
supportsUnionAll	
supportsOpenCursorsAcrossCommit	Returns true if any of the following values is PRESERVE: • Setting of COMMIT_BEHAVIOR in URL • Setting of COMMIT_BEHAVIOR in Properties info • Argument when the setCommit_Behavior method is executed

Method defined in JDBC1.0 standard	Limitation or return value
supportsOpenCursorsAcrossRollback	Unconditionally returns false.
supportsOpenStatementsAcrossCommit	Returns true if any of the following values is PRESERVE or CLOSE: • Setting of COMMIT_BEHAVIOR in URL • Setting of COMMIT_BEHAVIOR in Properties info • Argument when the setCommit_Behavior method is executed
supportsOpenStatementsAcrossRollback	Unconditionally returns false.
getMaxBinaryLiteralLength	Returns a value of 64000.
getMaxCharLiteralLength	Returns a value of 32000.
getMaxColumnNameLength	Returns a value of 30.
getMaxColumnsInGroupBy	Returns a value of 255.
getMaxColumnsInIndex	Returns a value of 16.
getMaxColumnsInOrderBy	Returns a value of 255.
getMaxColumnsInSelect	Returns a value of 30000.
getMaxColumnsInTable	
getMaxConnections	Returns a value of 0.
getMaxCursorNameLength	Returns a value of 30.
getMaxIndexLength	Returns a value of 4036.
getMaxSchemaNameLength	Returns a value of 8.
getMaxProcedureNameLength	Returns a value of 30.
getMaxCatalogNameLength	Returns a value of 0.
getMaxRowSize	
doesMaxRowSizeIncludeBlobs	Returns false.
getMaxStatementLength	Returns a value of 2000000.
getMaxStatements	Returns a value of 64.
getMaxTableNameLength	Returns a value of 30.
getMaxTablesInSelect	Returns a value of 64.

Method defined in JDBC1.0 standard	Limitation or return value
getMaxUserNameLength	Returns a value of 8.
getDefaultTransactionIsolation	Unconditionally returns TRANSACTION_REPEATABLE_READ.
supportsTransactions	Unconditionally returns true.
supportsTransactionIsolationLevel	Returns true when the given transaction isolation level is any of the following: • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ
SupportsDataDefinitionAndDataManipulationTransactions	Returns false.
supportsDataManipulationTransactionsOnly	Returns false.
dataDefinitionCausesTransactionCommit	Returns true.
dataDefinitionIgnoredInTransactions	Unconditionally returns false.
getProcedures	Returns information about the Java stored routines.
getProcedureColumns	Returns information about the parameters of the Java stored routines.
getTables	Returns information about tables. Only the table types returned by getTableTypes can be specified in the list of table types to be obtained (types).
getSchemas	Returns information about schemas.
getCatalogs	Always returns a 0 result.
getTableTypes	Returns information about table types. The following values are returned: "SYSTEM TABLE": System table "BASE TABLE": Base table "VIEW": View table "READ ONLY VIEW": Read-only view table "ALIAS": Another table
getColumns	Returns information about columns.
getColumnPrivileges	Returns information about column privileges.
getTablePrivileges	Returns information about table privileges.

Method defined in JDBC1.0 standard	Limitation or return value
getBestRowIdentifier	Always returns a 0 result.
getVersionColumns	
getPrimaryKeys	Returns information about primary key columns (always returns a 0 result).
getImportedKeys	Always returns a 0 result.
getExportedKeys	Returns information about external key columns that reference the primary key columns (always returns a 0 result).
getCrossReference	Returns information about the external key columns in the table with external keys that reference the primary key columns in the table with the primary key (always returns a 0 result).
getTypeInfo	Returns information about the standard SQL types supported for the database.
getIndexInfo	Returns information about indexes.

Table 17-31: Limitations to the methods in the DatabaseMetaData class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation or return value
supportsResultSetType	Returns true if the result set type is TYPE_FORWARD_ONLY or TYPE_SCROLL_INSENSITIVE.
SupportsResultSet Concurrency	Returns true if the result set type is TYPE_FORWARD_ONLY or TYPE_SCROLL_INSENSITIVE and the parallel processing type is CONCUR_READ_ONLY.
ownUpdatesAreVisible	Unconditionally returns false.
ownDeletesAreVisible	
ownInsertsAreVisible	
othersUpdatesAreVisible	
othersDeletesAreVisible	
othersInsertsAreVisible	
updatesAreDetected	
deletesAreDetected	

Method added in JDBC2.0 basic standard	Limitation or return value
insertsAreDetected	
supportsBatchUpdates	Unconditionally returns true.
getUDTs	Always returns a 0 result.
getConnection	Returns the Connection instance that is the DatabaseMetaData instance generation source.

17.13.9 Blob class

The following table lists limitations to the methods in the Blob class that are added in the JDBC2.0 basic standard.

Table 17-32: Limitations to the methods added by JDBC2.0 basic standards for Blob class

Method added by JDBC2.0 basic standard	Limitation
setBinaryStream	Cannot be used for JDBC1.4 methods. If used, the method unconditionally throws an SQLException.
setBytes	
truncate	

17.13.10 Array class

The following table lists limitations to the methods in the Array class that are added by the JDBC2.0 basic standard.

Table 17-33: Restrictions on the methods added by the JDBC2.0 basic specification for the Array class

Methods added in the JDBC2.0 basic specification	Restrictions
getArray	Because MAP cannot be used, the method throws an SQLException if MAP is specified for the argument.
getResultSet	

Chapter

18. Type4 JDBC Driver

This chapter explains the Type4 JDBC driver installation, environment setup, and JDBC functions. Note that the Type4 JDBC driver cannot be used in the Linux for AP8000 edition client.

Hereafter in this chapter, the Type4 JDBC driver is referred to as the *JDBC driver*.

18.1 Installation and environment setup
18.2 Database connection using the DriverManager class
18.3 Database connection using a DataSource object and JNDI
18.4 JDBC1.2 core API
18.5 JDBC2.1 Core API
18.6 JDBC2.0 Optional Package
18.7 Connection information setup and acquisition interface
18.8 Data types
18.9 Character conversion facility
18.10 Supported client environment definitions
18.11 Connection information priorities
18.12 Migration from a Type2 JDBC driver
18.13 Migration from DABroker for Java
18.14 JDBC interface method trace
18.15 Exception trace log
18.16 Example UAP that uses a JDBC driver
18.17 Estimating the memory requirements for using a JDBC driver

18.1 Installation and environment setup

18.1.1 Installation

The JDBC driver can be installed when you install HiRDB. After the driver is installed, the file configuration is as follows:

For UNIX

```
HiRDB/client/lib/pdjdbc2.jar
```

For Windows (server product)

```
HiRDB\client\utl\pdjdbc2.jar
```

For Windows (client product)

```
HiRDB\utl\pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

18.1.2 Environment setup

Before you use the JDBC driver to execute UAPs, you must specify the installed file in the OS's `CLASSPATH` environment variable. Also, before you compile a UAP you must set up the `CLASSPATH` environment variable in order to directly manipulate the classes provided by the JDBC driver, which is necessary for the methods provided by the JDBC driver that do not comply with the JDBC standards.

If you are using the JDBC driver from an application server, such as Cosminexus, the environment setup depends on the environment setup for the application server. Refer to the documentation for the particular application server, and check the specifications.

(1) UNIX environment

(a) Bourne shell

```
CLASSPATH=${CLASSPATH}:/HiRDB/client/lib/pdjdbc2.jar
export CLASSPATH
```

Note

The underlined portion indicates the HiRDB installation directory.

(b) C shell

```
setenv CLASSPATH ${CLASSPATH}:/HiRDB/client/lib/pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

(2) Windows environment (executing the program from the command prompt)

```
set CLASSPATH=%CLASSPATH%;C:\Program Files\HITACHI\HiRDB\client\utl\pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

18.1.3 Abbreviation of methods

- Methods that begin with `get` are referred to collectively as `get`*XXX* methods.
- Methods that begin with `set` are referred to collectively as `set`*XXX* methods.
- Methods that begin with `execute` are referred to collectively as `execute`*XXX* methods.
- This manual uses the notation `DataSource`-*type interface* to represent the following interfaces generically:
    - `DataSource`
    - `ConnectionPoolDataSource`
    - `XADataSource`

18.2 Database connection using the DriverManager class

The procedure for connecting from the `DriverManager` class to HiRDB and generating an instance of the `Connection class` is as follows:

1. Register the `Driver` class into the Java Virtual Machine.
2. Set the connection information in the arguments, and use the `getConnection` method of the `DriverManager` class to connect to HiRDB.

18.2.1 Registering the Driver class

The procedure for registering the JDBC driver into the Java Virtual Machine is described below.

The driver name that must be used to register the `Driver` class into the Java Virtual Machine is *package-name*.*class-name*. The package and class names of the JDBC driver are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `HiRDBDriver`

(1) Registering using the forName method of the Class class

Call the `forName` method of the `Class` class from within the application as follows:

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(2) Registering in the system properties

Set the following value in the `jdbc.drivers` system property of the Java Virtual Machine:

```
System.setProperty("jdbc.drivers", "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(3) Registering into the operation setup file of the Java Virtual machine (Applet)

Specify in the `[JAVA_HOME]\.hotjava\properties` file the information shown below (the value of `[JAVA_HOME]` depends on the Java execution environment). If you register multiple JDBC drivers, delimit them with colons (`:`).

```
jdbc.drivers="JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver"
```

18.2.2 Connecting to HiRDB with the getConnection method

The `getConnection` method of the `DriverManager` class is provided in the following three formats, each with its own set of arguments:

- `public static Connection getConnection(String url)`
- `public static Connection getConnection(String url, String user, String password)`
- `public static Connection getConnection(String url, Properties info)`

The arguments (`url`, `user`, `password`, and `info`) in these method formats specify connection information that is needed in order to connect to HiRDB.

When connection to HiRDB is established successfully, the JDBC driver returns a reference to a `Connection` class instance as the result of calling the method. However, the method throws an `SQLException` in the following cases:

- The required connection information is not specified in an argument.
- Specified connection information is invalid.
- Connection cannot be established (for example, because HiRDB has not been started at the connection destination).

The following table describes the information to be specified in the `getConnection` method arguments.

Table 18-1: Specification details of the getConnection method arguments

Argument	Specification details	External driver[#1]	Internal driver[#2]
`String url`	Specifies the URL. For details, see *(1) URL syntax*.	Y	Y
`String user`	Specifies the authorization identifier. If the null value is specified, the JDBC driver assumes that no authorization identifier has been specified. If the character string has a length of `0`, the method throws an `SQLException` and `user` is set to *aa....aaa*, which are characters embedded in the KFPJ20212-E message. For details about the specification priorities, see *18.11 Connection information priorities*.	Y	N

Argument	Specification details	External driver#1	Internal driver#2
`String password`	Specifies the password. For details about the specification priorities, see *18.11 Connection information priorities*. If the null value is specified, or if a character string of length 0 is specified, the JDBC driver assumes that no password was specified.	Y	N
`Properties info`	Specifies various connection information items. For details, see *(2) User properties*.	Y	Y

Legend:

Y: Specification takes effect

N: Specification is ignored

#1

JDBC driver used by Java applications

#2

JDBC driver used by Java stored procedures

(1) URL syntax

This section explains the URL syntax supported by the JDBC driver.

You must not specify any spaces within an item or between items in a URL. Note that the item names are case sensitive.

(a) URL syntax

```
jdbc:hitachi:hirdb[://[DBID=additional-connection-information]
                  [,DBHOST=database-host-name]
                  [,ENCODELANG=conversion-character-set]
                  [,HIRDB_CURSOR=cursor-operation-mode]
                  [,STATEMENT_COMMIT_BEHAVIOR=Statement-object-status-after-commit-execution]
                  [,JDBC_IF=whether-to-obtain-JDBC-interface-method-traces]
                  [,TRC_NO=trace-entries-count]
                  [,SQLWARNING_IGNORE=whether-to-return-warning-information]
                  [,LONGVARBINARY_ACCESS=method-of-accessing-BLOB-and-BINARY-types]
                  [,SQL_IN_NUM=?-parameters-count]
                  [,SQL_OUT_NUM=output-parameters-count]
                  [,SQLWARNING_LEVEL=warning-retention-level]
                  [,LONGVARBINARY_ACCESS_SIZE=LONGVARBINARY-data-access-size]
                  [,MAXBINARYSIZE=maximum-size-of-LONGVARBINARY-data]
                  [,LONGVARBINARY_TRUNCERROR=whether-to-throw-exceptions]
                  [,STATEMENT_CLOSE_BEHAVIOR=whether-to-ignore-preprocessing-results]
                  [,HiRDB_INI=directory-path-of-HiRDB.INI-file]
                  [,USER=user-name]
                  [,PASSWORD=password]
                  [,UAPNAME=application-name]
              [,BATCHEXCEPTION_BEHAVIOR=whether-to-set-JDBC-standard-compliant-update-count]
                  ]
```

(b) Explanation of URL items

`jdbc:hitachi:hirdb`

This item consists of the protocol name, subprotocol name, and subname. You must specify this item. This item is case sensitive.

`DBID=`*additional-connection-information*

Specifies the port number of the HiRDB server (corresponds to the `PDNAMEPORT` value in the client environment definitions). You can also specify a HiRDB environment variable group for this item.

If no port number is specified for the HiRDB server, the value specified by another method takes effect. For details about how to specify the HiRDB port number and specification priorities, see *18.11 Connection information priorities*.

If neither value is specified, the `getConnection` method throws an `SQLException` when it executes.

Note that the HiRDB port number specification is ignored for an internal driver.

Notes

You should note the following points about specifying an HiRDB environment variable group for the additional connection information:

- When you specify the name of the HiRDB environment variable group, specify `@HIRDBENVGRP=` followed by the absolute path name. If no value is specified after the equal sign, such as `@HIRDBENVGRP=,`, the JDBC driver assumes that no value is specified for this item.
- Note that an environment variable group name is case sensitive. Also, the environment variable group name depends on the OS.
- If the environment variable group name contains any single-byte space or single-byte `@` characters, you must enclose the name in single-byte double quotation marks (`"`). When an environment variable group name is enclosed in single-byte double quotation marks, the characters from the closing single-byte quotation mark to the next setting item or to the final character are ignored. Note that an environment variable group that includes a single-byte quotation mark or a single-byte comma cannot be specified.

  Below are examples of specifications that trigger an error:

```
@ΔHIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRPΔ=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=Δ/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.iniΔ
```

Note: Δ represents a single-byte space character.

`DBHOST=`*database-host-name*

Specifies the name of the HiRDB host.

If this specification is omitted, the value set by another method takes effect. For details about how to specify the HiRDB host name and specification priorities, see *18.11 Connection information priorities*.

If neither value is specified, the `getConnection` method throws an `SQLException` when it executes.

Note that this specification is ignored for an internal driver.

`ENCODELANG=`*conversion-character-set*

Specifies the conversion character set for the HiRDB character codes of the connection destination when the JDBC driver uses the `String` class to exchange data with HiRDB. Select a specifiable conversion character set from the encoding list shown under *Internationalization* in the *Java^{TM} 2 SDK, Standard Edition* documentation.

The following table lists the character codes of HiRDB and their corresponding conversion character sets.

Table 18-2: HiRDB character codes and corresponding conversion character sets

HiRDB character codes (character code set with pdntenv or pdsetup command)	Conversion character set to be specified
lang-c	ISO8859_1
sjis	SJIS or MS932#
ujis	EUC_JP
utf-8	UTF-8
chinese	EUC_CN
chinese-gb18030	GB18030

#

The specification of SJIS or MS932 depends on the handling of Windows special characters in the application.

When OFF is specified, the JDBC driver operates assuming that the conversion character set for the HiRDB character codes shown in *Table 18-2* was specified. If the HiRDB character code set is sjis, the conversion character set determined by the OS running the JDBC driver is as follows:

For UNIX: SJIS

For Windows: MS932

Note that the specification is case sensitive (except for OFF).

If a conversion character set that is not supported by the Java Virtual Machine is specified, the JDBC driver throws an SQLException during connection with the HiRDB server.

If this specification is omitted, the JDBC driver converts characters using the applicable conversion character set shown in *Table 18-2*. However, if one of the following is specified, the JDBC driver converts characters by using the default conversion character set of the Java Virtual Machine:

- Specification value for the UAP name (value specified by the UAPNAME property)
- Authorization identifier or password (value specified by the

getConnection method)

- Specification value for the client environment definition specified by EnvironmentVariables
- Specification value for an environment variable specified by the environment variable group name of the HiRDB client

HIRDB_CURSOR=*cursor-operation-mode*

Specifies whether objects of the ResultSet class are to be validated or invalidated after HiRDB executes commit processing.

TRUE: Validate objects of the ResultSet class even after commit processing.

FALSE: Invalidate objects of the ResultSet class after commit processing.

If this specification is omitted, FALSE is assumed.

If a value other than TRUE or FALSE is specified, the JDBC driver throws an SQLException.

If an invalidated ResultSet object executes an operation other than calling the close method, the JDBC driver throws an SQLException.

Note

For notes about specifying HIRDB_CURSOR, see *(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

STATEMENT_COMMIT_BEHAVIOR=*Statement-object-status-after-commit-execution*

Specifies whether objects of the Statement, PreparedStatement, and CallableStatement classes (referred to hereafter collectively as Statement) are to remain in effect even after HiRDB executes commit processing.

TRUE: Validate Statement objects even after HiRDB executes commit processing.

FALSE: Invalidate Statement objects after HiRDB executes commit processing.

The objects that become invalid after commit execution are SQL statements that were precompiled by the prepareStatement or prepareCall method of the Connection class, and ResultSet class objects obtained by the executeQuery method of Statement.

If this specification is omitted, TRUE is assumed.

Note

For notes about specifying STATEMENT_COMMIT_BEHAVIOR, see *(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

JDBC_IF=*whether-to-obtain-JDBC-interface-method-traces*

Specifies whether JDBC interface method traces are to be obtained.

`ON`: Obtain JDBC interface method traces.

`OFF`: Do not obtain JDBC interface method traces.

If this specification is omitted, `OFF` is assumed. If any other value is specified, the JDBC driver throws an `SQLException`.

For details about the specification, see *18.2.2(2)(d) JDBC_IF*.

TRC_NO=*trace-entries-count*

Specifies the number of JDBC interface method trace entries.

For details about the specification, see *18.2.2(2)(e) TRC_NO*.

SQLWARNING_IGNORE=*whether-to-retain-warning-information*

Specifies whether warnings returned from the database are to be retained by the `Connection` object.

`TRUE`: Do not retain warnings.

`FALSE`: Retain warnings.

If this specification is omitted, `FALSE` is assumed

If any other value is specified, the JDBC driver throws an `SQLException`.

For details about the specification, see *18.2.2(2)(g) SQLWARNING_IGNORE*.

LONGVARBINARY_ACCESS=*method-of-accessing-BLOB-and-BINARY-types*

Specifies the method of accessing a JDBC SQL-type `LONGVARBINARY` (`BLOB` and `BINARY` types in HiRDB) database.

`REAL`: Access the database using real data.

`LOCATOR`: Access the database using HiRDB's locator facility.

If any other value is specified, the JDBC driver throws an `SQLException`.

For details about the specification, see *18.2.2(2)(i) LONGVARBINARY_ACCESS*.

SQL_IN_NUM=*?-parameters-count*

Specifies the maximum number of input `?` parameters in the SQL statements to be executed.

For details about the specification, see *18.2.2(2)(j) HiRDB_for_Java_SQL_IN_NUM*.

SQL_OUT_NUM=*output-parameters-count*

Specifies the maximum number of output items for the SQL statements to be executed.

For details about the specification, see *18.2.2(2)(k) HiRDB_for_Java_SQL_OUT_NUM*.

SQLWARNING_LEVEL=*warning-retention-level*

Specifies the retention level for warning information that is issued during execution of SQL statements. For details about the warning information retention level, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*.

`IGNORE`: Retain warning information at the `IGNORE` level.

`SQLWARN`: Retain warning information at the `SQLWARN` level.

`ALLWARN`: Retain warning information at the `ALLWARN` level.

If this specification is omitted, `SQLWARN` is assumed.

If the specified value is invalid, the JDBC driver throws an `SQLException`.

LONGVARBINARY_ACCESS_SIZE=*LONGVARBINARY-data-access-size*

Specifies the length (in kilobytes) of JDBC SQL-type `LONGVARBINARY` data that can be requested at one time at the HiRDB server.

For details about the specification, see *18.2.2(2)(o) HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE*.

MAXBINARYSIZE=*maximum-size-of-LONGVARBINARY-data*

Specifies the maximum data size (in bytes) during acquisition of JDBC SQL-type `LONGVARBINARY` data.

For details about the specification, see *18.2.2(2)(p) HiRDB_for_Java_MAXBINARYSIZE*.

LONGVARBINARY_TRUNCERROR=*whether-to-throw-exceptions*

Specifies whether exceptions are to be thrown when truncation occurs during acquisition of JDBC SQL-type `LONGVARBINARY` data.

`TRUE`: Throw exceptions.

`FALSE`: Do not throw exceptions.

For details about the specification, see *18.2.2(2)(q) HiRDB_for_Java_LONGVARBINARY_TRUNCERROR*.

STATEMENT_CLOSE_BEHAVIOR=*whether-to-ignore-preprocessing-results*

Specifies whether preprocessing results are to be ignored during execution of the `close` method of `Statement` (the `Statement`, `PreparedStatement`, and `CallableStatement` classes).

TRUE: Ignore preprocessing results.

FALSE: Do not ignore preprocessing results.

If any other value is specified, the JDBC driver throws an SQLException.

For details about the specification, see *18.2.2(2)(r) HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR*.

HiRDB_INI=*directory-path-of-HiRDB.INI-file*

Specifies the absolute path of the directory that contains the HiRDB.INI file. This item is applicable when the HiRDB client environment variables specified in the HiRDB.INI file are to be in effect, in which case the HiRDB client environment variables in the HiRDB.ini file located in the directory specified here will take effect. If the specified directory does not contain a HiRDB.ini file and this is an interval driver, this specification is ignored.

If this specification is omitted, the contents of the HiRDB.ini file are ignored.

USER=*user-name*

Specifies the user name.

For details about the specification, see *18.2.2(2)(a) user*.

PASSWORD=*password*

Specifies the password.

For details about the specification, see *18.2.2(2)(b) password*.

UAPNAME=*application-name*

Specifies identification information (UAP identifier) for the UAP that is to access the HiRDB server.

For details about the specification, see *18.2.2(2)(c) UAPNAME*.

BATCHEXCEPTION_BEHAVIOR=*whether-to-set-JDBC-standard-compliant-update-count*

Specifies whether an update count that is compliant with the JDBC standard is to be set for the return value of the getUpdateCounts method of java.sql.BatchUpdateException.

For details about the specification, see *18.2.2(2)(v) HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR*.

(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR

The notes that follow apply to specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR.

When TRUE is specified in HIRDB_CURSOR or STATEMENT_COMMIT_BEHAVIOR

- If the PDDDLDEAPRPEXE and PDDDLDEAPRP client environment definitions are both set to NO and another user executes a definition SQL statement for a schema resource (table or index) to be accessed by a SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement, the definition SQL statement remains on lock-release wait status until the connection that was accessing the schema resource is disconnected or until the Statement object that was accessing the schema resource is closed and committed (applicable when TRUE is specified in STATEMENT_CLOSE_BEHAVIOR).
- If YES is set in either the PDDDLDEAPRPEXE or the PDDDLDEAPRP client environment definition and another user executes a definition SQL statement for a schema resource (table or index) to be accessed by a SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement, the preprocessing results of the SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement are ignored. If an SQL statement with an invalid preprocessing result is executed, an SQLException occurs (the value obtained by the getErrorCode method is -1542).
- When TRUE is specified for HIRDB_CURSOR or STATEMENT_COMMIT_BEHAVIOR,#1 the only precompiled SQL statements#2 that are valid after execution of commit processing#3 are the SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, and CALL statements.

#1

This also applies to either of the following specifications:

- TRUE is set for the following items in the properties specified by the getConnection method:
  - HIRDB_CURSOR
  - HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
- true is specified by the following methods of the PrdbDataSource, PrdbConnectionPoolDataSource, or PrdbXADataSource class:
  - setHiRDBCursorMode
  - setStatementCommitBehavior

#2

You precompile an SQL statement by executing a prepareStatement or prepareCall method of the Connection class.

#3

In addition to explicit commit processing by the `commit` method, the following cases also apply:

- Implicit commit processing by the `AUTO` commit mode
- Execution of a definition SQL statement
- Execution of the `PURGE TABLE` statement
- Implicit rollback processing by the `rollback` method
- Implicit rollback processing because of an SQL execution error

For SQL statements other than `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `PURGE TABLE`, and `CALL`, precompiled SQL statements become invalid during commit processing.

An error will occur if a `PreparedStatement` or `CallableStatement` class object that stores a precompiled SQL statement that has become invalid is used to execute the SQL statement. The following shows an example that results in an error:

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();  //Does not trigger an error.
con.commit();
pstmt2.execute();  //Triggers an error.
pstmt1.close();
pstmt2.close();
```

Because the SQL statements to be executed in this example are `LOCK` statements, after commit processing is executed, `PreparedStatement` becomes invalid and an error occurs, even if `TRUE` is specified for `STATEMENT_COMMIT_BEHAVIOR`.

- When `TRUE` is specified for `HIRDB_CURSOR`, the JDBC driver uses the holdable cursor facility of HiRDB.
- When `TRUE` is specified for `HIRDB_CURSOR`, the cursor operation mode for the result set that is returned by a procedure by using the results-set return facility is determined by the procedure's definition. Note that because the JDBC driver cannot detect a procedure's cursor operation mode, it assumes the operation mode specified for `HIRDB_CURSOR`. Therefore, if `TRUE` is specified for `HIRDB_CURSOR`, but the cursor operation mode for the result set returned by the procedure is not a holdable cursor, a database access performed after commit processing will result in `SQLException`. The following shows the status of the `ResultSet` object after commit processing:

Result set returned by procedure	HIRDB_CURSOR setting	
	TRUE	**FALSE**
Holdable cursor	Usable after commit processing.	If an operation performed after commit processing involves a database access, an error will result.#
Non-holdable cursor	If an operation performed after commit processing involves a database access, an error will result.	If an operation performed after commit processing involves a database access, an error will result.

#

Because the cursor is not closed during commit processing, it remains open until the ResultSet object is closed or the CallableStatement or Connection object that created the ResultSet object closes.

Combinations of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR specification values

The following table shows specification values for the combinations of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR whether ResultSet and Statement objects become valid after commit processing is executed.

Table 18-3: Status of ResultSet objects and Statement objects after commit execution

STATEMENT_COMMIT_BEHAVIOR specification value	HIRDB_CURSOR specification value	
	TRUE	**FALSE**
TRUE	ResultSet object: Valid Statement object: Valid	ResultSet object: Invalid Statement object: Valid
FALSE		ResultSet object: Invalid Statement object: Invalid

The following table shows the return value of the DatabaseMetaData method depending on the HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR specification values.

Table 18-4: Return values of the DatabaseMetaData method

STATEMENT_COMMIT_BEHAVIOR specification value	HIRDB_CURSOR specification value	
	TRUE	FALSE
TRUE	supportsOpenStatementsAcrossCommit: true supportsOpenCursorsAcrossCommit: true	supportsOpenStatementsAcrossCommit: true supportsOpenCursorsAcrossCommit: false
FALSE		supportsOpenStatementsAcrossCommit: false supportsOpenCursorsAcrossCommit: false

Examples of JDBC driver operation during COMMIT execution

The operation of the JDBC driver during `COMMIT` execution depends on the specification values of `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR`.

Specification examples

```
[A]
pstmt1=con.prepareStatement("select c1 from tb1");  [1]
[B]
rs1=pstmt1.executeQuery();                           [2]
[C]
rs1.next()                                           [3]
[D]
v1=rs1.getInt(1)                                     [4]
[E]
rs1.next()                                           [5]
[F]
v1=rs1.getInt(1)                                     [6]
[G]
rs1.close()                                          [7]
```

Driver operation at COMMIT execution

COMMIT timing	H=T and S=T#1	H=F and S=T#2	H=F and S=F#3
[A]	*[1]*-*[7]*: Operates normally.		

COMMIT timing	H=T and S=T#1	H=F and S=T#2	H=F and S=F#3
[B]	*[1]-[7]*: Operates normally.		*[1]*, *[2]*, and *[7]*: Operates normally. *[3]-[6]*: Throws an SQLException.
[C]	*[1]-[7]*: Operates normally.	*[1]*, *[2]*, and *[7]*: Operates normally. *[3]-[6]*: Throws an SQLException.	
[D]	*[1]-[7]*: Operates normally.	*[1]-[3]* and *[7]*: Operates normally. *[4]-[6]*: Throws an SQLException.	
[E]	*[1]-[7]*: Operates normally.	*[1]-[4]* and *[7]*: Operates normally. *[5]* and *[6]*: Throws an SQLException.	
[F]	*[1]-[7]*: Operates normally.	*[1]-[5]* and *[7]*: Operates normally. *[6]*: Throws an SQLException.	
[G]	*[1]-[7]*: Operates normally.		

#1: This represents the case when TRUE is specified for both HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR.

#2: This represents the case when FALSE is specified for HIRDB_CURSOR and TRUE is specified for STATEMENT_COMMIT_BEHAVIOR.

#3: This represents the case when FALSE is specified for both HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR.

Other notes

For notes about the PDDDLDEAPRP client environment definition, see *6.6.4 Environment definition information*.

For details about the rules for the DECLARE CURSOR holdable cursor, see the manual *HiRDB Version 9 SQL Reference*.

(2) User properties

The table below shows the properties that you can specify in the getConnection method of the DriverManager class. If the null value is specified for a property, the JDBC driver assumes that specification was omitted.

Table 18-5: Properties that can be specified in the getConnection method

Item	Property	Specified information
(a)	user	Authorization identifier
(b)	password	Password

Item	Property	Specified information
(c)	UAPNAME	UAP identifier
(d)	JDBC_IF	Whether or not a JDBC interface method trace is to be obtained
(e)	TRC_NO	Number of entries in the JDBC interface method trace
(f)	ENCODELANG	Conversion character set for the HiRDB character codes of the connection destination
(g)	SQLWARNING_IGNORE	Whether a warning returned from the database is to be retained by the Connection object
(h)	HIRDB_CURSOR	Cursor operation mode
(i)	LONGVARBINARY_ACCESS	Method of accessing a JDBC SQL-type LONGVARBINARY (BLOB and BINARY types, which are HiRDB data types) database
(j)	HiRDB_for_Java_SQL_IN_NUM	Maximum number of input ? parameters in the SQL statements to be executed
(k)	HiRDB_for_Java_SQL_OUT_NUM	Maximum number of output items for the SQL statements to be executed
(l)	HiRDB_for_Java_SQLWARNING_LEVEL	Retention level for warning information that is issued during execution of SQL statements
(m)	HiRDB_for_Java_ENV_VARIABLES	HiRDB client environment variables
(n)	HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	Statement object status after commit execution
(o)	HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE	Length of JDBC SQL-type LONGVARBINARY data to be requested at one time to the HiRDB server
(p)	HiRDB_for_Java_MAXBINARYSIZE	Maximum data size during acquisition of JDBC SQL-type LONGVARBINARY data
(q)	HiRDB_for_Java_LONGVARBINARY_TRUNCERROR	Whether or not an exception is to be thrown if truncation occurs during acquisition of JDBC SQL-type LONGVARBINARY data
(r)	HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	Whether preprocessing results are to be ignored during execution of the close method of the Statement (Statement, PreparedStatement, and CallableStatement classes)
(s)	HiRDB_for_Java_DBID	Additional connection information
(t)	HiRDB_for_Java_DBHOST	Host name

Item	Property	Specified information
(u)	HiRDB_for_Java_HiRDB_INI	Directory path of the HiRDB.INI file
(v)	HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR	Whether an update count that is compliant with the JDBC standard is to be set

(a) user

Specifies the authorization identifier.

If the null value is specified, the JDBC driver assumes that no authorization identifier was specified. If the character string has a length of 0, the JDBC driver throws an SQLException.

If this specification is omitted, either the PDNAMEPORT HiRDB client environment definition specified by HiRDB_for_Java_ENV_VARIABLES in the Properties argument of the getConnection method, or the PDUSER value in the HiRDB environment variable group specified for DBID in the URL takes effect. For details about the specification priorities, see *18.11 Connection information priorities*.

If neither value is specified, the JDBC driver throws an SQLException when the getConnection method is executed.

Note that this property specification is ignored for an internal driver.

(b) password

Specifies the password.

If the specification value is the null or has a length of 0, the JDBC driver assumes that no password was specified.

For details about when this specification is omitted, see *18.11 Connection information priorities*.

Note that this property specification is ignored for an internal driver.

(c) UAPNAME

Specifies UAP identification information (UAP identifier) for accessing the HiRDB server.

In the following cases, the JDBC driver assumes that no authorization identifier was specified:

- The null value is specified.
- A character string with a length of 0 or a character string of only single-byte space characters is specified.

For details about character strings that can be specified, see the description of the PDCLTAPNAME client environment definition in *6.6.4 Environment definition* *information*.

For details about when this specification is omitted, see *18.11 Connection information priorities*.

Note that this property specification is ignored for an internal driver.

Note

> The UAP specified by this property is encoded in the conversion character set specified by ENCODELANG, and the first 30 bytes of the encoded UAP identifier are transferred to the HiRDB server (if the UAP identifier consists of more than 30 bytes, it is truncated to the first 30 bytes). Therefore, the UAP identifier that the HiRDB server can obtain is up to the first 30 bytes after the identifier has been encoded.

(d) JDBC_IF

Specifies whether or not a JDBC interface method trace is to be obtained.

ON: Obtain a JDBC interface method trace.

OFF: Do not obtain a JDBC interface method trace.

If this specification is omitted, OFF is assumed.

If any other value is specified, the JDBC driver throws an SQLException.

This property is ignored in the following cases:

- A valid log writer is not specified in the setLogWriter method.
- This is an internal driver.

For details about the JDBC interface method trace, see *18.14 JDBC interface method trace*.

(e) TRC_NO

~<unsigned integer>((10-1000))<<500>>

Specifies the number of entries in the JDBC interface method trace.

The specification of this property is enabled when both of the following conditions are satisfied:

- The setLogWriter method has set valid log data.
- ON is specified for JDBC_IF.

Note that this property specification is ignored for an internal driver.

If the specification of this property is enabled but the specification value is invalid, the JDBC driver throws an SQLException.

For details about a JDBC interface method trace, see *18.14 JDBC interface method* *trace*.

(f) ENCODELANG

Specifies the conversion character set for the HiRDB character codes of the connection destination when the JDBC driver uses the `String` class to exchange data with HiRDB.

Select a specifiable conversion character set from the encoding list shown under *Internationalization* in the *Java^{TM} 2 SDK, Standard Edition* documentation.

For the character codes of HiRDB and their corresponding conversion character sets, see *Table 18-2 HiRDB character codes and corresponding conversion character sets*.

If `OFF` is specified, the JDBC driver operates assuming that the applicable conversion character set for the HiRDB character codes, as shown in *Table 18-2 HiRDB character codes and corresponding conversion character sets*, was specified. If the HiRDB character codes are `sjis`, the conversion characters determined by the OS running the JDBC driver are as follows:

For UNIX: `SJIS`

For Windows: `MS932`

Note that the specification is case sensitive (except for `OFF`).

If a conversion character set that the Java Virtual Machine does not support is specified, the JDBC driver throws an `SQLException` during connection with the HiRDB server.

If this specification is omitted, the JDBC driver converts characters using the conversion character set specified by `ENCODELANG` in the URL.

(g) SQLWARNING_IGNORE

Specifies whether warnings returned from the database are to be retained by the `Connection` object.

`TRUE`: Do not retain warnings.

`FALSE`: Retain warnings.

If this specification is omitted, the value specified for `SQLWARNING_IGNORE` in the URL takes effect. If the specified value is neither `TRUE` nor `FALSE`, the JDBC driver throws an `SQLException`.

Retention of warnings for the `Connection` object is determined by the value of `HiRDB_for_Java_SQLWARNING_LEVEL`. For details about the warning retention levels, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*. Note that the value of this property is not case sensitive.

(h) HIRDB_CURSOR

Specifies whether objects of the ResultSet class are to be validated or invalidated after HiRDB executes commit processing.

TRUE: Validate objects of the ResultSet class even after commit processing.

FALSE: Invalidate objects of the ResultSet class after commit processing.

If this specification is omitted, the value specified by HIRDB_CURSOR in the URL becomes valid. If a value other than TRUE or FALSE is specified, the JDBC driver throws an SQLException.

If an invalidated ResultSet object executes an operation other than calling the close method, the JDBC driver throws an SQLException.

Note

For notes about specifying this property, see *(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(i) LONGVARBINARY_ACCESS

Specifies the method of accessing a JDBC SQL-type LONGVARBINARY (BLOB and BINARY types, which are HiRDB data types) database.

REAL: Access the database using real data.

LOCATOR: Access the database using the locator facility of HiRDB.

Note that real data is assumed for an access to a BINARY column with a definition length of 1,024 bytes or less.

If this specification is omitted, REAL is assumed.

If any other value is specified, the JDBC driver throws an SQLException.

Note

The following notes apply to specification of LONGVARBINARY_ACCESS:

When LONGVARBINARY_ACCESS is specified together with HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

The following table describes the differences in how the JDBC driver gets BLOB and BINARY data (HiRDB data types) based on the specifications of HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE and LONGVARBINARY_ACCESS.

Table 18-6: Differences in how the HiRDB driver gets BLOB and BINARY data (HiRDB data types)

Execution method		LONGVARBINARY_ACCESS specification value#	
		REAL	LOCATOR
CallableStatment.execute ResultSet.next		Acquires all the BLOB or BINARY data from the connected database.	Acquires the locator that indicates the BLOB or BINARY data in the connected database, instead of the actual BLOB or BINARY data.
CallableStatement.getBytes CallableStatement.getString CallableStatement.getObject ResultSet.getBytes ResultSet.getString ResultSet.getObject		Uses the BLOB or BINARY data obtained by ResultSet.next.	Acquires from the connected database all BLOB or BINARY data in segments of ACCESSSIZE x 1024 bytes.
Blob.getBytes		Extracts the range of data specified by the argument from the BLOB or BINARY data obtained by ResultSet.next.	Acquires from the connected database the range of BLOB or BINARY data specified by the argument in segments of ACCESSSIZE x 1024 bytes.
CallableStatement.getBinaryStream ResultSet.getBinaryStream ResultSet.getAsciiStream ResultSet.getUnicodeStream Blob.getBinaryStream		When the InputStream read method obtained by the executed method is executed, the JDBC driver extracts data from the BLOB or BINARY data obtained by ResultSet.next.	When the InputStream read method obtained by the executed method is executed, the JDBC driver acquires data from the connected database.
Blob.length		Acquires the data length from the BLOB or BINARY data obtained by ResultSet.next.	Acquires the data length from the connected database.
Blob.position		Acquires the position of the data matching the search pattern from the BLOB or BINARY data obtained by ResultSet.next.	Acquires the position of the data matching the search pattern from the connected database.
InputStream obtained by CallableStatement.getBinaryStream, ResultSet.getBinaryStream, or Blob.getBinaryStream	InputStream.available	Returns a value equal to or less than the length of the actual data indicated by the locator.	Returns a value equal to or less than ACCESSSIZE x 1024 bytes.
	InputStream.skip	Skips reading data up to the range equal to or less than the length of the actual data indicated by the locator.	Skips reading data up to the maximum range of ACCESSSIZE x 1024 bytes.

Execution method	LONGVARBINARY_ACCESS specification value#	
	REAL	LOCATOR
CallableStatement.getCharcterStream ResultSet.getCharcterStream	When the Reader read method obtained by getCharcterStream is executed, the JDBC driver extracts data from the BLOB or BINARY data obtained by ResultSet.next.	When the Reader read method obtained by getCharcterStream is executed, the JDBC driver acquires data from the connected database.

Legend:

ACCESSSIZE: Specification value of HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

InputStream and Reader: Classes of objects returned by getBinaryStream, getAsciiStream, or getCharacterStream of the JDBC driver

#: Note that REAL is assumed for an access to a BINARY column with a definition length of 1,024 bytes or less.

Notes about execution performance

When LOCATOR is specified for LONGVARBINARY_ACCESS, execution performance might drop compared to when REAL is specified.

When REAL is specified, the JDBC driver accesses the connected database once during processing of ResultSet.next or CallableStatment.execute to obtain the locator. On the other hand, when LOCATOR is specified, in addition to the one access during processing of ResultSet.next or CallableStatment.execute, the JDBC driver accesses the connected database once to obtain the data length and once to obtain the data during execution of a data acquisition method (such as getBytes).

Notes about auto-commit

Even when auto-commit is enabled, commit processing does not occur in the following cases:

- Execution of a stored procedure in which LOCATOR is specified for LONGVARBINARY_ACCESS and an output parameter that satisfies either of the following conditions is specified:
  - Output parameter is BINARY type with a size greater than 1,024 bytes
  - Output parameter is BLOB type
- Execution of a stored procedure using the results-set return facility

Notes about data manipulation after the transaction terminates

When LOCATOR is specified for LONGVARBINARY_ACCESS, data manipulation cannot be performed if the transaction terminates during the period between acquisition of the SQL execution results (ResultSet.next or CallableStatement.execute) and the data manipulation (such as Blob.getBytes or InputStream.read). Data manipulation cannot be executed after the transaction terminates even if the HIRDB_CURSOR specification is TRUE.

Thus, you must ensure that all data manipulation will be performed before the transaction terminates.

(j) HiRDB_for_Java_SQL_IN_NUM

~<unsigned integer>((1-30000))<<300>>

Specifies the maximum number of input ? parameters in the SQL statements to be executed.

This specification becomes the number of input ? parameters that are obtained during SQL preprocessing. If the actual number of input ? parameters is greater than the specification value of this property, the JDBC driver gets the input ? parameter information from the HiRDB server after SQL preprocessing.

If the specification value is invalid, the JDBC driver throws an SQLException.

Note

If you will not be executing SQL statements that have input ? parameters, you should specify 1.

(k) HiRDB_for_Java_SQL_OUT_NUM

~<unsigned integer>((1-30000))<<300>>

Specifies the maximum number of output items for the SQL statements that are to be executed.

This specification becomes the number of output items obtained during SQL preprocessing. If the actual number of output items is greater than the specification value of this property, the JDBC driver gets output item information from the HiRDB server after SQL preprocessing.

If the specification value is invalid, the JDBC driver throws an SQLException.

Note

If you will not be executing SQL statements that have output items, you should specify 1.

(l) HiRDB_for_Java_SQLWARNING_LEVEL

Specifies the retention level for warning information that is issued during execution of SQL statements. For details about the retention levels for warning information, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*.

`IGNORE`: Retain warning information at the `IGNORE` level.

`SQLWARN`: Retain warning information at the `SQLWARN` level.

`ALLWARN`: Retain warning information at the `ALLWARN` level.

If this specification is omitted, `SQLWARN` is assumed.

If the specification value is invalid, the JDBC driver throws an `SQLException`.

(m) HiRDB_for_Java_ENV_VARIABLES

Specifies HiRDB client environment definitions in the following format:

variable-name=*value*;*variable-name*=*value*;...;*variable-name*=*value*

For details about client environment definitions supported by the JDBC driver, see *18.10 Supported client environment definitions*. If a client environment definition that is not supported by the JDBC driver is specified in a variable name, the JDBC driver ignores the specification. Note that variable names are case sensitive.

For details about the priorities for connection information that can be specified in multiple ways, see *18.11 Connection information priorities*.

Specification example

```
java.util.Properties prop;
prop=new java.util.Properties();
prop.setProperty("HiRDB_for_Java_ENV_VARIABLES",
 "PDFESHOST=FES1;PDCWAITTIME=0");
```

(n) HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR

Specifies whether `Statement` objects are to be validated or invalidated after HiRDB executes commit processing.

`TRUE`: Validate `Statement` objects after commit processing.

`FALSE`: Invalidate `Statement` objects after commit processing.

The objects that become invalid after commit execution are SQL statements that were precompiled by the `prepareStatement` or `prepareCall` method of the `Connection` class, and `ResultSet` class objects obtained by the `executeQuery` method of `Statement`.

If this specification is omitted, the value specified for STATEMENT_COMMIT_BEHAVIOR in the URL becomes effective.

Note

For notes about specification of this property, see *(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(o) HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

~<unsigned integer>((0-2097151))<<0>> (kilobytes)

Specifies the length of JDBC SQL-type LONGVARBINARY data to be requested at one time to the HiRDB server. If LONGVARBINARY_ACCESS specifies data other than LOCATOR data, this specification is invalid.

For example, suppose that 20 is specified for this property and the getBytes method of ResultSet attempts to get 100 kilobytes of JDBC SQL-type LONGVARBINARY data stored in the database. In such a case, the JDBC driver gets and returns the data by dividing the operation into five executions of 20 kilobytes each. If 0 is specified, the JDBC driver requests the data all at once.

If the specification value is invalid, the JDBC driver throws an SQLException.

Note

For notes about specification of this property, see *(i) LONGVARBINARY_ACCESS*.

(p) HiRDB_for_Java_MAXBINARYSIZE

~<unsigned integer>((0-2147483647)) (bytes)

Specifies the maximum data size during acquisition of JDBC SQL-type LONGVARBINARY data.

When the JDBC driver is getting JDBC SQL-type LONGVARBINARY data, it allocates memory of the defined length because it cannot determine the actual data length until it actually gets the data. Therefore, to get the value of a string for which the specified size is large (for example, 2,147,483,647 bytes, which is the maximum length for HiRDB's BINARY and BLOB data types), the JDBC driver attempts to allocate 2,147,483,647 bytes of memory, because that is the defined length. Consequently, a memory shortage may occur, depending on the execution environment.

You should specify for this property the maximum length of the data that is actually stored. If the data length of the BINARY or BLOB data to be acquired is larger than the size specified by this property, the JDBC driver truncates the acquired data to the specified size. When the JDBC driver does truncate data, it receives a warning from the HiRDB server when it executes the next method of ResultSet. In response to the received warning, the JDBC driver throws an SQLException or generates (or ignores) an SQLWarning, as determined by the specification of setLONGVARBINARY_TruncError.

If no upper limit is set by this property, the defined length of the target acquisition data becomes the upper limit.

If the specification value is invalid, the JDBC driver throws an SQLException.

Note

> This property value is ignored when a BLOB column or a BINARY column whose definition length is greater than 1,024 bytes is accessed with LOCATOR specified for LONGVARBINARY_ACCESS. The JDBC driver allocates an area based on the actual data length and gets the entire data.

(q) HiRDB_for_Java_LONGVARBINARY_TRUNCERROR

Specifies whether an exception is or is not to be thrown if truncation occurs during acquisition of JDBC SQL-type LONGVARBINARY data.

TRUE: Throw an exception if truncation occurs.

FALSE: Do not throw an exception if truncation occurs.

If this specification is omitted, TRUE is assumed.

If IGNORE is specified for HiRDB_for_Java_SQLWARNING_LEVEL, the JDBC driver operates as if FALSE were specified for this property.

Any truncation that occurs during acquisition of JDBC SQL-type LONGVARBINARY data indicates that the following condition is satisfied:

Actual length of JDBC SQL-type LONGVARBINARY *data obtained during SQL execution* > *data length specified by* HiRDB_for_Java_MAXBINARYSIZE

(r) HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR

Specifies whether preprocessing results are to be ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes) when the Statement or ResultSet object is specified to remain valid after commit processing.[#1]

This specification can be made by using the HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR system property.

If the server is XDM/RD E2, this specification is ignored.

TRUE: Ignore preprocessing results.

FALSE: Do not ignore preprocessing results.

If the specified value is invalid, the JDBC driver throws an SQLException.

The value of this property is not case sensitive.

If this specification is omitted, the value of the STATEMENT_CLOSE_BEHAVIOR item in the URL takes effect. If this item is omitted in the URL, FALSE is assumed as the value of the property.

If NO is set in both the PDDDLDEAPRP and the PDDDLDEAPRPEXE client environment definitions and another user executes a definition SQL statement for a schema resource (table or index) to be accessed by an SQL statement (SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement), the definition SQL statement is placed on lock-release wait status. This lock-release wait status is released at the times described below. Note that if you specify TRUE, you can release this status during commit processing.

- If Statement or ResultSet object is specified to remain valid after commit processing
  - If TRUE is specified

    When commit processing is performed after Statement close#2 or when the connection is closed.
  - If FALSE is specified

    When the connection is closed.
- Other

  When commit processing is performed#2 or when the connection is closed.

#1

One of the following applies:

1. TRUE is specified for the following items in the properties that are specified by the getConnection method:

   HIRDB_CURSOR

   HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
2. TRUE is specified for the following items in the URL that is specified by the getConnection method:

   HIRDB_CURSOR

   STATEMENT_COMMIT_BEHAVIOR
3. true is specified in the following methods of the PrdbDataSource, PrdbConnectionPoolDataSource, or PrdbXADataSource class:

   setHiRDBCursorMode

   setStatementCommitBehavior

4. The Connection.setHoldability method (with ResultSet.HOLD_CURSORS_OVER_COMMIT specified in the argument) is executed.
5. The following methods of the Connection class (with ResultSet.HOLD_CURSORS_OVER_COMMIT specified in the resultSetHoldability argument) are executed:

   createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

   prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

   prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

#2

In addition to explicit commit using the commit method, the following also apply:

- Implicit commit by automatic commit
- Execution of a definition SQL statement
- Execution of a PURGE TABLE statement
- Explicit rollback by rollback method
- Implicit rollback by an SQL execution error

If TRUE is specified and the Statement or ResultSet object is specified to remain valid after commit processing, HiRDB creates a transaction and executes DEALLOCATE PREPARE processing when the Statement is closed. Therefore, if the PDSWAITTIME client environment variable is specified, the interval monitoring specified in PDSWAITTIME begins. If automatic commit is disabled and neither an SQL execution, nor commit, nor disconnection is performed by another statement for an extended period after the Statement was closed, HiRDB closes the connection based on the interval monitoring.

If Cosminexus is connected and the connection pooling function is used by using the Type4 JDBC driver from DB Connector, you must specify FALSE. When the connection pooling function is used, connection returns to the pool and the transaction remains even if a close method for connection is called from the application. Therefore, you must execute commit processing after Statement is closed.

If you use the statement pooling facility, you must either omit this specification or specify FALSE.

(s) HiRDB_for_Java_DBID

Specifies the port number of HiRDB (information corresponding to PDNAMEPORT) or the HiRDB client's environment variable group file name. For details about the specification, see the explanation about DBID in *18.2.2(1)(b) Explanation of URL items*.

(t) HiRDB_for_Java_DBHOST

Specifies the host name of HiRDB. For details about the specification, see the explanation about DBHOST in *18.2.2(1)(b) Explanation of URL items*.

(u) HiRDB_for_Java_HiRDB_INI

Specifies the absolute path of the directory that contains the HiRDB.INI file. This property is applicable when the HiRDB client environment variables specified in the HiRDB.INI file are to be in effect, in which case the HiRDB client environment variables in the HiRDB.ini file located in the specified directory will take effect. If the specified directory does not contain a HiRDB.ini file and this is an interval driver, this specification is ignored.

If this specification is omitted, the value specified for the HiRDB_INI item in the URL is assumed. If the HiRDB_INI item is omitted in the URL, the contents of this file are ignored.

(v) HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR

Specifies whether an update count that is compliant with the JDBC standard is to be set for the return value of the getUpdateCounts method of java.sql.BatchUpdateException.

This property is applicable when the version of the connection-target HiRDB is 08-02 or later.

TRUE: Set an update count that is compliant with the JDBC standard.

FALSE: Set a HiRDB-specific update count.

The value of this property is not case sensitive.

If this specification is omitted, the BATCHEXCEPTION_BEHAVIOR value in the URL items takes effect. If this specification is omitted in the URL, the JDBC driver assumes that TRUE is specified.

18.3 Database connection using a DataSource object and JNDI

The JDBC2.0 Optional Package can now use database connections that use a `DataSource` object and JNDI.

Although use of JNDI is not required, the advantage of using it is that you only have to set up the connection information once. The standard JDK package does not include interface definitions for the `DataSource` class or JNDI, so you have to download these items from the JavaSoft Web site when you develop an AP.

To connect a database by using a `DataSource` object and JNDI:

1. Generate the `DataSource` object.
2. Set up the connection information.
3. Register the `DataSource` object into JNDI.
4. Get the `DataSource` object from JNDI.
5. Connect to the database.

If you are not using JNDI, steps 3 and 4 are not necessary.

If you are using JNDI, steps 1 to 3 need to be executed only once. Thereafter, you can connect to the database by performing only steps 4 and 5. Once you have performed step 4, you can change the connection information as necessary.

(1) Generating the DataSource object

Generate the `DataSource` class object to be provided by the JDBC driver.

The `DataSource` class name of the JDBC driver, which is necessary for generating the `DataSource` class object, is `PrdbDataSource`.

Below is an example of generating the `DataSource` class object:

```
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource() ;
```

(2) Setting up connection information

Call the method for setting up connection information for the `DataSource` object, and set up the connection information. Because there is also a method for acquiring connection information, you can use it to check the current connection information. For details about the connection information setup and acquisition methods, see *18.7 Connection information setup and acquisition interface*.

(3) Registering the DataSource object into JNDI

Register the `DataSource` object into JNDI.

JNDI can select from among several service providers, depending on the execution environment.

Shown below is an example of registering the `DataSource` object into JNDI (this example is for Windows). In the registration example, the File System service provider, which is one of the service providers, is used. For details about other service providers, see the JNDI documentation.

```
// Generate DataSource class object to be provided by JDBC driver.
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource();

// Set connection information.
        :


// Get system properties.
Properties sys_prop = System.getProperties() ;

// Set properties of File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set directory to be used by File System service provider.
// (Register under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");

// Update system properties.
System.setProperties(sys_prop) ;


// Initialize JNDI.
Context ctx = new InitialContext();

// Register DataSource class object to be provided by HiRDB driver
// into JNDI. Use logical name jdbc/TestDataSource.
ctx.bind("jdbc" + "\\" + "TestDataSource", ds);
                      :
```

When you register the logical name to be registered into JNDI, the JDBC2.0 specifications recommend that you register the logical name under a subcontext called `jdbc` (`jdbc/TestDataSource` in the registration example).

(4) Getting the DataSource object from JNDI

Get the `DataSource` object from JNDI.

Shown below is a registration example for the `DataSource` object (this is an example for Windows). This registration example uses the File System service provider, which is one of the service providers. For details about other service providers, see the JNDI documentation.

```
// Get system properties.
Properties sys_prop = System.getProperties() ;

// Set properties of File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set directory to be used by File System service provider.
// (Register under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");


// Update system properties.
System.setProperties(sys_prop) ;

// Initialize JNDI.
Context ctx = new InitialContext();


// Get object of local name jdbc/TestDataSource from JNDI.
Object obj = ctx.lookup("jdbc" + "\\" + "TestDataSource") ;

// Cast retrieved object to DataSource class type.
DataSource ds = (DataSource)obj;
              :
```

(5) Connecting to the database

Call the `getConnection` method for the `DataSource` object.

Shown below is an example of calling the `getConnection` method.

```
DataSource ds

// Get DataSource object from JNDI.
                     :


// Issue getConnection method.
Connection con = ds.getConnection();
  or
Connection con = ds.getConnection("USERID", "PASSWORD");#
```

#

The method's arguments (authorization identifier and password) take priority over the connection information that was set for the `DataSource` object. If needed connection information has not been set for the `DataSource` object, or if the contents of the connection information are invalid, or if connection with the HiRDB server fails, the `getConnection` method throws an `SQLException`.

After getting the `DataSource` object from JNDI, set up the connection information again, as necessary. In this case, you must cast the `DataSource` object to the `DataSource` class type provided by the JDBC driver before you set up the information. An example is shown below:

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource hirdb_ds;

// Get DataSource object from JNDI.
                     :


// Cast DataSource object to DataSource class type provided
// by JDBC driver.
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource)ds;

// Set up connection information again.
          :
```

18.4 JDBC1.2 core API

18.4.1 Driver interface

(1) Overview

The `Driver` interface provides the following principal functions:

- Database checking
- Validity check on a specified URL
- Acquisition of connection properties specified with the `DriverManager.getConnection` method
- Return of the driver version

(2) Methods

The table below lists the methods of the `Driver` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 18-7: Driver interface methods

Subsection	Method	Function
(a)	*acceptsURL(String url)*	Checks whether the driver can connect to the specified URL.
(b)	*connect(String url, Properties info)*	Attempts database connection to the specified URL.
(c)	*getMajorVersion()*	Acquires the driver's major version.
(d)	*getMinorVersion()*	Acquires the driver's minor version.
(e)	*getPropertyInfo(String url, Properties info)*	Acquires information about the driver's valid properties.
(f)	*jdbcCompliant()*	Reports whether the driver is JDBC Compliant^{TM}.

(a) acceptsURL(String url)

Function

This method checks whether the driver can connect to the specified URL.

Format

```
public boolean acceptsURL(String url) throws SQLException
```

Arguments

String url

URL of the database

Return value

The method returns `true` if the driver recognizes the specified URL; if not, the method returns `false`.

Exceptions

None.

(b) connect(String url, Properties info)

Function

This method attempts database connection to the specified URL.

Format

```
public Connection connect(String url, Properties info)
throws SQLException
```

Arguments

String url

Specifies the URL of the connected database. For details, see *18.2.2(1) URL syntax*.

Properties info

Specifies a list of property name and value pairs as the connection arguments. For details, see *18.2.2(2) User properties*.

Return value

`Connection` object indicating connection to the URL

Functional detail

Connects to the database indicated by the URL.

This method uses the value returned by `DriverManager.getLoginTimeout` as the maximum wait time for communication.

If `getLoginTimeout` returns `0`, the value specified in the `PDCONNECTTIME` client environment definition is used as the maximum wait time.

You can use `DriverManager.setLoginTimeout` to specify the wait time.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurs.
- The specified connection information is invalid.
- The return value of `DriverManager.getLoginTimeout` is not within the range from `0` to `300`.

(c) getMajorVersion()

Function

This method acquires the driver's major version.

Format

```
public synchronized int getMajorVersion()
```

Arguments

None.

Return value

This method returns this driver's major version number.

Exceptions

None.

(d) getMinorVersion()

Function

This method acquires the driver's minor version.

Format

```
public synchronized int getMinorVersion()
```

Arguments

None.

Return value

This method returns this driver's minor version number.

Exceptions

None.

(e) getPropertyInfo(String url, Properties info)

Function

This method acquires information about this driver's valid properties.

Format

```
public synchronized DriverPropertyInfo[]
getPropertyInfo(String url, Properties info) throws
SQLException
```

Arguments

String url

Specifies the URL of the connected database.

Properties info

Specifies a list of property name and value pairs as the connection arguments.

Return value

This method returns the array of the `DriverPropertyInfo` object for specifying valid properties (if properties are not needed, this array might be empty).

The following table lists the settings for the fields of `DriverPropertyInfo`.

Table 18-8: Settings for fields of DriverPropertyInfo

Property name	DriverPropertyInfo field				
	name	value	description	required	choices
user	Same as the property name	null	"UserID"	false	null
password	ditto	""	"Password"	false	null
UAPNAME	ditto	""	"UAPNAME"	false	null
JDBC_IF	ditto	"OFF"	"JDBC Interface Trace"	false	{"ON","OFF" }
TRC_NO	ditto	"500"	"Trace Entry Number"	false	null

Property name	DriverPropertyInfo field				
	name	value	description	required	choices
ENCODELANG	ditto	null	"Encode Lang"	false	null
HIRDB_CURSOR	ditto	"FALSE"	"HiRDB Cursor across commit"	false	null
LONGVARBINARY_ACCESS	ditto	"REAL"	"Longvarbinary locator access"	false	null
HiRDB_for_Java_SQL_IN_NUM	ditto	"300"	"SQL In Number"	false	null
HiRDB_for_Java_SQL_OUT_NUM	ditto	"300"	"SQL Out Number"	false	null
HiRDB_for_Java_SQLWARNING_LEVEL	ditto	"SQLWARN"	"SQL Warning Level"	false	null
HiRDB_for_Java_ENV_VARIABLES	ditto	null	"HiRDB Environment Variables"	false	null
HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	ditto	"TRUE"	"HiRDB Statement across commit"	false	{"TRUE","FALSE"}
HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE	ditto	0	"Longvarbinary locator access size"	false	null
HiRDB_for_Java_MAXBINARYSIZE	ditto	null	"Longvarbinary maximum binary size"	false	null
HiRDB_for_Java_LONGVARBINARY_TRUNCERROR	ditto	"TRUE"	"Longvarbinary truncate error"	false	{"TRUE","FALSE"}
HiRDB_for_Java_DBID	ditto	null	"Port number of HiRDB server or Environment variable group of HiRDB"	false	null
HiRDB_for_Java_DBHOST	ditto	null	"Host name with HiRDB"	false	null
HiRDB_for_Java_HiRDB_INI	ditto	null	"HiRDB.ini file "	false	null

Property name	DriverPropertyInfo field				
	name	value	description	required	choices
HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR	ditto	"TRUE"	"BatchUpdateException UpdateCounts that conforms to JDBC standard"	false	{"TRUE","FALSE"}
SQLWARNING_IGNORE	ditto	"FALSE"	"Warning generated by the Connection object is not maintained with the Connection object"	false	{"TRUE","FALSE"}
XDSHOST	ditto	null	"Host name of XDS"	false	null
XDSPORT	ditto	null	"Port number of XDS"	false	null
XDSSRVTYPE	ditto	"WS"	"Server type of XDS"	false	{"PC","WS"}
HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR	ditto	"FALSE"	"HiRDB Statement close behavior"	false	{"TRUE","FALSE"}

Functional detail

This method analyzes the information specified in the `url` and `info` argument and returns information needed for connecting to the database.

If `acceptsURL(String url)` is `false`, this method returns `null`.

Exceptions

None.

(f) jdbcCompliant()

Function

This method reports whether this driver is JDBC Compliant^{TM}.

Format

```
public synchronized boolean jdbcCompliant()
```

Arguments

None.

Return value

If the driver is JDBC-compliant, this method returns `true`; if not, the method returns `false`.

Exceptions

None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `HiRDBDriver`

(4) Escape clause

The part enclosed in curly brackets ({ }) in SQL statements is called the escape clause. An escape clause consists of a keyword and parameters. The keyword is not case sensitive.

The following table lists the escape clauses.

Table 18-9: List of escape clauses

Type of escape clause	Keyword
Date, time, timestamp	d, t, ts
LIKE escape character	escape
Outer join	oj
Procedure call	call
Scalar function	fn
Assignment	set

For details about the scalar functions that can be specified in an escape clause, see Appendix *I. Scalar Functions That Can Be Specified in the Escape Clause*.

Analysis of escape syntax

You can use the `setEscapeProcessing` method of the `Statement` class to specify whether analysis of the escape syntax is to be enabled. When this specification is omitted, analysis of escape syntax is enabled. Analysis of escape syntax means that the JDBC driver checks each SQL statement for an escape clause. If an SQL statement contains an escape clause, the JDBC driver converts the SQL statement so that the statement can be executed by HiRDB.

18.4.2 Connection interface

(1) Overview

The Connection interface provides the following principal functions:

- Creation of the Statement, PreparedStatement, and CallableStatement class objects
- Transaction settlement (COMMIT or ROLLBACK)
- Specification of the AUTO commit mode

(2) Methods

The table below lists the methods of the Connection interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-10: Connection interface methods

Subsection	Method	Function
(a)	clearWarnings()	Clears all warnings reported for this Connection object.
(b)	close()	Closes the connection with HiRDB.
(c)	commit()	Applies all changes made since the most recent commit or rollback.
(d)	createStatement()	Creates a Statement object for sending an SQL statement to the database.
(e)	createStatement(int resultSetType, int resultSetConcurrency)	Creates a Statement object for sending an SQL statement to the database.
(f)	createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a Statement object for sending an SQL statement to the database.
(g)	getAutoCommit()	Acquires the current automatic commit mode for this Connection object.
(h)	getCatalog()	Acquires the current catalog name for this Connection object.
(i)	getHoldability()	Acquires the current holding facility for ResultSet objects that are created by using this Connection object.
(j)	getMetaData()	Creates a DatabaseMetaData object.

Subsectio n	Method	Function
(k)	getTransactionIsolation()	Acquires this Connection object's current transaction cut-off level.
(l)	getTypeMap()	Acquires the Map object related to this Connection object.
(m)	getWarnings()	Acquires as an SQLWarning object the warnings reported by calls related to this Connection object.
(n)	isClosed()	Acquires a value indicating whether this Connection object is closed.
(o)	isReadOnly()	Acquires a value indicating whether this Connection object is in read-only mode.
(p)	nativeSQL(String sql)	Converts any escape clause in a specified SQL statement to a format that can be executed by HiRDB, and then returns the SQL statement.
(q)	prepareCall(String sql)	Creates a CallableStatement object for executing a CALL statement.
(r)	prepareCall(String sql, int resultSetType, int resultSetConcurrency)	Creates a CallableStatement object for executing a CALL statement.
(s)	prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a CallableStatement object for executing a CALL statement.
(t)	prepareStatement(String sql)	Creates a PreparedStatement object for sending an SQL statement with parameters to the database.
(u)	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Creates a PreparedStatement object for sending an SQL statement with parameters to the database.
(v)	prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a PreparedStatement object for sending an SQL statement with parameters to the database.
(w)	rollback()	Undoes all changes made by the current transaction and releases all database locks currently held by this Connection object.
(x)	setAutoCommit(boolean autoCommit)	Sets the automatic commit mode status for this connection.
(y)	setCatalog(String catalog)	This driver ignores this method's specification because HiRDB does not support catalogs.

Subsection	Method	Function
(z)	setHoldability(int holdability)	Changes the holding facility for a `ResultSet` object that is created by using this `Connection` object.
(aa)	setReadOnly(boolean readOnly)	This method's value is ignored because HiRDB does not have a dedicated mode.
(ab)	setTransactionIsolation(int level)	This method's value is ignored because the value is always `TRANSACTION_REPEATABLE_READ` in HiRDB.
(ac)	checkSession(int waittime)	Checks the current connection status.
(ad)	setHiRDB_Audit_Info(int pos, String userinf)	Specifies user-specific connection information (user-added information).

(a) clearWarnings()

Function

Clears all warnings reported for this `Connection` object.

Once this method has been called, `getWarnings()` returns `null` until a new warning is issued for this `Connection` object.

Format

```
public void clearWarnings() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

If this `Connection` object is closed (`close`), the JDBC driver throws an `SQLException`.

(b) close()

Function

Closes the connection with HiRDB.

Format

```
public void close() throws SQLException
```

Arguments

None.

Return value

None.

Functional detail

During a normal connection, this method disconnects HiRDB, disables the corresponding objects, and releases any unneeded resources.

In a pooling or XA environment, the physical connection is not closed; instead, `PooledConnection.close()` is used to close the physical connection.

If execution of `Connection.close()` results in an error, the method does not throw an `SQLException`.

If execution of `Connection.close()` in a pooling or XA environment results in a fatal error and connection pooling becomes unavailable, `ConnectionEventListener.connectionErrorOccurred()` does not occur.

If the `close` method is called by a `Connection` object that is already closed, this method does nothing.

Exceptions

None.

(c) commit()

Function

Applies all changes made since the most recent commit or rollback.

If this method is called while the automatic commit mode is enabled, the method performs `commit` processing without throwing an exception.

Format

```
public void commit() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurs.
- `close()` has already been issued to the `Connection` object.

(d) createStatement()

Function

Creates a `Statement` object for sending an SQL statement to the database.

Format

```
public synchronized Statement createStatement() throws
SQLException
```

Arguments

None.

Return value

`Statement` object

Functional detail

This method creates a `Statement` object for sending an SQL statement to the database.

The holding facility for the `ResultSet` that is created from the `Statement` object created by this method is set to the value specified by `Connection.setHoldability`. If `Connection.setHoldability` has never been executed, the `HIRDB_CURSOR` property setting is used.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to the `Connection` object.
- Creation of the `Statement` object resulted in an error.

(e) createStatement(int resultSetType, int resultSetConcurrency)

Function

Creates a `Statement` object for sending an SQL statement to the database.

Format

```
public synchronized Statement createStatement(int
resultSetType, int resultSetConcurrency) throws SQLException
```

Arguments

int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

Return value

`Statement` object

Functional detail

This method creates a `Statement` object for sending an SQL statement to the database.

If `TYPE_SCROLL_SENSITIVE` is specified for the type of the result set, this driver changes it to `TYPE_SCROLL_INSENSITIVE` and then sets an `SQLWarning`.

For parallel processing mode, the driver supports only `CONCUR_READ_ONLY`. If `CONCUR_UPDATABLE` is specified, the driver changes it to `CONCUR_READ_ONLY` and then sets an `SQLWarning`.

The holding facility for the `ResultSet` that is created from the `Statement` object created by this method is set to the value specified by `Connection.setHoldability`. If `Connection.setHoldability` has never been executed, the `HIRDB_CURSOR` property setting is used.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to the `Connection` object.
- Creation of the `Statement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the type of the result set.
- A value other than a `ResultSet` literal is specified for parallel processing mode.

(f) createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

Function

Creates a `Statement` object for sending an SQL statement to the database.

Format

```
public synchronized Statement createStatement(int
resultSetType, int resultSetConcurrency, int
resultSetHoldability) throws SQLException
```

Arguments

int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

int resultSetHoldability

Holding facility for `ResultSet`

Return value

`Statement` object

Functional detail

This method creates a `Statement` object for sending an SQL statement to the database.

If `TYPE_SCROLL_SENSITIVE` is specified for the type of the result set, this driver changes it to `TYPE_SCROLL_INSENSITIVE` and then sets an `SQLWarning`.

For parallel processing mode, the driver supports only `CONCUR_READ_ONLY`. If `CONCUR_UPDATABLE` is specified, the driver changes it to `CONCUR_READ_ONLY` and then sets an `SQLWarning`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to the `Connection` object.
- Creation of the `Statement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the type of the result set.
- A value other than a `ResultSet` literal is specified for parallel processing mode.
- A value other than a `ResultSet` literal is specified for the holding facility for the `ResultSet`.

(g) getAutoCommit()

Function

Acquires the current automatic commit mode for this `Connection` object.

Format

```
public boolean getAutoCommit() throws SQLException
```

Arguments

None.

Return value

Current automatic commit mode for the `Connection` object

Exceptions

If `close()` has already been issued to this `Connection` object, the JDBC driver throws an `SQLException`.

(h) getCatalog()

Function

Acquires the current catalog name for this `Connection` object.

This method always returns `null`.

Format

```
public synchronized String getCatalog() throws SQLException
```

Arguments

None.

Return value

This method always returns `null` because there is no catalog in HiRDB.

Exceptions

If `close()` has already been issued to this `Connection` object, the JDBC driver throws an `SQLException`.

(i) getHoldability()

Function

Acquires the current holding facility for `ResultSet` objects that are created by using this Connection object.

This method's return value will be the holding facility for a ResultSet object if no holding facility was specified when the Statement (PreparedStatement) object was created.

Format

```
public synchronized int getHoldability() throws SQLException
```

Arguments

None.

Return value

One of the following ResultSet types:

- ResultSet.HOLD_CURSORS_OVER_COMMIT
- ResultSet.CLOSE_CURSORS_AT_COMMIT

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(j) getMetaData()

Function

Creates a DatabaseMetaData object.

Format

```
public synchronized DatabaseMetaData getMetaData() throws
SQLException
```

Arguments

None.

Return value

DatabaseMetaData object

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(k) getTransactionIsolation()

Function

Acquires this Connection object's current transaction cut-off level.

This method always returns TRANSACTION_REPEATABLE_READ.

Format

```
public int getTransactionIsolation() throws SQLException
```

Arguments

None.

Return value

This method always returns TRANSACTION_REPEATABLE_READ.

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(l) getTypeMap()

Function

Acquires the Map object related to this Connection object.

This driver returns an empty java.util.HashMap object that contains no information.

Format

```
public synchronized java.util.Map getTypeMap() throws
SQLException
```

Arguments

None.

Return value

Empty java.util.HashMap object

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(m) getWarnings()

Function

Acquires as an SQLWarning object the warnings reported by calls related to this Connection object.

Format

```
public SQLWarning getWarnings() throws SQLException
```

Arguments

None.

Return value

First SQLWarning object (if there is no such SQLWarning object, the method returns null)

Functional detail

This method acquires the SQLWarning object held by the corresponding Connection object.

By executing the getNextWarning method of the acquired SQLWarning object, you can acquire the next warning.

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(n) isClosed()

Function

Acquires a value indicating whether this Connection object is closed.

Format

```
public boolean isClosed()
```

Arguments

None.

Return value

If this Connection object is closed, the method returns true; if not, the method returns false.

Functional detail

This method acquires a value indicating whether this `Connection` object is closed. The database connection is closed when the `close` method is called or when a specific fatal error has occurred. This method is guaranteed to return `true` only when this method is called after the `Connection.close` method. This method cannot be used to determine whether the database connection is valid.

Exceptions

None.

(o) isReadOnly()

Function

Acquires a value indicating whether this `Connection` object is in the read-only mode.

This driver always returns `false`.

Format

```
public synchronized boolean isReadOnly() throws SQLException
```

Arguments

None.

Return value

This method always returns `false`.

Exceptions

If `close()` has already been issued to this `Connection` object, the JDBC driver throws an `SQLException`.

(p) nativeSQL(String sql)

Function

Converts any escape clause in a specified SQL statement to a format that can be executed by HiRDB, and then returns the SQL statement.

Format

```
public String nativeSQL(String sql) throws SQLException
```

Arguments

String sql

SQL statement that has not been converted

Return value

SQL statement that can be executed by HiRDB (if the `sql` argument is `NULL`, the method returns `NULL`; if it is the null character, the method returns the null character)

Functional detail

This method converts any escape clause in a specified SQL statement to a format that can be executed by HiRDB, and then returns the SQL statement.

The following are the syntax rules for escape clauses:

```
escape-clause ::= escape-sequence-for-date-or-time-or-timestamp
                | LIKE-escape-sequence
                | outer-join-escape-sequence
                | procedure-call-escape-sequence
                | scalar-function-escape-sequence
                | assignment-escape-sequence

escape-sequence-for-date-or-time-or-timestamp ::= date-escape-sequence
                                                | time-escape-sequence
                                                | timestamp-escape-sequence

date-escape-sequence ::=
          escape-start-code d default-character-string-representation-of-date-data escape-end-code

time-escape-sequence ::=
          escape-start-code t default-character-string-representation-of-time-data escape-end-code

timestamp-escape-sequence ::=
          escape-start-code ts default-character-string-representation-of-timestamp-data escape-end-code

LIKE-escape-sequence ::=
          escape-start-code escape escape-character escape-end-code

outer-join-escape-sequence ::= escape-start-code oj joined-table escape-end-code

procedure-call-escape-sequence ::=
          escape-start-code call [authorization identifier.]routine-identifier
          [([argument[,argument]...])] escape-end-code

assignment-escape-sequence ::= escape-start-code set assignment-statement escape-end-code

scalar-function-escape-sequence ::= escape-start-code fn scalar-function escape-end-code
scalar-function ::= scalar-function-in-default-format#

escape-start-code ::= '{'
escape-end-code ::= '}'
```

#

For details about the scalar function in the default format, see Appendix *I. Scalar Functions That Can Be Specified in the Escape Clause*.

For details about the underlined parts, see the manual *HiRDB Version 9 SQL Reference*. Note that an escape clause cannot be specified in an underlined part. Because the JDBC driver does not perform syntax analysis on the underlined parts, they will remain the same after conversion and will be subject to syntax analysis by the HiRDB server.

The following keywords can be used in escape sequences (these keywords are not case sensitive):

1. `d` in a date escape sequence
2. `t` in a time escape sequence
3. `ts` in a timestamp escape sequence
4. `escape` in a `LIKE` escape sequence
5. `oj` in an outer join escape sequence
6. `call` in a procedure call escape sequence
7. `fn` in a scalar function escape sequence
8. `set` in an assignment escape sequence

The escape clause entry rules are as follows:

- The single-byte space is used as the delimiter character in an escape clause.
- Delimiters can be inserted following an escape start code, following a keyword, and before an escape end code.
- In a procedure call escape sequence, insert delimiters immediately after `call`.
- You can specify multiple escape clauses in a single SQL statement.
- Curly brackets ({}) in a comment (a character string enclosed by`/*` and `*/`) and curly brackets enclosed in single (`'`) or double (`"`) quotation marks are not treated as part of an escape clause.
- The driver converts the escape clauses in an SQL statement to a format that can be executed by HiRDB. Note that only the part of each escape clause that is bracketed by the curly brackets is converted. The driver converts nothing outside the escape clauses.

The following table shows the escape clause conversion rules.

Table 18-11: Escape clause conversion rules

Escape clause	Before conversion	After conversion
Date	*escape-start-code* `d` *default-character-string-representation-of-date-data escape-end-code*	*default-character-string-representation-of-date-data*
Time	*escape-start-code* `t` *default-character-string-representation-of-time-data escape-end-code*	*default-character-string-representation-of-time-data*
Timestamp	*escape-start-code* `ts` *default-character-string-representation-of-timestamp-data escape-end-code*	*default-character-string-representation-of-timestamp-data*
`LIKE`	*escape-start-code* `escape` *escape-character escape-end-code*	`escape` *escape-character*
Outer join	*escape-start-code* `oj` *joined-table escape-end-code*	*joined-table*
Procedure call	*escape-start-code* `call` `[`*authorization identifier*`.]`*routine-identifier*`[([`*argument*`[,`*argument*`]...])]` *escape-end-code*	For HiRDB connection: `call[`*authorization identifier*`.]`*routine-identifier*`([`*argument*`[,`*argument*`]...])`#1 For XDM/RD E2 connection: `call[`*authorization identifier*`.]`*routine-identifier*`[([`*argument*`[,`*argument*`]...])]`
Scalar function	*escape-start-code* `fn` *scalar-function escape-end-code*	*scalar-function-in-HiRDB-format*#2
Assignment	*escape-start-code* `set` *assignment-statement escape-end-code*	`set` *assignment-statement*

#1

If the parentheses following the routine identifier are missing, the driver adds them.

#2

For details about the conversion processing, see conversion processing for the scalar function escape clause below.

The driver converts a scalar function in the default format to the HiRDB format. For an XDM/RD E2 connection, the driver converts it to the XDM/RD E2 format.

If a HiRDB scalar function corresponding to the default format is HiRDB's system-defined scalar function, the driver adds MASTER at the beginning of the function name.

Table 18-12 shows the conversion formats of scalar functions whose default format differs from the HiRDB format of scalar functions, and *Table 18-13* shows the conversion formats of scalar functions whose default format differs from the XDM/RD E2 format.

In general, the driver does not check the number of arguments in scalar functions. However, if the scalar function name is LOCATE, the driver converts a comma indicating an argument delimiter to IN or FROM, thereby checking the number of arguments.

Table 18-12: Conversion formats of scalar functions whose default format differs from the HiRDB format

Scalar function	Format before conversion	Format after conversion (HiRDB format)
Mathematical function	CEILING(number)	MASTER.CEIL(number)
	LOG(float)	MASTER.LN(float)
	TRUNCATE(number, places)	MASTER.TRUNC(number, places)
String functions	CHAR(code)	MASTER.CHR(code)
	INSERT(string1, start, length, string2)	MASTER.INSERTSTR(string 1, start, length, string2)
	LCASE(string)	LOWER(string)
	LEFT(string, count)	MASTER.LEFTSTR(string, count)
	LOCATE(string1, string2[, start])	POSITION(string1 IN string2 [FROM start])
	RIGHT(string, count)	MASTER.RIGHTSTR(string, count)
	SUBSTRING(string, start, length)	SUBSTR(string, start, length)
	UCASE(string)	UPPER(string)
Time and date functions	CURDATE()	CURRENT DATE
	CURRENT_DATE()	CURRENT DATE
	CURTIME()	CURRENT TIME

Scalar function	Format before conversion	Format after conversion (HiRDB format)
	CURRENT_TIME()	CURRENT TIME
	CURRENT_TIME(time-precision)	CURRENT TIME
	NOW()	CURRENT TIMESTAMP(6)
System function	USER()	USER

Table 18-13: Conversion formats of scalar functions whose default format differs from the XDM/RD E2 format

Scalar function	Format before conversion	Format after conversion (XDM/RD E2 format)
Mathematical function	LOG(float)	LN(float)
String functions	LCASE(string)	LOWER(string)
	LOCATE(string1, string2[, start])	POSITION(string1 IN string2 [FROM start])
	LTRIM(string)	TRIM(LEADING FROM string)
	RTRIM(string)	TRIM(TRAILING FROM string)
	SUBSTRING(string, start, length)	SUBSTR(string, start, length)
	UCASE(string)	UPPER(string)
Time and date functions	CURDATE()	CURRENT DATE
	CURRENT_DATE()	CURRENT DATE
	CURTIME()	CURRENT TIME
	CURRENT_TIME()	CURRENT TIME
	NOW()	CURRENT TIMESTAMP(6)
System function	USER()	USER

The following shows examples of conversion for HiRDB connection:

Scalar function		Before conversion	After conversion
Scalar function whose default format is the same as the HiRDB format	System built-in scalar function	**{fn** ABS(number)**}**	ABS(number)
	System-defined scalar function	**{fn** ASCII(string)**}**	**MASTER.**ASCII(string)
Scalar function whose default format differs from the HiRDB format	System built-in scalar function	**{fn** UCASE(string)**}**	**UPPER**(string)
	System-defined scalar function	**{fn** CEILING(number)**}**	**MASTER.CEIL**(number)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to this `Connection` object.
- The format of an escape clause in the specified SQL statement is invalid:
  - { and a keyword are specified, but } is missing.
  - There is no procedure name after `{call`.
  - There is no space between `{call` and the procedure name.
  - `{call` *procedure-name* `(` is not followed by `)`.
  - `{? = call` is specified.
  - The `LOCATE` function is specified in an escape clause, but no argument is specified.
  - The `LOCATE` function is specified in an escape clause, but the number of arguments within the parentheses is invalid.
  - For XDM/RD E2 connection, the `LTRIM` or `RTRIM` function is specified in an escape clause, but no argument is specified.

(q) prepareCall(String sql)

Function

Creates a `CallableStatement` object for executing a `CALL` statement.

Format

```
public synchronized CallableStatement prepareCall(String
sql) throws SQLException
```

Arguments

String sql

SQL statement to be executed (you can specify a statement other than the CALL statement)

Return value

CallableStatement object

Functional detail

This method creates a CallableStatement object for executing a CALL statement.

The holding facility for the ResultSet that is created from the CallableStatement object created by this method is set to the value specified by Connection.setHoldability. If Connection.setHoldability has never been executed, the HIRDB_CURSOR property setting is used.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued to this Connection object.
- Creation of the CallableStatement object resulted in an error.

(r) prepareCall(String sql, int resultSetType, int resultSetConcurrency)

Function

Creates a CallableStatement object for executing a CALL statement.

Format

```
public synchronized CallableStatement prepareCall(String
sql, int resultSetType, int resultSetConcurrency) throws
SQLException
```

Arguments

String sql

SQL statement to be executed (you can specify a statement other than the CALL statement)

int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

Return value

CallableStatement object

Functional detail

This method creates a CallableStatement object for executing a CALL statement.

If TYPE_SCROLL_SENSITIVE is specified for the type of the result set, this driver changes it to TYPE_SCROLL_INSENSITIVE and then sets an SQLWarning.

For parallel processing mode, the driver supports only CONCUR_READ_ONLY. If CONCUR_UPDATABLE is specified, the driver changes it to CONCUR_READ_ONLY and then sets an SQLWarning.

The holding facility for the ResultSet that is created from the CallableStatement object created by this method is set to the value specified by Connection.setHoldability. If Connection.setHoldability has never been executed, the HIRDB_CURSOR property setting is used.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued to this Connection object.
- Creation of the CallableStatement object resulted in an error.
- A value other than a ResultSet literal is specified for the type of the result set.
- A value other than a ResultSet literal is specified for parallel processing mode.

(s) prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

Function

Creates a CallableStatement object for executing a CALL statement.

Format

```
public synchronized CallableStatement prepareCall(String
sql, int resultSetType, int resultSetConcurrency, int
resultSetHoldability) throws SQLException
```

Arguments

String sql

SQL statement to be executed (you can specify a statement other than the `CALL` statement)

int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

int resultSetHoldability

Holding facility for `ResultSet`

Return value

`CallableStatement` object

Functional detail

This method creates a `CallableStatement` object for executing a `CALL` statement.

If `TYPE_SCROLL_SENSITIVE` is specified for the type of the result set, this driver changes it to `TYPE_SCROLL_INSENSITIVE` and then sets an `SQLWarning`.

For parallel processing mode, the driver supports only `CONCUR_READ_ONLY`. If `CONCUR_UPDATABLE` is specified, the driver changes it to `CONCUR_READ_ONLY` and then sets an `SQLWarning`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to this `Connection` object.
- Creation of the `CallableStatement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the type of the result set.
- A value other than a `ResultSet` literal is specified for parallel processing mode.
- A value other than a `ResultSet` literal is specified for the holding facility for the `ResultSet`.

(t) prepareStatement(String sql)

Function

Creates a `PreparedStatement` object for sending an SQL statement with parameters to the database.

Format

```
public synchronized PreparedStatement
prepareStatement(String sql) throws SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

`PreparedStatement` object

Functional detail

This method creates a `PreparedStatement` object for sending an SQL statement with parameters to the database.

The holding facility for the `ResultSet` that is created from the `PreparedStatement` object created by this method is set to the value specified by `Connection.setHoldability`. If `Connection.setHoldability` has never been executed, the `HIRDB_CURSOR` property setting is used.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to this `Connection` object.
- Creation of the `PreparedStatement` object resulted in an error.

(u) prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

Function

Creates a `PreparedStatement` object for sending an SQL statement with parameters to the database.

Format

```
public synchronized PreparedStatement
prepareStatement(String sql, int resultSetType, int
resultSetConcurrency) throws SQLException
```

Arguments

String sql

SQL statement to be executed (you can specify a statement other than the `CALL` statement) int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

Return value

PreparedStatement object

Functional detail

This method creates a PreparedStatement object for sending an SQL statement with parameters to the database.

If TYPE_SCROLL_SENSITIVE is specified for the type of the result set, this driver changes it to TYPE_SCROLL_INSENSITIVE and then sets an SQLWarning.

For parallel processing mode, the driver supports only CONCUR_READ_ONLY. If CONCUR_UPDATABLE is specified, the driver changes it to CONCUR_READ_ONLY and then sets an SQLWarning.

The holding facility for the ResultSet that is created from the PreparedStatement object created by this method is set to the value specified by Connection.setHoldability. If Connection.setHoldability has never been executed, the HIRDB_CURSOR property setting is used.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued to this Connection object.
- Creation of the PreparedStatement object resulted in an error.
- A value other than a ResultSet literal is specified for the type of the result set.
- A value other than a ResultSet literal is specified for parallel processing mode.

(v) prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

Function

Creates a PreparedStatement object for sending an SQL statement with parameters to the database.

Format

```
public synchronized PreparedStatement
```

```
prepareStatement(String sql, int resultSetType, int
resultSetConcurrency, int resultSetHoldability) throws
SQLException
```

Arguments

String sql

SQL statement to be executed (you can specify a statement other than the `CALL` statement)

int resultSetType

Type of result set

int resultSetConcurrency

Parallel processing mode

int resultSetHoldability

Holding facility for `ResultSet`

Return value

`PreparedStatement` object

Functional detail

This method creates a `PreparedStatement` object for sending an SQL statement with parameters to the database.

If `TYPE_SCROLL_SENSITIVE` is specified for the type of the result set, this driver changes it to `TYPE_SCROLL_INSENSITIVE` and then sets an `SQLWarning`.

For parallel processing mode, the driver supports only `CONCUR_READ_ONLY`. If `CONCUR_UPDATABLE` is specified, the driver changes it to `CONCUR_READ_ONLY` and then sets an `SQLWarning`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to this `Connection` object.
- Creation of the `PreparedStatement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the type of the result set.
- A value other than a `ResultSet` literal is specified for parallel processing mode.
- A value other than a `ResultSet` literal is specified for the holding facility

for the `ResultSet`.

(w) rollback()

Function

Undoes all changes made by the current transaction and releases all database locks currently held by this `Connection` object.

If you call this method while the automatic commit mode is enabled, the method performs `rollback` processing without throwing an exception.

Format

```
public void rollback() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error has occurred.
- `close()` has already been issued to the `Connection` object.

(x) setAutoCommit(boolean autoCommit)

Function

Sets the automatic commit mode status for this connection.

Format

```
public void setAutoCommit(boolean autoCommit) throws
SQLException
```

Arguments

boolean autoCommit

Specifies `true` to enable the automatic commit mode and `false` to disable it.

Return value

None.

Functional detail

This method sets the automatic commit mode status for this connection. When a connection is in automatic commit mode, each of its SQL statements is committed as a separate transaction after it has executed. When a connection is not in automatic commit mode, its SQL statements are grouped as transactions to be terminated by a call to the `commit` method or as transactions to be terminated by a call to the `rollback` method. The default mode for a new connection is automatic commit mode.

Automatic commit is performed upon completion of an SQL statement. If the SQL statement returns a `ResultSet` object, the SQL statement is completed when the `ResultSet` object is closed.

A transaction that is executing when this method is called will not be committed.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `Connection` object is running under a distributed transaction.
- `close()` has already been issued to this `Connection` object.

(y) setCatalog(String catalog)

Function

This driver ignores this method's specification because HiRDB does not support catalogs.

Format

```
public synchronized void setCatalog(String catalog) throws
SQLException
```

Arguments

String catalog

Ignores the specified value because there is no catalog in HiRDB.

Return value

This method always returns `false`.

Exceptions

If `close()` has already been issued to this `Connection` object, the JDBC driver throws an `SQLException`.

(z) setHoldability(int holdability)

Function

Changes the holding facility for a `ResultSet` object that is created by using this `Connection` object.

Format

```
public synchronized void setHoldability(int holdability)
throws SQLException
```

Arguments

int holdability

Holding facility literals for `ResultSet` (`ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`)

Return value

None.

Functional detail

This method changes the holding facility for a `ResultSet` object that is created by using this `Connection` object.

The value of this method is the setting for `createStatement` (`prepareStatement`, `prepareCall`) that is called without the holding facility specified. It has no effect on an existing `Statement` or a `ResultSet` that is created (has been created) by that `Statement`.

The following table shows the default value for the holding facility.

Table 18-14: Default value for holding facility

HiRDB_CURSOR property value	Default value
TRUE	ResultSet.HOLD_CURSORS_OVER_COMMIT
FALSE	ResultSet.CLOSE_CURSORS_AT_COMMIT
Not specified	ResultSet.CLOSE_CURSORS_AT_COMMIT

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued to this `Connection` object.
- A value other than `ResultSet.HOLD_CURSORS_OVER_COMMIT` or

ResultSet.CLOSE_CURSORS_AT_COMMIT is specified in the holdability argument.

(aa) setReadOnly(boolean readOnly)

Function

This method's value is ignored because HiRDB does not have a dedicated mode.

Format

```
public synchronized void setReadOnly(boolean readOnly)
throws SQLException
```

Arguments

boolean readOnly

The specified value is ignored because there is no dedicated mode in HiRDB.

Return value

None.

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(ab) setTransactionIsolation(int level)

Function

This method's value is ignored because the value is always TRANSACTION_REPEATABLE_READ in HiRDB.

Format

```
public void setTransactionIsolation(int level) throws
SQLException
```

Arguments

int level

Transaction cut-off level

Return value

None.

Exceptions

If close() has already been issued to this Connection object, the JDBC driver throws an SQLException.

(ac) checkSession(int waittime)

Function

Checks the current connection status.

Format

```
public int checkSession (int waittime) throws SQLException
```

Arguments

int waittime:

Specifies the wait time (in seconds). If 0 is specified, the JDBC driver waits until the time specified by the PDCWAITTIME client environment definition.

Return value

PrdbConnection.SESSION_ALIVE:

The method was able to confirm that a connection is currently established.

PrdbConnection.SESSION_NOT_ALIVE:

Because of a cause other than a timeout within the time specified in the argument, the method was unable to confirm that a connection is currently established.

PrdbConnection.SESSION_CHECK_TIMEOUT:

Because of a timeout within the time specified in the argument, the method was unable to confirm that a connection is currently established.

Exceptions

If the waittime argument is a negative value, the driver throws a java.sql.SQLException.

(ad) setHiRDB_Audit_Info(int pos, String userinf)

Function

Specifies user-specific connection information (user-added information).

Format

```
public void  setHiRDB_Audit_Info(int pos, String userinf)
throws SQLException
```

Arguments

int pos

Item number of the user-added information:

1: User-added information 1

2: User-added information 2

3: User-added information 3

String userinf

User-added information. The data specified for user-added information is converted to the character set specified in the ENCODELANG property during connection establishment, or the setEncodeLang method, or the HiRDB server's character codes during HiRDB character data processing. Specify data whose length will not exceed 100 bytes after code conversion. To cancel an existing setting, specify the NULL value.

Return value

None.

Functional detail

This method specifies user-added information, such as the account information for an application that accesses the HiRDB server. The specified user-added information remains in effect until it is canceled. The specified user-added information is output to the audit trail when an SQL statement that uses the Statement, PreparedStatement, or CallableStatement object created by using the corresponding Connection object is executed.

This method corresponds to the specification of user-specific connection information (DECLARE AUDIT INFO SET) in an embedded SQL statement.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- The corresponding Connection object is closed.
- Code conversion of user-added information fails.
- A value other than 1, 2, or 3 is specified in pos.
- The length of the data obtained after code conversion of the user-added information exceeds 100 bytes.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbConnection

(4) Notes

(a) Holdability specification

If holdability is specified with one of the methods shown below, the HIRDB_CURSOR specification value in either the URL syntax or the properties can be overwritten for each Statement object (Statement, PreparedStatement or CallableStatement object) and Connection object:

- resultSetHoldability argument of the createStatement, preparedStatement or prepareCall method
- holdability argument of the setHoldability method
- Whether or not UNTIL DISCONNECT is specified in the SQL statement (SELECT statement) to be executed

For ResultSet and DatabaseMetaData objects (when the setHoldability method is used) generated by the applicable method, the holdability specifications that become effective change depending on the combinations of these specifications and the HIRDB_CURSOR specifications.

Table 18-5 shows the holdability specifications that become effective for Statement objects generated by the following methods:

- createStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)
- prepareStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)
- prepareCall(int resultSetType,int resultSetConcurrency, int resultSetHoldability)

Table 18-15: Effective holdability specifications (1/2)

HIRDB_CURSOR or setHiRDBCursorMode specification		Specification value of resultSetHoldability argument		
		ResultSet. HOLD_CURSORS_OVER_COMMIT	ResultSet. CLOSE_CURSORS_AT_COMMIT	
			Execution of SELECT statement with UNTIL DISCONNECT specified	Execution of other SQL statement
TRUE specified for HIRDB_CURSOR in properties	TRUE specified for HIRDB_CURSOR in URL syntax	T	T	F
	FALSE specified for HIRDB_CURSOR in URL syntax	T	T	F
FALSE specified for HIRDB_CURSOR in properties	TRUE specified for HIRDB_CURSOR in URL syntax	T	T	F
	FALSE specified for HIRDB_CURSOR in URL syntax	T	T	F
true specified for setHiRDBCursorMode		T	T	F
false specified for setHiRDBCursorMode		T	T	F

Legend:

T: The JDBC driver operates as if TRUE were specified for HIRDB_CURSOR.

F: The JDBC driver operates as if FALSE were specified for HIRDB_CURSOR.

Table 18-16 shows the holdability specifications that become effective for Statement or DatabaseMetaData objects generated by methods other than the *Table 18-15* methods.

Table 18-16: Effective holdability specifications (2/2)

HIRDB_CURSOR or setHiRDBCursorMode specification		**Specification value of setHoldability method**			**No execution of setHoldability method**	
		ResultSet. **HOLD_CURSORS_OVER_COMMIT**	**ResultSet.** **CLOSE_CURSORS_AT_COMMIT**			
			Execution of SELECT statement with UNTIL DISCONNECT specified	**Execution of other SQL statement**	**Execution of SELECT statement with UNTIL DISCONNECT specified**	**Execution of other SQL statement**
`TRUE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	T
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	F
`FALSE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	T
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	F
`true` specified for `setHiRDBCursorMode`		T	T	F	T	T
`false` specified for `setHiRDBCursorMode`		T	T	F	T	F

Legend:

T: The JDBC driver operates as if `TRUE` were specified for `HIRDB_CURSOR`.

F: The JDBC driver operates as if `FALSE` were specified for `HIRDB_CURSOR`.

For details about `HIRDB_CURSOR`, see *18.2.2 Connecting to HiRDB with the* *getConnection method.*

18.4.3 Statement interface

(1) Overview

The Statement interface provides the following principal functions:

- SQL execution
- Creation of a result set (ResultSet object) as a retrieval result
- Return of the number of updated rows as an updating result
- Specification of the maximum number of rows to be retrieved
- Specification of the maximum query wait time

(2) Methods

The table below lists the methods of the Statement interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-17: Statement interface methods

Subsection	Method	Function
(a)	addBatch(String sql)	Adds specified SQL statements to this Statement object's batch.
(b)	cancel()	Cancels the SQL statements executing in the corresponding object and in objects using the same connection as the corresponding object.
(c)	clearBatch()	Clears all SQL statements registered in this Statement object's batch.
(d)	clearWarnings()	Clears all warnings that have been reported for this Statement object.
(e)	close()	Closes the Statement object and any ResultSet object created from this Statement object.
(f)	execute(String sql)	Executes a specified SQL statement.
(g)	executeBatch()	Executes the SQL statements registered in a batch and returns an array of the numbers of updated rows.
(h)	executeQuery(String sql)	Executes a specified retrieval SQL statement and returns a ResultSet object as the result.

Subsection	Method	Function
(i)	executeUpdate(String sql)	Executes a specified non-retrieval SQL statement and returns the number of updated rows.
(j)	getConnection()	Returns the `Connection` object that created this `Statement` object.
(k)	getFetchDirection()	Acquires the default fetch direction for a result set that is created from this `Statement` object.
(l)	getFetchSize()	Acquires the default fetch size for a `ResultSet` object that is created from this `Statement` object.
(m)	getMaxFieldSize()	Acquires the maximum number of bytes for the character columns or binary columns of a `ResultSet` object that is created by this `Statement` object.
(n)	getMaxRows()	Acquires the maximum number of rows that can be stored in a `ResultSet` object created by this `Statement` object.
(o)	getMoreResults()	Moves to the next result set.
(p)	getQueryTimeout()	Returns the SQL execution timeout value (in seconds).
(q)	getResultSet()	Acquires the current results as a `ResultSet` object.
(r)	getResultSetConcurrency()	Acquires the parallel processing mode for a `ResultSet` object that is created from this `Statement` object.
(s)	getResultSetHoldability()	Acquires the holding facility for a `ResultSet` object that is created from this `Statement` object.
(t)	getResultSetType()	Acquires the type of the result set of a `ResultSet` object that is created from this `Statement` object.
(u)	getUpdateCount()	Returns the number of updated rows.
(v)	getWarnings()	Acquires the first warning that is reported by a call related to this `Statement` object.
(w)	setCursorName(String name)	This method ignores the specified value because the driver does not support positioned updating or deletion processing.

Subsection	Method	Function
(x)	setEscapeProcessing(boolean enable)	Enables or disables escape syntax analysis by this `Statement` object.
(y)	setFetchDirection(int direction)	Specifies the default fetch direction for a result set that is created from this `Statement` object.
(z)	setFetchSize(int rows)	Specifies the default fetch size for a `ResultSet` object that is created from this `Statement` object.
(aa)	setMaxFieldSize(int max)	Sets a maximum number of bytes for each character column or binary column in a `ResultSet` object that is created from this `Statement` object.
(ab)	setMaxRows(int max)	Specifies the maximum number of rows that can be stored in a `ResultSet` object that is created from this `Statement` object.
(ac)	setQueryTimeout(int seconds)	Specifies an SQL execution timeout value (in seconds).

(a) addBatch(String sql)

Function

Adds specified SQL statements to this `Statement` object's batch.

You can register a maximum of 2,147,483,647 SQL statements to be executed.

Format

```
public synchronized void addBatch(String sql) throws
SQLException
```

Arguments

String sql

SQL statements to be executed

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.

- close() has already been issued for the Connection object that created this Statement object.
- An attempt was made to register more than 2,147,483,647 SQL statements.
- null or a character string with a length of 0 is specified for the SQL statements.

(b) cancel()

Function

Cancels the SQL statements executing in the corresponding object and in objects using the same connection as the corresponding object.

Format

```
public void cancel() throws SQLException
```

Arguments

None.

Return value

None.

Functional detail

You can use the cancel method to cancel executing SQL statements# asynchronously.

If the corresponding Statement object is not executing any SQL statements, but another object is executing SQL statements on the same connection object, this method performs asynchronous cancellation.

When cancellation processing is performed on the HiRDB server, any ResultSet, PreparedStatement, and CallableStatement objects created prior to the cancellation are ignored regardless of the following settings:

- HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR property in the Properties argument of DriverManager.getConnection
- STATEMENT_COMMIT_BEHAVIOR in URL
- setStatementCommitBehavior of the DataSource interface
- HIRDB_CURSOR property in the Properties argument of DriverManager.getConnection
- HIRDB_CURSOR in URL
- setHiRDBCursorMode of the DataSource interface

- setHoldability method of the Connection interface
- resultSetHoldability argument in the createStatement and prepareStatement methods of the Connection interface
- SQL statement (until disconnect specification)

If the corresponding Statement object is not executing any SQL statements and no other object is executing SQL statements on the same connection object, this method does not perform cancellation processing on HiRDB.

For a connection that uses XADataSource, an asynchronous cancellation request is ignored.

#

SQL statements being processed by the server when the HiRDB server has control (this JDBC driver is waiting for a response).

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(c) clearBatch()

Function

Clears all SQL statements registered in this Statement object's batch.

Format

```
public synchronized void clearBatch() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(d) clearWarnings()

Function

Clears all warnings that have been reported for this `Statement` object.

Format

```
public synchronized void clearWarnings() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

None.

(e) close()

Function

Closes the `Statement` object and any `ResultSet` object created from this `Statement` object.

Format

```
public void close() throws SQLException
```

Arguments

None.

Return value

None.

Functional detail

This method closes the `Statement` object and any `ResultSet` object created from this `Statement` object.

If the `close` method of `Statement` results in an error while a pooling connection is being used, this method does not throw an `SQLException`.

In a pooling environment or an XA environment, if the `close` method of `Statement` results in an error because of physical disconnection from the database, thereby disabling connection pooling, ConnectionEventListener.connectionErrorOccurred() does not occur.

Exceptions

If a database access error occurs, the JDBC driver throws an SQLException.

(f) execute(String sql)

Function

Executes a specified SQL statement. You can use Statement.getResultSet and Statement.getUpdateCount to acquire the ResultSet object and the number of updated rows.

Format

```
public synchronized boolean execute(String sql) throws
SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

If a retrieval SQL statement was executed, this method returns true; if not, the method returns false.

Functional detail

This method executes the specified SQL statement. You can use Statement.getResultSet and Statement.getUpdateCount to acquire the ResultSet object and the number of updated rows.

The following table shows the return values of Statement.getResultSet and Statement.getUpdateCount after execution of this method.

Table 18-18: Relationship between the executed SQL statement and the return values of Statement.getResultSet and Statement.getUpdateCount

Type of executed SQL statement	Return value of Statement.getResultSet		Return value of Statement.getUpdateCount
	HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property setting		
	Other than TRUE	TRUE	
Retrieval SQL statement	ResultSet object obtained as the execution result		-1

Type of executed SQL statement	Return value of Statement.getResultSet		Return value of Statement.getUpdateCount
	HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property setting		
	Other than TRUE	TRUE	
Non-retrieval SQL statement	null	ResultSet object of 0 columns	0 or a greater value
SQL execution resulting in an error	null	null	-1

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.
- null or a character string with a length of 0 was specified in the sql argument.
- A database access error occurred.

(g) executeBatch()

Function

Executes the SQL statements registered in a batch and returns an array of the numbers of updated rows.

If an error occurs while the SQL statements registered in the batch are executing, or after they have all executed, the method calls Statement.clearBatch() and clears the batch registration information.

Format

```
public synchronized int[] executeBatch() throws SQLException
```

Arguments

None.

Return value

This method returns as an array the numbers of updated rows for all the executed SQL statements. The elements of the array are in the order the SQL statements were registered in the batch. If no SQL statements were registered in the batch, or the first SQL statement in the batch resulted in an error, the method returns an array containing no elements.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

The method throws a BatchUpdateException (subclass of SQLException) in the following cases:

- A retrieval SQL statement was executed in the batch.
- A database access error occurred.

(h) executeQuery(String sql)

Function

Executes a specified retrieval SQL statement and returns a ResultSet object as the result.

Format

```
public synchronized ResultSet executeQuery(String sql)
throws SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

ResultSet object for the execution result

Functional detail

- When TRUE is not specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  The method executes the specified non-retrieval SQL statement and returns the number of updated rows. For an SQL statement that has no retrieval result (such as an INSERT statement), the JDBC driver throws an SQLException.

- When TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

The method executes the specified SQL statement and returns the resulting ResultSet object.

For an SQL statement that has no retrieval result (such as an INSERT statement), the method returns a ResultSet object containing no columns. You can also use the Statement.getUpdateCount method to acquire the number of updated rows. The following table shows the return values of the executeQuery method and the Statement.getUpdateCount method that is executed after the executeQuery method.

Table 18-19: Return values of the executeQuery method and the Statement.getUpdateCount method that is executed after the executeQuery method

Executed SQL statement type		executeQuery method's return value	Statement.getUpdateCount method's return value
Retrieval SQL statement		ResultSet object for the execution result	-1

Non-retrieval SQL statement	INSERT,
	UPDATE,
	DELETE
		ResultSet object containing no columns	Number of updated rows

	Other	ResultSet object containing no columns	0
SQL statement resulting in an error		--	-1

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.
- A non-retrieval SQL statement was specified (other than when TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property).
- null or a character string with a length of 0 was specified in the sql argument.
- A database access error occurred.

(i) executeUpdate(String sql)

Function

Executes a specified non-retrieval SQL statement and returns the number of updated rows.

Format

```
public synchronized int executeUpdate(String sql) throws
SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

The following table shows the return values:

Executed SQL statement type		**HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property setting**	
		Other than TRUE	**TRUE**
Retrieval SQL statement		--	-1
Non-retrieval SQL statement	INSERT, UPDATE, DELETE	Number of updated rows	Number of updated rows
	Other	0	0

Functional detail

- When `TRUE` is not specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property

  The method executes the specified non-retrieval SQL statement and returns the number of updated rows.

  For an SQL statement that returns retrieval results (such as a `SELECT` statement), the JDBC driver throws an `SQLException`.

- When `TRUE` is specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property

  The method executes the specified SQL statement.

  If the return value is `-1`, you can use the `Statement.getResultSet` method to acquire the `ResultSet` object. The following table shows the return value of the `Statement.getResultSet` method that is executed after the `executeUpdate` method.

Table 18-20: Return value of the Statement.getResultSet method that is executed after the executeUpdate method

Executed SQL statement type		Statement.getResultSet method's return value
Retrieval SQL statement		`ResultSet` object for the execution result

Non-retrieval SQL statement	INSERT,
	UPDATE,
	DELETE
		`ResultSet` object containing no columns

	Other	`ResultSet` object containing no columns
SQL statement resulting in an error		null

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.
- A retrieval SQL statement was specified (other than when `TRUE` is specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property).
- `null` or a character string with a length of 0 was specified in the `sql` argument.
- A database access error occurred.

(j) getConnection()

Function

Returns the `Connection` object that created this `Statement` object.

Format

```
public synchronized Connection getConnection() throws
SQLException
```

Arguments

None.

Return value

`Connection` object

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(k) getFetchDirection()

Function

Acquires the default fetch direction for a result set that is created from this Statement object.

This method's return value is always ResultSet.FETCH_FORWARD because only the forward fetch direction is supported in HiRDB.

Format

```
public synchronized int getFetchDirection() throws
SQLException
```

Arguments

None.

Return value

This method always returns ResultSet.FETCH_FORWARD.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(l) getFetchSize()

Function

Acquires the default fetch size for a ResultSet object that is created from this Statement object.

Format

```
public synchronized int getFetchSize() throws SQLException
```

Arguments

None.

Return value

Default fetch size for a ResultSet object that is created from this Statement object.

Functional detail

This method acquires the default fetch size for a ResultSet object that is created from this Statement object.

If 0 is specified for setFetchSize, this method returns 0 even though the actual fetch size depends on the client environment definition. The following table shows the relationship between the fetch size and the return value.

Table 18-21: Relationship between fetch size and return value

setFetchSize setting (m)	Return value
0	0
1 ≤ *m* ≤ 4096	m

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(m) getMaxFieldSize()

Function

Acquires the maximum number of bytes for the character columns or binary columns of a ResultSet object that is created by this Statement object. Such a maximum size is applicable only to [M|N] [VAR] CHAR, BINARY, and BLOB columns. Any excess bytes are discarded.

This method returns the value set by setMaxFieldSize.

Format

```
public synchronized int getMaxFieldSize() throws
SQLException
```

Arguments

None.

Return value

Current maximum size (in bytes) for `[M|N] [VAR] CHAR`, `BINARY`, and `BLOB` columns. A value of `0` means that there is no size limit.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(n) getMaxRows()

Function

Acquires the maximum number of rows that can be stored in a `ResultSet` object created by this `Statement` object. Any excess rows are excluded without notification.

This method returns the value set by `setMaxRows`.

Format

```
public synchronized int getMaxRows() throws SQLException
```

Arguments

None.

Return value

Maximum number of rows that can be stored in a `ResultSet` object that is created by this `Statement` object (a value of `0` means that there is no limit to the number of rows).

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(o) getMoreResults()

Function

Moves to the next result set.

Format

```
public synchronized boolean getMoreResults() throws
```

```
SQLException
```

Arguments

None.

Return value

true

There is another result set.

false

There are no more result sets.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.
- A database access error occurred.

(p) getQueryTimeout()

Function

Returns the SQL execution timeout value (in seconds).

This method returns the value set by `setQueryTimeout`.

If `setQueryTimeout` has not been executed, the method returns `0`.

Format

```
public synchronized int getQueryTimeout() throws
SQLException
```

Arguments

None.

Return value

Timeout value in seconds

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(q) getResultSet()

Function

Acquires the current results as a `ResultSet` object.

Format

```
public synchronized ResultSet getResultSet() throws
SQLException
```

Arguments

None.

Return value

`ResultSet` object held by the `Statement` object (if there are no results, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(r) getResultSetConcurrency()

Function

Acquires the parallel processing mode for a `ResultSet` object that is created from this `Statement` object.

The method always returns `ResultSet.CONCUR_READ_ONLY` because the JDBC driver does not support an update cursor.

Format

```
public synchronized int getResultSetConcurrency() throws
SQLException
```

Arguments

None.

Return value

This method always returns `ResultSet.CONCUR_READ_ONLY`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(s) getResultSetHoldability()

Function

Acquires the holding facility for a `ResultSet` object that is created from this `Statement` object.

The holding facility is determined when the `Statement` object is created and cannot be changed subsequently.

Format

```
public synchronized int getResultSetHoldability() throws
SQLException
```

Arguments

None.

Return value

ResultSet.HOLD_CURSORS_OVER_COMMIT

The `ResultSet` object may be manipulated even after commit or rollback processing.

ResultSet.CLOSE_CURSORS_AT_COMMIT

Any attempt after commit or rollback processing to manipulate the `ResultSet` object by a method other than `close` will result in an `SQLException`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(t) getResultSetType()

Function

Acquires the type of the result set of a ResultSet object that is created from this Statement object.

This method returns ResultSet.TYPE_FORWARD_ONLY or ResultSet.TYPE_SCROLL_INSENSITIVE because the JDBC driver does not support an update cursor.

Format

```
public synchronized int getResultSetType() throws
SQLException
```

Arguments

None.

Return value

ResultSet.TYPE_FORWARD_ONLY

The cursor can move only forward.

ResultSet.TYPE_SCROLL_INSENSITIVE

The cursor can be scrolled, but a value change does not take effect.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(u) getUpdateCount()

Function

Returns the number of updated rows.

Format

```
public synchronized int getUpdateCount() throws SQLException
```

Arguments

None.

Return value

The following table shows the return values:

<table>
<tr><th colspan="4">Statement object's method execution status</th><th>Return value of getUpdateCount</th></tr>
<tr><td colspan="4">execute*XXX* method has not been executed.</td><td>-1</td></tr>
<tr><td rowspan="6">execute*XXX* method has been executed.</td><td colspan="3">getMoreResults method was executed after the last execute*XXX* method was executed.</td><td>-1</td></tr>
<tr><td colspan="3">The last execute*XXX* method executed resulted in an error.</td><td>-1</td></tr>
<tr><td colspan="3">executeBatch method was executed at the end.</td><td>-1</td></tr>
<tr><td rowspan="3">An execute*XXX* method other than executeBatch was executed at the end.</td><td colspan="2">A retrieval SQL statement was executed at the end.</td><td>-1</td></tr>
<tr><td rowspan="2">A non-retrieval SQL statement was executed at the end.</td><td>INSERT, UPDATE, DELETE</td><td>Number of updated rows</td></tr>
<tr><td>Other</td><td>0</td></tr>
</table>

Functional detail

This method returns the number of updated rows.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(v) getWarnings()

Function

Acquires the first warning that is reported by a call related to this Statement object. If there is more than one warning, each subsequent warning is changed to the first warning and can be acquired by calling the SQLWarning.getNextWarning method for the immediately preceding warning that was acquired.

Format

```
public synchronized SQLWarning getWarnings() throws
SQLException
```

Arguments

None.

Return value

First SQLWarning object (if there is no such SQLWarning object, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(w) setCursorName(String name)

Function

This method ignores the specified value because the driver does not support positioned updating or deletion processing.

Format

```
public synchronized void setCursorName(String name) throws
SQLException
```

Arguments

String name

Cursor name

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.

(x) setEscapeProcessing(boolean enable)

Function

Enables or disables escape syntax analysis by this Statement object.

The default is that escape syntax analysis is enabled.

Format

```
public synchronized void setEscapeProcessing(boolean enable)
throws SQLException
```

Arguments

boolean enable

Specifies `true` to enable escape syntax analysis and `false` to disable it.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.

(y) setFetchDirection(int direction)

Function

Specifies the default fetch direction for a result set that is created from this `Statement` object.

The default value is `ResultSet.FETCH_FORWARD`.

Format

```
public synchronized void setFetchDirection(int direction)
throws SQLException
```

Arguments

int direction

Only `ResultSet.FETCH_FORWARD`, which is the default fetch direction, can be specified.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.
- A value other than `ResultSet.FETCH_FORWARD` is specified for `direction`.

(z) setFetchSize(int rows)

Function

Specifies the default fetch size for a `ResultSet` object that is created from this `Statement` object. If `0` is specified, the default fetch size depends on the client environment definition.

Format

```
public synchronized void setFetchSize(int rows) throws
SQLException
```

Arguments

int rows

Number of rows to be fetched, in the range from 0 to 4096.

Return value

None.

Functional detail

This method specifies the default fetch size for a `ResultSet` object that is created from this `Statement` object. If `0` is specified, the default fetch size depends on the client environment definition.

The default value is `0`. When a nonzero value is specified, the JDBC driver uses the block transfer facility by applying the specified value as the value of the `PDBLKF` client environment definition. For details about the block transfer facility, see *4.7 Block transfer facility*.

The following table shows the relationship between this setting and the actual `PDBLKF` value that is used.

Table 18-22: Setting and actual PDBLKF value that is used

setFetchSize setting (m)	Value of PDBLKF client environment definition (n)	PDBLKF value used by block transfer facility
0	$1 \leq n \leq 4096$	*n*
	Not specified	Not specified
$1 \leq m \leq 4096$	$1 \leq n \leq 4096$	*m*
	Not specified	*m*

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `Statement` object.
- `close()` has already been issued for the `Connection` object that created this `Statement` object.
- The condition `rows` < 0, maximum number of rows (value of `getMaxRows()`) < `rows`, or 4096 (maximum value of block fetch) < `rows` is true.

(aa) setMaxFieldSize(int max)

Function

Sets a maximum number of bytes for each character column or binary column in a `ResultSet` object that is created from this `Statement` object. This maximum size is applicable to `[M|N] [VAR] CHAR`, `BINARY`, and `BLOB` columns. Any excess bytes are discarded.

The default value is `0`.

This setting has no effect on `ResultSet` objects that have already been created.

Format

```
public synchronized void setMaxFieldSize(int max) throws
SQLException
```

Arguments

int max

New maximum number of bytes. A value of `0` means that there is no size limit. If the maximum number of bytes is an odd number, that value minus 1 is used as the maximum number of bytes for `NCHAR` and `NVARCHAR` columns. As a result of this calculation, if the specified value is `1`, the maximum number of bytes will become 0 for NCHAR or NVARCHAR columns, in which case the JDBC driver will assume that there is no size limit.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.
- A value less than 0 is specified for max.

(ab) setMaxRows(int max)

Function

Specifies the maximum number of rows that can be stored in a ResultSet object that is created from this Statement object. Any excess rows are excluded without notification.

The default value is 0.

This setting has no effect on ResultSet objects that have already been created.

Format

```
public synchronized void setMaxRows(int max) throws
SQLException
```

Arguments

int max

New maximum number of rows that can be stored. A value of 0 means that there is no limit to the number of rows. If 0 is specified when the type of the result set is ResultSet.TYPE_SCROLL_INSENSITIVE, the maximum number of rows that can be stored becomes Integer.MAX_VALUE.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.

- close() has already been issued for the Connection object that created this Statement object.
- A value less than 0 is specified for max.

(ac) setQueryTimeout(int seconds)

Function

Specifies an SQL execution timeout value (in seconds).

Format

```
public synchronized void setQueryTimeout(int seconds) throws
SQLException
```

Arguments

int seconds

Timeout value in seconds

Return value

None.

Functional detail

This method specifies the maximum wait time (in seconds) for communication with the HiRDB server during SQL statement execution.

If 0 is specified, the value depends on the setting of the PDCWAITTIME client environment definition.

The default value is 0.

This method ignores the specified value if it is greater than 65535.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the Statement object.
- close() has already been issued for the Connection object that created this Statement object.
- A value less than 0 is specified for seconds.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbStatement

(4) Notes

(a) Using the block transfer facility by specifying the setFetchSize method

If the value 1 or greater is specified for the setFetchSize method, the JDBC driver uses the block transfer facility and requests the HiRDB server to transfer all at once the retrieval results for the number of rows specified in the argument. For details about the block transfer facility, see *4.7 Block transfer facility*.

Although there is no maximum specification value for the setFetchSize method, the block transfer facility can transfer only up to 4,096 rows at a time. Therefore, when a value greater than 4,096 is specified, the number of rows actually transferred at once will not exceed 4,096.

The following table shows the priorities that determine the number of rows that the JDBC driver requests the HiRDB server to transfer in a single transmission.

Table 18-23: Priorities for number of rows that the JDBC driver requests the HiRDB server to transfer in one transmission

Priority	Specification value
1	Value specified in the argument of the setFetchSize method of the ResultSet class
2	Value specified in the argument of the setFetchSize method of the Statement class
3	Value specified in the PDBLKF client environment definition

For details about the number of rows that the JDBC driver actually receives from the HiRDB server in one communication when the driver requests the number of rows indicated in *Table 18-23*, see *4.7(4) Number of rows transferred in one transmission*. However, when reading this section, replace *PDBLKF* with *number of rows requested for transfer as determined by the priorities shown in Table 18-23*, and replace *FETCH statement* with *next method of the ResultSet class*.

If the retrieval result is larger than the number of transfer rows shown in *Table 18-23*, the JDBC driver requests transfer to the HiRDB server as many times as necessary until retrieval is completed (or until all retrieval requests from the UAP are processed).

If one of the following conditions is satisfied, the number of rows that the JDBC driver receives from the HiRDB server in one transmission is 1:

- A projection column of the result set contains HiRDB BLOB type data.
- A projection column of the result set contains HiRDB BINARY type data with a defined length greater than 32,000, and the specification of the PDBINARYBLKF client environment definition is NO.

- All of the following conditions are satisfied:
  - During connection setup, LOCATOR is specified for the LONGVARBINARY_ACCESS property or the setLONGVARBINARY_Access argument of the DataSource class.
  - One of the following is specified:

    - UNTIL DISCONNECT is specified in a SELECT statement.

    - ResultSet.HOLD_CURSORS_OVER_COMMIT is specified in the resultSetHoldability argument of the createStatement or prepareStatement method of the Connection class.

    - During connection setup, TRUE is specified for the HIRDB_CURSOR setup item in the properties or the URL.

    - true is specified for the setHiRDBCursorMode argument of the DataSource class.

(b) Asynchronous cancellation by the cancel method

You can use the cancel method to execute asynchronous cancellation of SQL statements being processed by the HiRDB server. Even if the target Statement object is not executing an SQL statement, asynchronous cancellation is executed if another object is executing an SQL statement for the same connected object.

When asynchronous cancellation is executed at the HiRDB server, all PreparedStatement and ResultSet objects that were created before the asynchronous cancellation become invalid, regardless of the specification for validating or invalidating Statement and ResultSet objects after commit execution.

The following methods specify whether or not objects are to remain valid after commit execution:

- HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR property in the Properties argument of the getConnection method of the DriverManager class
- STATEMENT_COMMIT_BEHAVIOR in the URL
- setStatementCommitBehavior of the DataSource system interface
- HIRDB_CURSOR property in the Properties argument of the getConnection method of the DriverManager class
- HIRDB_CURSOR in the URL
- setHiRDBCursorMode of a DataSource-type interface
- setHoldability method of the Connection interface

- resultSetHoldability argument of the createStatement or prepareStatement method of the Connection interface
- SQL statement (with UNTIL DISCONNECT specification)

Asynchronous cancellation is not executed for the HiRDB server if the target Statement object is not executing an SQL statement, and if no other object is executing an SQL statement for that same connection object.

If XADataSource was used for the connection, an asynchronous cancellation request is not valid.

(c) Closing the ResultSet object during execution of an executeXXX method

If you execute an execute*XXX* method before the ResultSet object created by the corresponding Statement object has been closed, the JDBC driver closes the previous ResultSet object that was created. If an attempt is made to use the previously created ResultSet object to acquire search results after the execute*XXX* method has been executed, the JDBC driver will throw an SQLException. The following shows an example that results in an SQLException:

```
Statement st  = con.createStatement();
ResultSet rs1 = st.executeQuery("select * from tb1");
ResultSet rs2 = st.executeQuery("select * from tb2");
rs1.next();  // Throw SQLException
rs2.next();
```

18.4.4 PreparedStatement interface

(1) Overview

The PreparedStatement interface provides the following principal functions:

- Execution of SQL statements in which the ? parameter is specified
- Specification of the ? parameter
- Generation and return of a ResultSet object as a retrieval result
- Return of the number of updated rows as an updating result

Because the PreparedStatement interface is a subinterface of the Statement interface, it inherits all of the Statement interface functions.

(2) Methods

The table below lists the methods of the PreparedStatement interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-24: PreparedStatement interface methods

Subsection	Method	Function
(a)	addBatch()	Adds the current parameter set to this `PreparedStatement` object's batch.
(b)	clearParameters()	Clears all values from the current parameter set that is specified.
(c)	execute()	Executes the preprocessed SQL statement.
(d)	execute(String sql)	Executes a specified SQL statement.
(e)	executeQuery()	Executes the preprocessed retrieval SQL statement and returns the resulting `ResultSet` object.
(f)	executeQuery(String sql)	Executes a specified retrieval SQL statement and returns a `ResultSet` object as the result.
(g)	executeUpdate()	Executes the preprocessed non-retrieval SQL statement and returns the number of updated rows.
(h)	executeUpdate(String sql)	Executes a specified non-retrieval SQL statement and returns the number of updated rows.
(i)	setArray(int i, Array x)	Sets an `Array` object in a specified parameter.
(j)	setAsciiStream(int parameterIndex, java.io.InputStream x, int length)	Sets the value of a specified `InputStream` object as a `?` parameter value.
(k)	setBigDecimal(int parameterIndex,BigDecimal x)	Sets a specified `BigDecimal` object as a `?` parameter value.
(l)	setBinaryStream(int parameterIndex, java.io.InputStream x, int length)	Sets the value of a specified `InputStream` object as a `?` parameter value.
(m)	setBlob(int parameterIndex, Blob x)	Sets the value of a specified `Blob` object as a `?` parameter value.
(n)	setBoolean(int parameterIndex,boolean x)	Sets a specified `boolean` value as a `?` parameter value.
(o)	setByte(int parameterIndex,byte x)	Sets a specified `byte` value as a `?` parameter value.
(p)	setBytes(int parameterIndex,byte x[])	Sets a specified `byte` array as a `?` parameter value.

Subsection	Method	Function
(q)	setCharacterStream(int parameterIndex,Reader reader,int length)	Sets the value of a specified Reader object as a ? parameter value.
(r)	setDate(int parameterIndex, java.sql.Date x)	Sets a specified java.sql.Date object as a ? parameter value.
(s)	setDate(int parameterIndex, java.sql.Date x,Calendar cal)	Converts a java.sql.Date object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a ? parameter value.
(t)	setDouble(int parameterIndex,double x)	Sets a specified double value as a ? parameter value.
(u)	setFloat(int parameterIndex,float x)	Sets a specified float value as a ? parameter value.
(v)	setInt(int parameterIndex,int x)	Sets a specified int value as a ? parameter value.
(w)	setLong(int parameterIndex,long x)	Sets a specified long value as a ? parameter value.
(x)	setNull(int parameterIndex,int sqlType)	Sets the NULL value in a specified ? parameter.
(y)	setObject(int parameterIndex,Object x)	Sets the value of a specified object as a ? parameter value.
(z)	setObject(int parameterIndex,Object x,int targetSqlType)	Sets the value of a specified object as a ? parameter value.
(aa)	setObject(int parameterIndex,Object x,int targetSqlType,int scale)	Sets the value of a specified object as a ? parameter value.
(ab)	setShort(int parameterIndex,short x)	Sets a specified short value as a ? parameter value.
(ac)	setString(int parameterIndex,String x)	Sets a specified String object as a ? parameter value.
(ad)	setTime(int parameterIndex, java.sql.Time x)	Sets a specified java.sql.Time object as a ? parameter value.
(ae)	setTime(int parameterIndex, java.sql.Time x,Calendar cal)	Converts a java.sql.Time object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a ? parameter value.
(af)	setTimestamp(int parameterIndex, java.sql.Timestamp x)	Sets a specified java.sql.Timestamp object as a ? parameter value.

Subsection	Method	Function
(ag)	setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal)	Converts a `java.sql.Timestamp` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a `?` parameter value.

(a) addBatch()

Function

Adds the current parameter set to the this `PreparedStatement` object's batch.

You can register a maximum of 2,147,483,647 parameter sets.

Format

```
public synchronized void addBatch() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.
- No value is set for some `?` parameters.
- More than 2,147,483,647 items have been registered in the batch.

(b) clearParameters()

Function

Clears all values from the current parameter set that is specified.

Format

```
public synchronized void clearParameters() throws
SQLException
```

Arguments

None.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.

(c) execute()

Function

Executes the preprocessed SQL statement.

You can use `PreparedStatement.getResultSet` and `PreparedStatement.getUpdateCount` to obtain the `ResultSet` object and the number of updated rows as execution results.

Format

```
public synchronized boolean execute() throws SQLException
```

Arguments

None.

Return value

If a retrieval SQL statement was executed, this method returns `true`; if not, the method returns `false`.

Functional detail

This method executes the preprocessed SQL statement.

You can use `PreparedStatement.getResultSet` and `PreparedStatement.getUpdateCount` to obtain the `ResultSet` object and the number of updated rows. For the return values of `Statement.getResultSet` and `Statement.getUpdateCount` after method execution, see *18.4.3(2)(f) execute(String sql)*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.

- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.
- No value is set in at least one `?` parameter.
- A database access error occurred.

(d) execute(String sql)

Function

Executes a specified SQL statement. You can use `PreparedStatement.getResultSet` and `PreparedStatement.getUpdateCount` to obtain the `ResultSet` object and the number of updated rows.

Format

```
public synchronized boolean execute(String sql) throws
SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

If a retrieval SQL statement was executed, this method returns `true`; if not, the method returns `false`.

Functional detail

This method executes the specified SQL statement. You can use `PreparedStatement.getResultSet` and `PreparedStatement.getUpdateCount` to obtain the `ResultSet` object and the number of updated rows.

For the return values of `PreparedStatement.getResultSet` and `PreparedStatement.getUpdateCount` after execution of this method, see *Table 18-18 Relationship between the executed SQL statement and the return values of Statement.getResultSet and Statement.getUpdateCount*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.

- null or a character string with a length of 0 was specified in the sql argument.
- A database access error occurred.

(e) executeQuery()

Function

Executes the preprocessed retrieval SQL statement and returns the resulting ResultSet object.

Format

```
public synchronized ResultSet executeQuery() throws
SQLException
```

Arguments

None.

Return value

ResultSet object for the execution result

Functional detail

- When TRUE is not specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  This method executes the preprocessed retrieval SQL statement and returns the resulting ResultSet object. If a non-retrieval SQL statement is executed, the JDBC driver throws an SQLException.

- When TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  This method executes the preprocessed SQL statement and returns the resulting ResultSet object. If a non-retrieval SQL statement is executed, the method returns a ResultSet object containing no columns.

  You can use the Statement.getUpdateCount method to acquire the number of updated rows.

  For the return values of the executeQuery method and the Statement.getUpdateCount method that is executed after the executeQuery method, see *18.4.3(2)(h) executeQuery(String sql)*.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.

- close() has already been issued for the Connection object that created this PreparedStatement object.
- A non-retrieval SQL statement was specified (other than when TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property).
- No value is set in at least one ? parameter.
- A database access error occurred.

(f) executeQuery(String sql)

Function

Executes a specified retrieval SQL statement and returns a ResultSet object as the result.

Format

```
public synchronized ResultSet executeQuery(String sql)
throws SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

ResultSet object for the execution result

Functional detail

- When TRUE is not specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  This method executes a specified non-retrieval SQL statement and returns the number of updated rows. For an SQL statement that has no retrieval results (such as an INSERT statement), the JDBC driver throws an SQLException.

- When TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  The method executes a specified SQL statement and returns the resulting ResultSet object. If a non-retrieval SQL statement is executed, the method returns a ResultSet object containing no columns. You can use the Statement.getUpdateCount method to acquire the number of updated rows.

For the return values of the `executeQuery` method and the `Statement.getUpdateCount` method that is executed after the `executeQuery` method, see *18.4.3(2)(h) executeQuery(String sql)*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.
- A non-retrieval SQL statement was specified (other than when `TRUE` is specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property).
- `null` or a character string with a length of 0 was specified in the `sql` argument.
- A database access error occurred.

(g) executeUpdate()

Function

Executes the preprocessed non-retrieval SQL statement and returns the number of updated rows.

Format

```
public synchronized int executeUpdate() throws SQLException
```

Arguments

None.

Return value

The following table shows the return values:

Executed SQL statement type		**HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property setting**	
		Other than TRUE	**TRUE**
Retrieval SQL statement		--	-1
Non-retrieval SQL statement	INSERT, UPDATE, DELETE	Number of updated rows	Number of updated rows
	Other	0	0

Functional detail

- When TRUE is not specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  This method executes the preprocessed non-retrieval SQL statement and returns the number of updated rows.

  If a retrieval SQL statement is executed, the JDBC driver throws an SQLException.

- When TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property

  The method executes the preprocessed SQL statement.

  If the return value is -1, you can use the Statement.getResultSet method to acquire the ResultSet object. For the return value of the Statement.getResultSet method that is executed after the executeUpdate method was executed, see *18.4.3(2)(i) executeUpdate(String sql)*.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has been issued for the object.
- close() has already been issued for the Connection object that created this object.
- A retrieval SQL statement was executed (other than when TRUE is specified for the HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property).
- No value is set in at least one ? parameter.
- A database access error occurred.

(h) executeUpdate(String sql)

Function

Executes a specified non-retrieval SQL statement and returns the number of updated rows.

Format

```
public synchronized int executeUpdate(String sql) throws
SQLException
```

Arguments

String sql

SQL statement to be executed

Return value

The following table shows the return values:

Executed SQL statement type		HiRDB_for_Java_DAB_EXECUTESQL_NOCHK system property setting	
		Other than TRUE	TRUE
Retrieval SQL statement		--	-1

Non-retrieval SQL statement	INSERT,
	UPDATE,
	DELETE
		Number of updated rows	Number of updated rows

	Other	0	0

Functional detail

- When `TRUE` is not specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property

  This method executes a specified non-retrieval SQL statement and returns the number of updated rows.

  For an SQL statement that returns retrieval results (such as a `SELECT` statement), the JDBC driver throws an `SQLException`.

- When `TRUE` is specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property

  The method executes a specified SQL statement.

  If the return value is `-1`, you can use the `Statement.getResultSet` method to acquire the `ResultSet` object. For the return value of the `Statement.getResultSet` method that is executed after the `executeUpdate` method was executed, see *18.4.3(2)(i) executeUpdate(String sql)*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued for the `Connection` object that created this `PreparedStatement` object.
- A retrieval SQL statement was specified (other than when `TRUE` is specified for the `HiRDB_for_Java_DAB_EXECUTESQL_NOCHK` system property).

- `null` or a character string with a length of 0 was specified in the `sql` argument.
- A database access error occurred.

(i) setArray(int i,Array x)

Function

Sets an `Array` object in a specified parameter.

Format

```
public void setArray(int i,Array x) throws SQLException
```

Arguments

int i

`?` parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

Array x

`Array` object that is to be set in the `?` parameter.

Return value

None.

Functional detail

This method sets an `Array` object in the parameter with a specified parameter number.

If the data type of the `Object` array acquired by the `Array` object's `getArray()` does not correspond to the data type acquired by the `Array` object's `getBaseType()`, an `SQLException` occurs.

The following table shows the correspondence between the data type of the `Object` array acquired by the `Array` object's `getArray()` and the data type acquired by the `Array` object's `getBaseType()`:

Data type acquired by getBaseType()	Data type of Object array acquired by getArray()
java.sql.Types.SMALLINT	short[],java.lang.Short[]
java.sql.Types.INTEGER	int[],java.lang.Integer[]
java.sql.Types.REAL	float[],java.lang.Float[]
java.sql.Types.FLOAT	double[],java.lang.Double[]

Data type acquired by getBaseType()	Data type of Object array acquired by getArray()
java.sql.Types.Decimal	java.math.BigDecimal[]
java.sql.Types.CHAR	java.lang.String[]
java.sql.Types.VARCHAR	java.lang.String[]
java.sql.Types.DATE	java.sql.Date
java.sql.Types.TIME	java.sql.Time
java.sql.Types.TIMESTAMP	java.sql.Timestamp

The following table shows the relationships among the `i` and `x` arguments and their values:

Argument i	Argume nt x	Number of elements in Object array acquired by getArray()	Each element of Object array	Element of repetition column set in HiRDB
Number of an existing `?` parameter	!=null	0 < number of elements ≤ 30000	All elements are `null`	Repetition column whose elements are all `null`
			Other than the above	Repetition column other than the above
		Number of elements > 30000	--	SQLException
		0	--	Repetition column that contain no element
	null	--	--	Entire column is `null`
Number of an existing `?` parameter that is not a repetition column	--	--	--	SQLException
Number of a nonexistent `?` parameter	SQLException			

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified for the i argument.
- The column for the parameter number specified in the i argument is not an HiRDB repetition column.
- The data type of the Object array acquired from the Array object specified in the x argument cannot be converted to the data type of the column of the ? parameter.
- The length of the Object array acquired from the Array object specified in the x argument exceeds 30000 (the number of array elements exceeds 30000).

(j) setAsciiStream(int parameterIndex, java.io.InputStream x, int length)

Function

Sets the value of a specified InputStream object as a ? parameter value.

Format

```
public synchronized void setAsciiStream(int parameterIndex,
java.io.InputStream x, int length) throws SQLException
```

Arguments

int parameterIndex

? parameter number

java.io.InputStream x

InputStream object that contains the value to be set in the ? parameter

int length

Number of bytes to be set

Return value

None.

Functional detail

This method sets the value of a specified InputStream object as a ? parameter value.

This method does not execute the close() method on x even after input from x has been completed.

If the HiRDB data type of the ? parameter is not [M|N] [VAR] CHAR or BINARY or BLOB, the JDBC driver throws an SQLException result.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A value less than 0 was specified for length.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(k) setBigDecimal(int parameterIndex, BigDecimal x)

Function

Sets a specified BigDecimal object as a ? parameter value.

Format

```
public synchronized void setBigDecimal(int parameterIndex,
BigDecimal x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

BigDecimal x

BigDecimal object to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.

- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(l) setBinaryStream(int parameterIndex, java.io.InputStream x, int length)

Function

Sets the value of a specified `InputStream` object as a `?` parameter value.

Format

```
public synchronized void setBinaryStream(int parameterIndex,
java.io.InputStream x, int length) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

java.io.InputStream x

`InputStream` object that contains the value to be set in the `?` parameter

int length

Number of bytes to be set

Return value

None.

Functional detail

This method sets the value of a specified `InputStream` object as a `?` parameter value.

This method does not execute the `close()` method on `x` even after input from `x` has been completed.

If the HiRDB data type of the `?` parameter is not `BINARY` or `BLOB`, the JDBC driver throws an `SQLException` result.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A value less than `0` was specified for `length`.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(m) setBlob(int parameterIndex, Blob x)

Function

Sets the value of a specified `Blob` object as a `?` parameter value.

Format

```
public synchronized void setBlob(int parameterIndex, Blob x)
throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

Blob x

`Blob` object that contains the value to be set in the `?` parameter

Return value

None.

Functional detail

This method sets the value of a specified `InputStream` object as a `?` parameter value.

If the HiRDB data type of the `?` parameter is not `BINARY` or `BLOB`, the JDBC driver throws an `SQLException` result.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(n) setBoolean(int parameterIndex, boolean x)

Function

Sets a specified boolean value as a ? parameter value.

Format

```
public synchronized void setBoolean(int parameterIndex,
boolean x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

boolean x

Value to be set in the ? parameter

Return value

None.

Functional detail

This method sets a specified boolean value as a ? parameter value.

If the HiRDB data type of the ? parameter specified by parameterIndex is CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR and x is true, then the ? parameter value is true. If the HiRDB data type of the ? parameter specified by parameterIndex is CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR and x is false, then the ? parameter value is false∆ (∆: single-byte space).

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified ? parameter is the OUT parameter.

(o) setByte(int parameterIndex, byte x)

Function

Sets a specified byte value as a ? parameter value.

Format

```
public synchronized void setByte(int parameterIndex, byte x)
throws SQLException
```

Arguments

int parameterIndex

? parameter number

byte x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified ? parameter is the OUT parameter.

(p) setBytes(int parameterIndex, byte x[])

Function

Sets a specified `byte` array as a `?` parameter value.

Format

```
public synchronized void setBytes(int parameterIndex, byte
x[]) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

byte x[]

`byte` array that contains the values to be set in the `?` parameter

Return value

None.

Functional detail

This method provides only referencing without copying the `byte` array. Therefore, if you change a value in the `byte` array before executing the `execute`*XXX* method, this method will use the new value after the change.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- The HiRDB data type of the `?` parameter cannot be set by this method (the data type is not `BINARY` or `BLOB`).
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(q) setCharacterStream(int parameterIndex, Reader reader, int length)

Function

Sets the value of a specified `Reader` object as a `?` parameter value.

Format

```
public synchronized void setCharacterStream(int
parameterIndex, Reader reader, int length) throws
SQLException
```

Arguments

int parameterIndex

? parameter number

Reader reader

Reader object that contains the value to be set in the ? parameter

int length

Number of characters

Return value

None.

Functional detail

This method sets the value of a specified Reader object as a ? parameter value.

If the HiRDB data type of the ? parameter is not [M|N][VAR]CHAR or BINARY or BLOB, the JDBC driver throws an SQLException.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A value less than 0 was specified for length.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- Encoding failed.
- The specified ? parameter is the OUT parameter.

(r) setDate(int parameterIndex, java.sql.Date x)

Function

Sets a specified `java.sql.Date` object as a `?` parameter value.

Format

```
public synchronized void setDate(int parameterIndex,
java.sql.Date x) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

java.sql.Date x

`java.sql.Date` object that contains the value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(s) setDate(int parameterIndex, java.sql.Date x,Calendar cal)

Function

Converts a `java.sql.Date` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a `?` parameter value.

Format

```
public synchronized void setDate(int parameterIndex,
java.sql.Date x,Calendar cal) throws SQLException
```

Arguments

int parameterIndex

? parameter number

java.sql.Date x

java.sql.Date object that contains the value to be set in the ? parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If null is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(t) setDouble(int parameterIndex, double x)

Function

Sets a specified double value as a ? parameter value.

Format

```
public synchronized void setDouble(int parameterIndex,
double x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

double x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(u) setFloat(int parameterIndex, float x)

Function

Sets a specified float value as a ? parameter value.

Format

```
public synchronized void setFloat(int parameterIndex, float
x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

float x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(v) setInt(int parameterIndex, int x)

Function

Sets a specified `int` value as a `?` parameter value.

Format

```
public synchronized void setInt(int parameterIndex, int x)
throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

int x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.

- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(w) setLong(int parameterIndex, long x)

Function

Sets a specified long value as a ? parameter value.

Format

```
public synchronized void setLong(int parameterIndex, long x)
throws SQLException
```

Arguments

int parameterIndex

? parameter number

long x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(x) setNull(int parameterIndex,int sqlType)

Function

Sets the NULL value in a specified ? parameter.

This driver ignores the sqlType argument.

Format

```
public synchronized void setNull(int parameterIndex,int
sqlType) throws SQLException
```

Arguments

int parameterIndex

? parameter number

int sqlType

JDBC's SQL data type

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the OUT parameter.

(y) setObject(int parameterIndex, Object x)

Function

Sets the value of a specified object as a ? parameter value.

Format

```
public synchronized void setObject(int parameterIndex,
Object x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

Object x

Object that contains the value to be set in the ? parameter

Return value

None.

Functional detail

This method sets the value of a specified object as a ? parameter value.

If the type of the ? parameter specified by parameterIndex is HiRDB's CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR, x is a Boolean object, and x is true, then the ? parameter value is true. If the type of the ? parameter specified by parameterIndex is HiRDB's CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR, x is a Boolean object, and x is false, then the ? parameter value is false**Δ** (**Δ**: single-byte space).

If x is a byte array, this method provides only referencing without copying the byte array. Therefore, if you change a value in the byte array before executing the execute*XXX* method, this method uses the new value after the change.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(z) setObject(int parameterIndex, Object x, int targetSqlType)

Function

Sets the value of a specified object as a ? parameter value.

Format

```
public synchronized void setObject(int parameterIndex,
Object x, int targetSqlType) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

Object x

Object that contains the value to be set in the `?` parameter

int targetSqlType

JDBC's SQL data type

Return value

None.

Functional detail

This method sets the value of a specified object as a `?` parameter value.

If `targetSqlType` is `java.sql.Types.CHAR`, `java.sql.Types.VARCHAR`, or `java.sql.Types.LONGVARCHAR`, `x` is a `Boolean` object, and `x` is `true`, then the `?` parameter value is `1,0`.

If the `?` parameter's data type is `NCHAR` or `NVARCHAR` (which are HiRDB data types) and its value is `1,0`, the JDBC driver throws an `SQLException`.

If `x` is a `byte` array, this method provides only referencing without copying the `byte` array. Therefore, if you change a value in the `byte` array before executing the `execute`*XXX* method, this method uses the new value after the change.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- `targetSqlType` is one of the following:

  `Types.ARRAY`, `Types.CLOB`, `Types.REF`, or `Types.STRUCT`

- The specified `?` parameter is the `OUT` parameter.

(aa) setObject(int parameterIndex, Object x, int targetSqlType, int scale)

Function

Sets the value of a specified object as a `?` parameter value.

Format

```
public synchronized void setObject(int parameterIndex,
Object x, int targetSqlType, int scale) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

Object x

Object that contains the value to be set in the `?` parameter

int targetSqlType

JDBC's SQL data type

int scale

Scaling (ignored, if specified)

Return value

None.

Functional detail

This method sets the value of a specified object as a `?` parameter value.

If `targetSqlType` is `java.sql.Types.CHAR`, `java.sql.Types.VARCHAR`, or `java.sql.Types.LONGVARCHAR`, `x` is a `Boolean` object, and `x` is `true`, then the `?` parameter value is `1,0`.

If the `?` parameter's data type is `NCHAR` or `NVARCHAR` type (which are HiRDB data types) and its value is `1,0`, the JDBC driver throws an `SQLException`.

If `x` is a `byte` array, this method provides only referencing without copying the `byte` array. Therefore, if you change a value in the `byte` array before executing the `execute`*XXX* method, this method uses the new value after the change.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.

- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- targetSqlType is one of the following:

  Types.ARRAY, Types.CLOB, Types.REF, or Types.STRUCT
- The specified ? parameter is the OUT parameter.

(ab) setShort(int parameterIndex, short x)

Function

Sets a specified short value as a ? parameter value.

Format

```
public synchronized void setShort(int parameterIndex, short
x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

short x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.

- The specified ? parameter is the OUT parameter.

(ac) setString(int parameterIndex, String x)

Function

Sets a specified String object as a ? parameter value.

Format

```
public synchronized void setString(int parameterIndex,
String x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

String x

String object that contains the value to be set in the ? parameter

Return value

None.

Functional detail

This method sets in a ? parameter the String object specified in the x argument.

The following table shows the value of the ? parameter when the corresponding instance is CallableStatement and the x argument value is a character string with a length of 0:

Data type of ? parameter	Value of ? parameter
[M|N][VAR]CHAR	• When the HiRDB_for_Java_DAB_CONVERT_NULL system property is set to TRUE null • Otherwise Character string with a length of 0
BINARY or BLOB	Character string with a length of 0
Other	null

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.

- CLOSE() has already been issued to the CONNECTION object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- Encoding failed.
- The specified ? parameter is the OUT parameter.

(ad) setTime(int parameterIndex, java.sql.Time x)

Function

Sets a specified java.sql.Time object as a ? parameter value.

Format

```
public synchronized void setTime(int parameterIndex,
java.sql.Time x) throws SQLException
```

Arguments

int parameterIndex

? parameter number

java.sql.Time x

java.sql.Time object that contains the value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the PreparedStatement object.
- close() has already been issued to the Connection object that created this PreparedStatement object.
- A nonexistent ? parameter number was specified.
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified ? parameter is the OUT parameter.

(ae) setTime(int parameterIndex, java.sql.Time x,Calendar cal)

Function

Converts a `java.sql.Time` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a `?` parameter value.

Format

```
public synchronized void setTime(int parameterIndex,
java.sql.Time x,Calendar cal) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

java.sql.Time x

`java.sql.Time` object that contains the value to be set in the `?` parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If `null` is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified `?` parameter is the `OUT` parameter.

(af) setTimestamp(int parameterIndex, java.sql.Timestamp x)

Function

Sets a specified `java.sql.Timestamp` object as a `?` parameter value.

Format

```
public synchronized void setTimestamp(int parameterIndex,
java.sql.Timestamp x) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

java.sql.Timestamp x

`java.sql.Timestamp` object that contains the value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(ag) setTimestamp(int parameterIndex, java.sql.Timestamp x,Calendar cal)

Function

Converts a `java.sql.Timestamp` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a `?` parameter value.

Format

```
public synchronized void setTimestamp(int parameterIndex,
java.sql.Timestamp x,Calendar cal) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number

java.sql.Timestamp x

`java.sql.Timestamp` object that contains the value to be set in the `?` parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If `null` is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `PreparedStatement` object.
- `close()` has already been issued to the `Connection` object that created this `PreparedStatement` object.
- A nonexistent `?` parameter number was specified.
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbPreparedStatement`

(4) Notes

Because the `PreparedStatement` interface is a subinterface of the `Statement` interface, all notes for the `Statement` interface also apply to the `PreparedStatement` interface.

This section describes additional notes that apply to the `PreparedStatement` interface.

(a) ? parameter setup

- For details about whether mapping is possible with a `set`*XXX* method, see *18.8.3 Mapping when a ? parameter is set*.
- If the column number or name specified in a `set`*XXX* method does not exist, the JDBC driver throws an `SQLException`.
- If a value specified in a `set`*XXX* method exceeds the value range that can be represented by the data type of the corresponding `?` parameter, an overflow occurs, resulting in an `SQLException`. For details about the combinations of `set`*XXX* methods for which overflow can occur and the HiRDB data types, see *18.8.5 Overflow handling*.
- The values specified by a `set`*XXX* method remain effective until one of the following operations is executed:
    - The `clearParameters` method is executed for the target `PreparedStatement` object.
    - A `set`*XXX* method is executed for the target `PreparedStatement` object, and the `?` parameters to be specified are the same.
    - The `close` method is executed for the target `PreparedStatement` object.

(b) Retaining SQL preprocessing results beyond commit or rollback processing

For details about retaining SQL preprocessing results beyond commit or rollback processing, see *18.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(c) Specification values for ? parameters of HiRDB's DECIMAL type

Described below are operations that are executed when a `set`*XXX* method is used to specify a value for a `?` parameter of HiRDB's `DECIMAL` type, and when the precision and decimal scaling position of the `?` parameter do not match those of the specification value.

When the precision of the specification value is greater than the actual precision: the HiRDB driver throws an `SQLException`.

When the precision of the specification value is smaller than the actual precision: the HiRDB driver expands the precision of the specification value.

When the decimal scaling position of the specification value is greater than the actual decimal scaling position: the HiRDB driver truncates the actual decimal scaling position.

When the decimal scaling position of the specification value is smaller than the actual decimal scaling position: the HiRDB driver expands the decimal scaling position by adding zeros.

(d) Specification values for ? parameters of HiRDB's TIMESTAMP type

When a set*XXX* method is used to specify a value for a ? parameter of HiRDB's TIMESTAMP type, and the fraction-of-a-second precision of the value is greater than the fraction-of-a-second precision of the ? parameter, the JDBC driver truncates the fraction-of-a-second precision to match that of the ? parameter.

(e) Specification values for ? parameters of HiRDB's CHAR, VARCHAR, NCHAR, NVARCHAR, MCHAR, or MVARCHAR type

When a set*XXX* method is used to specify a value for a ? parameter of HiRDB's CHAR, VARCHAR, NCHAR, NVARCHAR, MCHAR, or MVARCHAR type, and when the length of the value after conversion to a character string expression is greater that the defined length of the ? parameter, the JDBC driver throws an SQLException.

(f) Objects that can be specified with setObject

The objects that can be specified for the x argument of setObject are objects of the following types:

- byte[]
- java.lang.Byte
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short
- java.lang.String
- java.math.BigDecimal
- java.sql.Blob
- java.sql.Boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.sql.Array

18.4.5 CallableStatement interface

(1) Overview

The CallableStatement interface provides the following main functions:

- Execution of stored procedures
- Setting IN and INOUT parameters (by using a set*XXX* method of the PreparedStatement class)
- Registering INOUT and OUT parameters
- Acquiring the values of INOUT and OUT parameters

The CallableStatement class inherits all the functions of the PreparedStatement and Statement classes because the CallableStatement class is a subclass of the PreparedStatement class.

(2) Methods

The table below lists the methods of the CallableStatement interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-25: CallableStatement interface methods

Subsectio n	Method	Function
(a)	getBigDecimal(int parameterIndex)	Acquires the value of a specified ? parameter as a java.math.BigDecimal object in Java programming language.
(b)	getBigDecimal(int parameterIndex,int scale)	Acquires the value of a specified ? parameter as a java.math.BigDecimal object in Java programming language, with as many decimal places as are specified in scale.
(c)	getBlob(int parameterIndex)	Acquires the value of a specified ? parameter as a java.sql.Blob object in Java programming language.
(d)	getBlob(String parameterName)	Acquires the value of a specified ? parameter as a java.sql.Blob object in Java programming language.
(e)	getBoolean(int parameterIndex)	Acquires the value of a specified ? parameter as boolean in Java programming language.
(f)	getBoolean(String parameterName)	Acquires the value of a specified ? parameter as boolean in Java programming language.

Subsectio n	Method	Function
(g)	getByte(int parameterIndex)	Acquires the value of a specified ? parameter as byte in Java programming language.
(h)	getByte(String parameterName)	Acquires the value of a specified ? parameter as byte in Java programming language.
(i)	getBytes(int parameterIndex)	Acquires the value of a specified ? parameter as a byte array in Java programming language.
(j)	getBytes(String parameterName)	Acquires the value of a specified ? parameter as a byte array in Java programming language.
(k)	getDate(int parameterIndex)	Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.
(l)	getDate(int parameterIndex, java.util.Calendar cal)	Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.
(m)	getDate(String parameterName)	Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.
(n)	getDate(String parameterName, java.util.Calendar cal)	Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.
(o)	getDouble(int parameterIndex)	Acquires the value of a specified ? parameter as double in Java programming language.
(p)	getDouble(String parameterName)	Acquires the value of a specified ? parameter as double in Java programming language.
(q)	getFloat(int parameterIndex)	Acquires the value of a specified ? parameter as float in Java programming language.
(r)	getFloat(String parameterName)	Acquires the value of a specified ? parameter as float in Java programming language.

Subsectio n	Method	Function
(s)	getInt(int parameterIndex)	Acquires the value of a specified `?` parameter as `int` in Java programming language.
(t)	getInt(String parameterName)	Acquires the value of a specified `?` parameter as `int` in Java programming language.
(u)	getLong(int parameterIndex)	Acquires the value of a specified `?` parameter as `long` in Java programming language.
(v)	getLong(String parameterName)	Acquires the value of a specified `?` parameter as `long` in Java programming language.
(w)	getObject(int parameterIndex)	Acquires the value of a specified `?` parameter as `java.lang.Object` in Java programming language.
(x)	getObject(String parameterName)	Acquires the value of a specified `?` parameter as `java.lang.Object` in Java programming language.
(y)	getShort(int parameterIndex)	Acquires the value of a specified `?` parameter as `short` in Java programming language.
(z)	getShort(String parameterName)	Acquires the value of a specified `?` parameter as `short` in Java programming language.
(aa)	getString(int parameterIndex)	Acquires the value of a specified `?` parameter as a `java.lang.String` object in Java programming language.
(ab)	getString(String parameterName)	Acquires the value of a specified `?` parameter as a `java.lang.String` object in Java programming language.
(ac)	getTime(int parameterIndex)	Acquires the value of a specified `?` parameter as a `java.sql.Time` object in Java programming language.
(ad)	getTime(int parameterIndex, java.util.Calendar cal)	Acquires the value of a specified `?` parameter as a `java.sql.Time` object in Java programming language.

Subsectio n	Method	Function
(ae)	getTime (String parameterName)	Acquires the value of a specified ? parameter as a java.sql.Time object in Java programming language.
(af)	getTime(String parameterName, java.util.Calendar cal)	Acquires the value of a specified ? parameter as a java.sql.Time object in Java programming language.
(ag)	getTimestamp(int parameterIndex)	Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.
(ah)	getTimestamp(int parameterIndex, java.util.Calendar cal)	Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.
(ai)	getTimestamp (String parameterName)	Acquires the value of specified ? parameter as a java.sql.Timestamp object in Java programming language.
(aj)	getTimestamp(String parameterName, java.util.Calendar cal)	Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.
(ak)	registerOutParameter(int parameterIndex,int sqlType)	Registers the data type of a specified OUT parameter as a specified JDBC type.
(al)	registerOutParameter(int parameterIndex,int sqlType,int scale)	Registers the data type of a specified OUT parameter as a specified JDBC type.
(am)	registerOutParameter(String parameterName,int sqlType)	Registers the data type of a specified OUT parameter as a specified JDBC type.
(an)	registerOutParameter(String parameterName,int sqlType,int scale)	Registers the data type of a specified OUT parameter as a specified JDBC type.
(ao)	setAsciiStream(String parameterName, java.io.InputStream x, int length)	Sets the value of a java.io.InputStream object as a ? parameter value with a length no more than the length specified by length.

Subsectio n	Method	Function
(ap)	setBigDecimal(String parameterName, java.math.BigDecimal x)	Sets a specified java.math.BigDecimal value as a ? parameter value.
(aq)	setBinaryStream(String parameterName, java.io.InputStream x, int length)	Acquires the value of a specified ? parameter as a java.math.BigDecimal object in Java programming language.
(ar)	setBoolean(String parameterName,boolean x)	Sets a specified boolean value as a ? parameter value.
(as)	setByte(String parameterName,byte x)	Sets a specified byte value as a ? parameter value.
(at)	setBytes(String parameterName,byte[] x)	Sets a specified byte array as a ? parameter value.
(au)	setCharacterStream(String parameterName,Reader x,int length)	Sets the value of a specified Reader object as a ? parameter value.
(av)	setDate(String parameterName, java.sql.Date x)	Sets the value of a java.sql.Date object as a ? parameter value.
(aw)	setDate(String parameterName, java.sql.Date x,Calendar cal)	Converts a java.sql.Date object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a ? parameter value.
(ax)	setDouble(String parameterName,double x)	Sets a specified double value as a ? parameter value.
(ay)	setFloat(String parameterName,float x)	Sets a specified float value as a ? parameter value.
(az)	setInt(String parameterName,int x)	Sets a specified int value as a ? parameter value.
(ba)	setLong(String parameterName,long x)	Sets a specified long value as a ? parameter value.
(bb)	setNull(String parameterName,int sqlType)	Sets the NULL value in a specified ? parameter.
(bc)	setObject(String parameterName,Object x)	Sets the value of a specified object as a ? parameter value.
(bd)	setObject(String parameterName,Object x,int targetSqlType)	Sets the value of a specified object as a ? parameter value.

Subsection	Method	Function
(be)	setObject(String parameterName,Object x,int targetSqlType,int scale)	Sets the value of a specified object as a ? parameter value.
(bf)	setShort(String parameterName,short x)	Sets a specified short value as a ? parameter value.
(bg)	setString(String parameterName,String x)	Sets a specified String object as a ? parameter value.
(bh)	setTime(String parameterName,Time x)	Sets a specified java.sql.Time object as a ? parameter value.
(bi)	setTime(String parameterName, java.sql.Time x,Calendar cal)	Converts a java.sql.Time object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a ? parameter value.
(bj)	setTimestamp(String parameterName, java.sql.Timestamp x)	Sets a specified java.sql.Timestamp object as a ? parameter value.
(bk)	setTimestamp(String parameterName, java.sql.Timestamp x,Calendar cal)	Converts a java.sql.Timestamp object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a ? parameter value.
(bl)	wasNull()	Acquires a value indicating whether the value of the last OUT or INOUT parameter read was NULL.

(a) getBigDecimal(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a java.math.BigDecimal object in Java programming language.

Format

```
public synchronized java.math.BigDecimal getBigDecimal(int
parameterIndex) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

`java.math.BigDecimal` object containing the value of the specified `?` parameter (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified `?` parameter as a `java.math.BigDecimal` object in Java programming language.

The following table shows the relationships among the HiRDB data type, the `?` parameter's value, and the return value:

HiRDB data type	? parameter's value	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*]	`BigDecimal` object containing the `?` parameter value. The value without the single-byte spaces at the beginning and end of the character string is used as the `BigDecimal` object.
	Other than the above	`SQLException` is thrown
SMALLINT	NULL	null
	Non-NULL	`BigDecimal` object containing the `?` parameter value
INTEGER	NULL	null

HiRDB data type	? parameter's value	Return value
	Non-NULL	BigDecimal object containing the ? parameter value
REAL	NULL	null
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Other than the above	BigDecimal object containing the ? parameter value
FLOAT	NULL	null
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Other than the above	BigDecimal object containing the ? parameter value
DECIMAL	NULL	null

HiRDB data type	? parameter's value	Return value
	Non-NULL	BigDecimal object containing the ? parameter value.
BOOLEAN	NULL	null
	true	BigDecimal object obtained based on BigDecimal (1)
	false	BigDecimal object obtained based on BigDecimal (0)
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(b) getBigDecimal(int parameterIndex, int scale)

Function

Acquires the value of a specified ? parameter as a java.math.BigDecimal object in Java programming language, with as many decimal places as are specified in `scale`.

Format

```
public synchronized java.math.BigDecimal getBigDecimal(int
parameterIndex, int scale) throws SQLException
```

Arguments

int parameterIndex

`?` parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

int scale

Scaling. The permitted range is ≤ `scale` ≤ 2147483647.

Return value

`java.math.BigDecimal` object value with the number of decimal places specified in the `scale` argument of the specified `?` parameter value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified `?` parameter as a `java.math.BigDecimal` object in Java programming language, with as many decimal places as are specified in `scale`.

If the `scale` argument value is smaller than the number of decimal places in the `?` parameter's value, the method discards the excess decimal places from the `?` parameter's value. If the `scale` argument value is greater than the number of decimal places in the `?` parameter's value, the method pads the `?` parameter's value with 0s so that the number of decimal places match the `scale` value.

For details about the relationships among the HiRDB data type, the `?` parameter's value, and the return value, see *Functional detail* in *(a) getBigDecimal(int parameterIndex)*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.

- A value less than 0 was specified for scale.
- The specified ? parameter's data type cannot be acquired by this method.

(c) getBlob(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a java.sql.Blob object in Java programming language.

Format

```
public synchronized java.sql.Blob getBlob(int
parameterIndex) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

java.sql.Blob object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified ? parameter as a java.sql.Blob object in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter's value, and the return value:

HiRDB data type	? parameter value	Return value
BINARY BLOB	NULL	null
	Non-NULL	java.sql.Blob object containing the ? parameter's value
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(d) getBlob(String parameterName)

Function

Acquires the value of a specified ? parameter as a java.sql.Blob object in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter's value, and the return value, see *Functional detail* in *(c) getBlob(int parameterIndex)*.

Format

```
public synchronized java.sql.Blob getBlob (String
parameterName) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is assumed to be part of the ? parameter's name.

Return value

java.sql.Blob object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).

- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(e) getBoolean(int parameterIndex)

Function

Acquires the value of a specified ? parameter as boolean in Java programming language.

Format

```
public synchronized boolean getBoolean(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns false)

Functional detail

This method acquires the value of a specified ? parameter as boolean in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M][VAR]CHAR NVARCHAR	NULL	false
	[*single-byte-space*] true (in single-byte characters; not case-sensitive) [*single-byte-space*]	true
	[*single-byte-space*] 1 (single-byte character) [*single-byte-space*]	true
	Other than the above	false
NCHAR	NULL	false

HiRDB data type	? parameter value	Return value
	Value beginning with [*single-byte-space*]`true` (in single-byte characters; not case-sensitive)	true
	Other than the above	false
SMALLINT	NULL	false
	0	false
	Non-zero value	true
INTEGER	NULL	false
	0	false
	Non-zero value	true
REAL	NULL	false
	Infinity	true
	-Inifnity	true
	NaN	true
	`0.0` or `-0.0`	false
	Other than the above	true
FLOAT	NULL	false
	Infinity	true
	-Inifnity	true
	NaN	true
	`0.0` or `-0.0`	false
	Other than the above	true
DECIMAL	NULL	false
	0[.00...0]	false
	Other than the above	true
BOOLEAN	NULL	false
	Non-NULL	`?` parameter value
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(f) getBoolean(String parameterName)

Function

Acquires the value of a specified `?` parameter as `boolean` in Java programming language.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(e) getBoolean(int parameterIndex)*.

Format

```
public synchronized boolean getBoolean (String
parameterName) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `false`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.

- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(g) getByte(int parameterIndex)

Function

Acquires the value of a specified ? parameter as byte in Java programming language.

Format

```
public synchronized byte getByte(int parameterIndex) throws
SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Functional detail

This method acquires the value of a specified ? parameter as byte in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0

HiRDB data type	? parameter value	Return value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and Byte.MIN_VALUE or greater, and Byte.MAX_VALUE or less	? parameter value converted to a byte value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and greater than Byte.MAX_VALUE or less than Byte.MIN_VALUE	SQLException is thrown
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] -Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	SQLException is thrown
	Other than the above (double value cannot be obtained)	SQLException is thrown
SMALLINT	NULL	0
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	? parameter value converted to a byte value
	Other than the above	SQLException is thrown
INTEGER	NULL	0

HiRDB data type	? parameter value	Return value
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	? parameter value converted to a byte value
	Other than the above	SQLException is thrown
REAL	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the ? parameter converted to a byte value
	Other than the above	SQLException is thrown
FLOAT	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown

HiRDB data type	? parameter value	Return value
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the ? parameter converted to a byte value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the ? parameter converted to a byte value
	Other than the above	SQLException is thrown
BOOLEAN	NULL	0
	true	1
	false	0
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.

- The specified ? parameter's data type cannot be acquired by this method.

(h) getByte(String parameterName)

Function

Acquires the value of a specified ? parameter as byte in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(g) getByte(int parameterIndex)*.

Format

```
public synchronized byte getByte (String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(i) getBytes(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a byte array in Java programming language.

Format

```
public synchronized byte[] getBytes(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

byte array containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified ? parameter as a byte array in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR BINARY BLOB	NULL	null
	Non-NULL	? parameter value converted to a byte array
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.

- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- A database access error occurred.

(j) getBytes(String parameterName)

Function

Acquires the value of a specified ? parameter as a byte array in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(i) getBytes(int parameterIndex)*.

Format

```
public synchronized byte[] getBytes(String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

byte array containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

- A database access error occurred.

(k) getDate(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.

Format

```
public synchronized java.sql.Date getDate(int
parameterIndex) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

java.sql.Date object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *date-format* [*single-byte-space*]	Value obtained by removing single-byte spaces at the beginning and end of the ? parameter value and then converting to the java.sql.Date object
	Other than the above	SQLException is thrown
DATE	NULL	null

HiRDB data type	? parameter value	Return value
	Non-NULL	? parameter value converted to a java.sql.Date object
TIMESTAMP	NULL	null
	Non-NULL	? parameter value converted to a java.sql.Date object
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Date.

(l) getDate(int parameterIndex, java.util.Calendar cal)

Function

Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language. This method creates the date value from the appropriate millisecond value obtained by using a specified calendar.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(k) getDate(int parameterIndex)*.

Format

```
public synchronized java.sql.Date getDate (int
parameterIndex, java.util.Calendar cal) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Date object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Date.

(m) getDate(String parameterName)

Function

Acquires the value of a specified ? parameter as a java.sql.Date object in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(k) getDate(int parameterIndex)*.

Format

```
public synchronized java.sql.Date getDate (String
parameterName) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

`java.sql.Date` object containing the value of the specified `?` parameter (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.
- The `?` parameter value cannot be acquired as `java.sql.Date`.

(n) getDate(String parameterName, java.util.Calendar cal)

Function

Acquires the value of a specified `?` parameter as a `java.sql.Date` object in Java programming language. This method creates the date value from the appropriate millisecond value obtained by using a specified calendar.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(k) getDate(int parameterIndex)*.

Format

```
public synchronized java.sql.Date getDate (String
parameterName, java.util.Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Date object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Date.

(o) getDouble(int parameterIndex)

Function

Acquires the value of a specified ? parameter as double in Java programming language.

Format

```
public synchronized double getDouble(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `0`)

Functional detail

This method acquires the value of a specified `?` parameter as `double` in Java programming language.

The following table shows the relationships among the HiRDB data type, the `?` parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and `-Double.MAX_VALUE` or greater, and `-Double.MAX_VALUE` or greater, and `Double.MIN_VALUE` or less, and `Double.MIN_VALUE` or greater, and `Double.MAX_VALUE` or less	`?` parameter value converted to a `double` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `Double.MAX_VALUE`	Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `-Double.MAX_VALUE`	-Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `Double.MIN_VALUE`, and greater than `0`	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `-Double.MIN_VALUE`, and less than 0	-0.0

HiRDB data type	? parameter value	Return value
	[*single-byte-space*] -Infinity [*single-byte-space*]	-Infinity
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	Infinity
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	NaN
	Other than the above (double value cannot be obtained)	SQLException is thrown
SMALLINT	NULL	0.0
	Non-NULL	? parameter value converted to a double value
INTEGER	NULL	0
	Non-NULL	? parameter value converted to a double value
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	? parameter value converted to a double value
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	? parameter value

HiRDB data type	? parameter value	Return value
DECIMAL	NULL	0
	Non-NULL	`?` parameter value converted to a `double` value
BOOLEAN	NULL	0.0
	true	1.0
	false	0.0
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(p) getDouble(String parameterName)

Function

Acquires the value of a specified `?` parameter as `double` in Java programming language.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(o) getDouble(int parameterIndex)*.

Format

```
public synchronized double getDouble (String parameterName)
```

```
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `0`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(q) getFloat(int parameterIndex)

Function

Acquires the value of a specified `?` parameter as `float` in Java programming language.

Format

```
public synchronized float getFloat(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

`?` parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `0`)

Functional detail

This method acquires the value of a specified `?` parameter as `float` in Java programming language.

The following table shows the relationships among the HiRDB data type, the `?` parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and one of the following: • `-Float.MAX_VALUE` or greater and `-Float.MIN_VALUE` or less • `Float.MIN_VALUE` or greater and `Float.MAX_VALUE` or less	`?` parameter value converted to a `float` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `Float.MAX_VALUE`	Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `-Float.MAX_VALUE`	-Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `Float.MIN_VALUE`, and greater than `0`	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `-Float.MIN_VALUE`, and less than 0	-0.0

HiRDB data type	? parameter value	Return value
	[*single-byte-space*] -Infinity [*single-byte-space*]	-Infinity
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	Infinity
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	NaN
	Other than the above (cannot be converted to a float value)	SQLException is thrown
SMALLINT	NULL	0.0
	Non-NULL	? parameter value converted to a float value
INTEGER	NULL	0.0
	Non-NULL	? parameter value converted to a float value
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	? parameter value
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	-Float.MAX_VALUE or greater and -Float.MIN_VALUE or less, or Float.MIN_VALUE or greater and Float.MAX_VALUE or less	? parameter value converted to a float value

HiRDB data type	? parameter value	Return value
	Greater than Float.MAX_VALUE	Infinity
	Less than -Float.MAX_VALUE	-Infinity
	Less than Float.MIN_VALUE and greater than 0	0.0
	Greater than -Float.MIN_VALUE and less than 0	-0.0
DECIMAL	NULL	0.0
	Non-NULL	? parameter value converted to a float value
BOOLEAN	NULL	0.0
	true	1.0
	false	0.0
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(r) getFloat(String parameterName)

Function

Acquires the value of a specified ? parameter as float in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(q) getFloat(int parameterIndex)*.

Format

```
public synchronized float getFloat (String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(s) getInt(int parameterIndex)

Function

Acquires the value of a specified ? parameter as int in Java programming language.

Format

```
public synchronized int getInt(int parameterIndex) throws
SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Functional detail

This method acquires the value of a specified ? parameter as int in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and Integer.MIN_VALUE or greater, and Integer.MAX_VALUE or less	Integer part of the ? parameter converted to an int value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than Integer.MAX_VALUE or less than Integer.MIN_VALUE	SQLException is thrown
	[*single-byte-space*] -Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	SQLException is thrown

HiRDB data type	? parameter value	Return value
	Other than the above (double value cannot be obtained)	SQLException is thrown
SMALLINT	NULL	0
	Non-NULL	? parameter value converted to an int value
INTEGER	NULL	0
	Non-NULL	? parameter value
REAL	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Integer.MIN_VALUE or greater and Integer.MAX_VALUE or less	Integer part of the ? parameter converted to an int value
	Other than the above	SQLException is thrown
FLOAT	NULL	0
	Infinity	SQLException is thrown

HiRDB data type	? parameter value	Return value
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Integer.MIN_VALUE or greater and Integer.MAX_VALUE or less	Integer part of the ? parameter converted to an int value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Integer.MIN_VALUE or greater and Integer.MAX_VALUE or less	Integer part of the ? parameter converted to an int value
	Other than the above	SQLException is thrown
BOOLEAN	NULL	0
	true	1
	false	0
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.

- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(t) getInt(String parameterName)

Function

Acquires the value of a specified `?` parameter as `int` in Java programming language.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(s) getInt(int parameterIndex)*.

Format

```
public synchronized int getInt (String parameterName) throws
SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `0`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `IN` parameter.

- The specified ? parameter's data type cannot be acquired by this method.

(u) getLong(int parameterIndex)

Function

Acquires the value of a specified ? parameter as long in Java programming language.

Format

```
public synchronized long getlong(int parameterIndex) throws
SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Functional detail

This method acquires the value of a specified ? parameter as long in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 15 characters or less, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and 15 characters or less	Integer part of the ? parameter converted to a long value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and characters or more, or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and Long.MIN_VALUE or greater, and Long.MAX_VALUE or less	Integer part of the ? parameter converted to a long value

HiRDB data type	? parameter value	Return value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], greater than `Long.MAX_VALUE` or less than `Long.MIN_VALUE`	`SQLException` is thrown
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] [+] `Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] [+|-] `NaN` [*single-byte-space*]	`SQLException` is thrown
	Other than the above (cannot be converted to a `double` value or `BigDecimal` object)	`SQLException` is thrown
SMALLINT	NULL	0
	Non-NULL	`?` parameter value converted to a `long` value
INTEGER	NULL	0
	Non-NULL	`?` parameter value converted to a `long` value
REAL	NULL	0
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown

HiRDB data type	? parameter value	Return value
	NaN	SQLException is thrown
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the ? parameter converted to a long value
	Other than the above	SQLException is thrown
FLOAT	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the ? parameter converted to a long value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Long.MIN_VALUE and greater and Long.MAX_VALUE and less	Integer part of the ? parameter converted to a long value

HiRDB data type	? parameter value	Return value
	Other than the above	`SQLException` is thrown
BINARY BLOB	NULL	0
	Data with a length of 0	0
	1 byte or more	Maximum of 8 bytes converted to a `long` value in little endian format
BOOLEAN	NULL	0
	true	1
	false	0
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(v) getLong(String parameterName)

Function

Acquires the value of a specified `?` parameter as `long` in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(u) getLong(int parameterIndex)*.

Format

```
public synchronized long getLong (String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(w) getObject(int parameterIndex)

Function

Acquires the value of a specified ? parameter as java.lang.Object in Java programming language.

Format

```
public synchronized Object getObject (int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

`?` parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

Return value

`java.lang.Object` containing the value of the specified `?` parameter (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified `?` parameter as `java.lang.Object` in Java programming language.

The following table shows the relationships among the HiRDB data type, the `?` parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR	NULL	null
	Non-NULL	`?` parameter value
SMALLINT	NULL	null
	Non-NULL	`Integer` object created by the `?` parameter value
INTEGER	NULL	null
	Non-NULL	`Integer` object created by the `?` parameter value
REAL	NULL	null
	Non-NULL	`Float` object created by the `?` parameter value
FLOAT	NULL	null
	Non-NULL	`Double` object created by the `?` parameter value
DECIMAL	NULL	null
	Non-NULL	`?` parameter value
DATE	NULL	null

HiRDB data type	? parameter value	Return value
	Non-NULL	`java.sql.Date` object created by the `?` parameter value
TIME	NULL	null
	Non-NULL	`java.sql.Time` object created by the `?` parameter value
TIMESTAMP	NULL	null
	Non-NULL	`java.sql.Timestamp` object created by the `?` parameter value
BINARY BLOB	NULL	null
	Non-NULL	`?` parameter value
BOOLEAN	NULL	null
	Non-NULL	`Boolean` object created by the `?` parameter value

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- A database access error occurred.

(x) getObject(String parameterName)

Function

Acquires the value of a specified `?` parameter as `java.lang.Object` in Java programming language.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(w) getObject(int parameterIndex)*.

Format

```
public synchronized Object getObject (String parameterName)
```

```
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

`java.lang.Object` containing the value of the specified `?` parameter (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.

(y) getShort(int parameterIndex)

Function

Acquires the value of a specified `?` parameter as `short` in Java programming language.

Format

```
public synchronized short getShort(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

`?` parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

Return value

Value of the specified `?` parameter (if the value is `NULL`, the method returns `0`)

Functional detail

This method acquires the value of a specified `?` parameter as `short` in Java programming language.

The following table shows the relationships among the HiRDB data type, the `?` parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and `Short.MIN_VALUE` or greater, and `Short.MAX_VALUE` or less	Integer part of the `?` parameter converted to a `short` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and greater than `Short.MAX_VALUE` or less than `Short.MIN_VALUE`	`SQLException` is thrown
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] `[+] Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] `[+ | -] NaN` [*single-byte-space*]	`SQLException` is thrown
	Other than the above (`double` value cannot be obtained)	`SQLException` is thrown
SMALLINT	NULL	0
	Non-NULL	`?` parameter value

HiRDB data type	? parameter value	Return value
INTEGER	NULL	0
	`Short.MIN_VALUE` or greater and `Short.MAX_VALUE` or less	`?` parameter value converted to a `short` value
	Other than the above	`SQLException` is thrown
REAL	NULL	0
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown
	NaN	`SQLException` is thrown
	`Short.MIN_VALUE` or greater and `Short.MAX_VALUE` or less	Integer part of the `?` parameter converted to a `short` value
	Other than the above	`SQLException` is thrown
FLOAT	NULL	0
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown
	NaN	`SQLException` is thrown

HiRDB data type	? parameter value	Return value
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Integer part of the ? parameter converted to a short value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Short.MIN_VALUE and greater and Short.MAX_VALUE or less	Integer part of the ? parameter converted to a short value
	Other than the above	SQLException is thrown
BOOLEAN	NULL	0
	true	1
	false	0
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.

- The specified ? parameter's data type cannot be acquired by this method.

(z) getShort(String parameterName)

Function

Acquires the value of a specified ? parameter as short in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(y) getShort(int parameterIndex)*.

Format

```
public synchronized short getShort (String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

Value of the specified ? parameter (if the value is NULL, the method returns 0)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(aa) getString(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a java.lang.String object in

Java programming language.

Format

```
public synchronized String getString(int parameterIndex)
throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

java.lang.String object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified ? parameter as a java.lang.String object in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR	NULL	null
	Non-NULL	? parameter value. If the HiRDB_for_Java_DAB_CONVERT_NULL system property is set to TRUE and the ? parameter value is a character string with a length of 0, the return value is null.
SMALLINT	NULL	null
	Non-NULL	String object with the ? parameter value in character string representation
INTEGER	NULL	null
	Non-NULL	String object with the ? parameter value in character string representation
REAL	NULL	null
	Infinity	String object of the character string "Infinity"
	-Infinity	String object of the character string "-Infinity"

HiRDB data type	? parameter value	Return value
	NaN	`String` object of the character string `"NaN"`
	Other than the above	`String` object with the `?` parameter value in character string representation
FLOAT	NULL	null
	Infinity	`String` object of the character string `"Infinity"`
	-Infinity	`String` object of the character string `"-Infinity"`
	NaN	`String` object of the character string `"NaN"`
	Other than the above	`String` object with the `?` parameter value in character string representation
DECIMAL	NULL	null
	Non-NULL	`String` object with the `?` parameter value in character string representation
DATE	NULL	null
	Non-NULL	`String` object of a character string in *yyyy*-*MM*-*DD* format acquired by `JdbConvert.convertDate()`
TIME	NULL	null
	Non-NULL	`String` object of a character string in *hh*:*mm*:*ss* format
TIMESTAMP	NULL	null
	Non-NULL	`String` object of a character string in *yyyy*-*MM*-*DD* Δ *hh*:*mm*:*ss*[.*ffffff*] format
BINARY BLOB	NULL	null
	Non-NULL	`String` object of the `?` parameter value
BOOLEAN	NULL	null
	true	`String` object of the character string `"true"`
	false	`String` object of the character string `"false"`

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.

- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

(ab) getString(String parameterName)

Function

Acquires the value of a specified ? parameter as a java.lang.String object in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(aa) getString(int parameterIndex)*.

Format

```
public synchronized String getString(String parameterName)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

java.lang.String object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.

- The specified ? parameter's data type cannot be acquired by this method.

(ac) getTime(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a `java.sql.Time` object in Java programming language.

Format

```
public synchronized java.sql.Time getTime(int
parameterIndex) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as `1` for the first parameter, `2` for the second parameter, etc.

Return value

`java.sql.Time` object containing the value of the specified ? parameter (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified ? parameter as a `java.sql.Time` object in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *time-format* [*single-byte-space*]	Value obtained by removing single-byte spaces at the beginning and end of the ? parameter value and then converting to a `java.sql.Time` object
	Other than the above	`SQLException`
TIME	NULL	null

HiRDB data type	? parameter value	Return value
	Non-NULL	? parameter value converted to a `java.sql.Time` object
TIMESTAMP	NULL	null
	Non-NULL	? parameter value converted to a `java.sql.Time` object
Other	--	`SQLException`

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent `?` parameter number was specified.
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.
- The `?` parameter value cannot be acquired as `java.sql.Time`.

(ad) getTime(int parameterIndex, java.util.Calendar cal)

Function

Acquires the value of a specified `?` parameter as a `java.sql.Time` object in Java programming language. This method creates the time value from the appropriate millisecond value obtained by using a specified calendar.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(ac) getTime(int parameterIndex)*.

Format

```
public synchronized java.sql.Time getTime (int
parameterIndex, java.util.Calendar cal) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Time object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Time.

(ae) getTime(String parameterName)

Function

Acquires the value of a specified ? parameter as a java.sql.Time object in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(ac) getTime(int parameterIndex)*.

Format

```
public synchronized java.sql.Time getTime (String
parameterName) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Return value

`java.sql.Time` object containing the value of the specified `?` parameter (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `IN` parameter.
- The specified `?` parameter's data type cannot be acquired by this method.
- The `?` parameter value cannot be acquired as `java.sql.Time`.

(af) getTime(String parameterName, java.util.Calendar cal)

Function

Acquires the value of a specified `?` parameter as a `java.sql.Time` object in Java programming language. This method creates the time value from the appropriate millisecond value obtained by using a specified calendar.

For details about the relationships among the HiRDB data type, the `?` parameter value, and the return value, see *Functional detail* in *(ac) getTime(int parameterIndex)*.

Format

```
public synchronized java.sql.Time getTime (String
parameterName, java.util.Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Time object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Time.

(ag) getTimestamp(int parameterIndex)

Function

Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.

Format

```
public synchronized java.sql.Timestamp getTimestamp (int
parameterIndex) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

Return value

java.sql.Timestamp object containing the value of the specified ? parameter

(if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.

The following table shows the relationships among the HiRDB data type, the ? parameter value, and the return value:

HiRDB data type	? parameter value	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *timestamp-format* [*single-byte-space*]	Value obtained by removing single-byte spaces at the beginning and end of the ? parameter value and converting to a java.sql.Timestamp object
	Other than the above	SQLException is thrown
DATE	NULL	null
	Non-NULL	? parameter value converted to a java.sql.Timestamp object
TIMESTAMP	NULL	null
	Non-NULL	? parameter value converted to a java.sql.Timestamp object
Other	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.

- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Timestamp.

(ah) getTimestamp(int parameterIndex, java.util.Calendar cal)

Function

Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language. This method creates the timestamp value from the appropriate millisecond value obtained by using a specified calendar.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(ag) getTimestamp(int parameterIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (int
parameterIndex, java.util.Calendar cal) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Timestamp object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.

- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Timestamp.

(ai) getTimestamp(String parameterName)

Function

Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(ag) getTimestamp(int parameterIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (String
parameterName) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Return value

java.sql.Timestamp object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.
- The ? parameter value cannot be acquired as java.sql.Timestamp.

(aj) getTimestamp(String parameterName, java.util.Calendar cal)

Function

Acquires the value of a specified ? parameter as a java.sql.Timestamp object in Java programming language. This method creates the timestamp value from the appropriate millisecond value obtained from a specified calendar.

For details about the relationships among the HiRDB data type, the ? parameter value, and the return value, see *Functional detail* in *(ag) getTimestamp(int parameterIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (String
parameterName, java.util.Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.util.Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

java.sql.Timestamp object containing the value of the specified ? parameter (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified ? parameter's data type cannot be acquired by this method.

- The ? parameter value cannot be acquired as java.sql.Timestamp.

(ak) registerOutParameter(int parameterIndex, int sqlType)

Function

Registers the data type of a specified OUT parameter as a specified JDBC type.

If this method is used to set a DECIMAL-type OUT or INOUT parameter, 0 is assumed as the scale value.

Format

```
public synchronized void registerOutParameter(int
parameterIndex, int sqlType) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

int sqlType

JDBC type to be registered

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued for the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- The specified JDBC type cannot be mapped to an HiRDB data type.

(al) registerOutParameter(int parameterIndex, int sqlType, int scale)

Function

Registers the data type of a specified OUT parameter as a specified JDBC type.

Format

```
public synchronized void registerOutParameter(int
parameterIndex, int sqlType, int scale) throws SQLException
```

Arguments

int parameterIndex

? parameter number, such as 1 for the first parameter, 2 for the second parameter, etc.

int sqlType

JDBC type to be registered

int scale

Scaling. The permitted value range is 0 ≤ scale ≤ 29.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued for the Connection object that created this CallableStatement object.
- A nonexistent ? parameter number was specified.
- The specified ? parameter is the IN parameter.
- Specified JDBC type cannot be mapped to an HiRDB data type.
- The value specified for scale is less than 0 or greater than 29.

(am)registerOutParameter(String parameterName, int sqlType)

Function

Registers the data type of a specified OUT parameter as a specified JDBC type.

If this method is used to set a DECIMAL-type OUT or INOUT parameter, 0 is assumed as the scale value.

Format

```
public synchronized void registerOutParameter(String
parameterName, int sqlType) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

int sqlType

JDBC type to be registered

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued for the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified JDBC type cannot be mapped to an HiRDB data type.

(an) registerOutParameter(String parameterName, int sqlType, int scale)

Function

Registers the data type of a specified OUT parameter as a specified JDBC type.

Format

```
public synchronized void registerOutParameter(String
parameterName, int sqlType, int scale) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

int sqlType

JDBC type to be registered

int scale

Scaling. The permitted value range is 0 ≤ scale ≤ 29.

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued for the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the IN parameter.
- The specified JDBC type cannot be mapped to an HiRDB data type.
- The value specified for scale is less than 0 or greater than 29.

(ao) setAsciiStream(String parameterName, java.io.InputStream x, int length)

Function

Sets the value of a java.io.InputStream object as a ? parameter value with a length no more than the length specified by length.

Format

```
public synchronized void setAsciiStream(String
parameterName, java.io.InputStream x, int length) throws
SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.io.InputStream x

InputStream object that contains the value to be set in the ? parameter int length

Length of the stream to be set in bytes

Return value

None.

Functional detail

This method sets the value of a specified InputStream object in a ? parameter.

After entry from x is completed, this method does not execute the close() method on x.

If the ? parameter's HiRDB data type is not [M|N] [VAR] CHAR, or BINARY, or BLOB, the JDBC driver throws an SQLException.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the OUT parameter.
- A value less than 0 was specified for length.
- The ? parameter's data type cannot be set by this method.
- The specified value is outside the range of data types for the ? parameter or is in a format that cannot be converted.
- Data loading from the stream failed.

(ap) setBigDecimal(String parameterName, java.math.BigDecimal x)

Function

Sets a specified java.math.BigDecimal value as a ? parameter value.

Format

```
public synchronized void setBigDecimal(String parameterName,
java.math.BigDecimal x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

java.math.BigDecimal x

`BigDecimal` object that contains the value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `OUT` parameter.
- The specified data cannot be converted to the data type of the table definition because its integer part is longer than the integer part for the table definition.
- The `?` parameter's data type cannot be set by this method.
- The specified value is outside the range of data types for the `?` parameter or is in a format that cannot be converted.

(aq) setBinaryStream(String parameterName, java.io.InputStream x, int length)

Function

Acquires the value of a specified ? parameter as a `java.math.BigDecimal object` in Java programming language.

Format

```
public synchronized void setBinaryStream(String
parameterName, java.io.InputStream x, int length) throws
SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.io.InputStream x

InputStream object that contains the value to be set in the ? parameter

int length

Length of the stream to be set in bytes

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified ? parameter is the OUT parameter.
- A value less than 0 was specified for length.
- The ? parameter's data type cannot be set by this method.
- The specified value is outside the range of data types for the ? parameter or is in a format that cannot be converted.
- Data loading from the stream has failed.

(ar) setBoolean(String parameterName, boolean x)

Function

Sets a specified boolean value as a ? parameter value.

If the HiRDB data type of the specified ? parameter is CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR and x is true, then the ? parameter value is true. If the HiRDB data type of the specified ? parameter is CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR and x is false, then the ? parameter value is false∆ (∆: single-byte space).

Format

```
public synchronized void setBoolean(String parameterName,
boolean x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

boolean x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified ? parameter is the OUT parameter.

(as) setByte(String parameterName, byte x)

Function

Sets a specified byte value as a ? parameter value.

Format

```
public synchronized void setByte(String parameterName, byte
x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

byte x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified `?` parameter is the `OUT` parameter.

(at) setBytes(String parameterName,byte[] x)

Function

Sets a specified `byte` array as a `?` parameter value.

Format

```
public synchronized void setBytes(String parameterName,
byte[] x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

byte[] x

byte array that contains the values to be set in the ? parameter

Return value

None.

Functional detail

If the ? parameter's HiRDB data type is neither BINARY nor BLOB, this method results in an SQLException.

This method does not reference the settings in the byte array; instead, they are referenced when the execute*XXX*() method is executed. Therefore, if settings in the byte array are changed between when this method executes and when the execute*XXX*() method executes, the new settings become the ? parameter values.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the ? parameter.
- A specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(au) setCharacterStream(String parameterName, Reader x, int length)

Function

Sets the value of a specified Reader object as a ? parameter value.

If the ? parameter's HiRDB data type is not [M|N][VAR]CHAR, or BINARY, or BLOB, the JDBC driver throws an SQLException.

Format

```
public synchronized void setCharacterStream(String
parameterName, Reader x, int length) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Reader x

`Reader` object that contains the value to be set in the `?` parameter

int length

Number of characters

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A value less than `0` was specified for `length`.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- Encoding failed.
- The specified `?` parameter is the `OUT` parameter.

(av) setDate(String parameterName, java.sql.Date x)

Function

Sets the value of a `java.sql.Date` object as a `?` parameter value

Format

```
public synchronized void setDate(String parameterName,
java.sql.Date x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

java.sql.Date x

`java.sql.Date` object that contains the value to be set in the `?` parameter

Return value

None.

Functional detail

This method converts a `java.sql.Date` object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a `?` parameter value.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(aw)setDate(String parameterName, java.sql.Date x,Calendar cal)

Function

Converts a `java.sql.Date` object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a `?` parameter value.

Format

```
public synchronized void setDate(String parameterName,
java.sql.Date x,Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

java.sql.Date x

`java.sql.Date` object that contains the value to be set in the `?` parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If `null` is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(ax) setDouble(String parameterName, double x)

Function

Sets a specified `double` value as a `?` parameter value.

Format

```
public synchronized void setDouble(String parameterName,
double x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

double x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(ay) setFloat(String parameterName, float x)

Function

Sets a specified `float` value as a `?` parameter value.

Format

```
public synchronized void setFloat(String parameterName,
float x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

float x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(az) setInt(String parameterName, int x)

Function

Sets a specified `int` value as a `?` parameter value.

Format

```
public synchronized void setInt(String parameterName, int x)
throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

int x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(ba) setLong(String parameterName, long x)

Function

Sets a specified `long` value as a `?` parameter value.

Format

```
public synchronized void setLong(String parameterName, long
x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

long x

Value to be set in the ? parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(bb)setNull(String parameterName,int sqlType)

Function

Sets the NULL value in a specified ? parameter.

This driver ignores the sqlType argument.

Format

```
public synchronized void setNull(String parameterName,int
sqlType) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

int sqlType

JDBC's SQL data type

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- The specified `?` parameter is the `OUT` parameter.

(bc) setObject(String parameterName, Object x)

Function

Sets the value of a specified object as a `?` parameter value.

Format

```
public synchronized void setObject(String parameterName,
Object x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

Object x

Object that contains the value to be set in the `?` parameter

Return value

None.

Functional detail

This method sets the value of a specified object as a `?` parameter value.

If the type of the `?` parameter specified by `parameterName` is HiRDB's `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, or `NVARHAR`, `x` is a `Boolean` object, and `x` is `true`, then the `?` parameter value is `true`. If the type of the `?` parameter specified by parameterName is HiRDB's CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, or NVARHAR, x is a Boolean object, and x is false, then the ? parameter value is false∆ (∆: single-byte space).

If x is a byte array, this method does not reference settings in the byte array; instead, the byte array settings are referenced when an execute*XXX*() method is executed. Therefore, if settings in the byte array are changed between when this method executes and when the execute*XXX*() method executes, the new settings become the ? parameter values.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(bd)setObject(String parameterName, Object x, int targetSqlType)

Function

Sets the value of a specified object as a ? parameter value.

Format

```
public synchronized void setObject(String parameterName,
Object x, int targetSqlType) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Object x

Object that contains the value to be set in the `?` parameter

int targetSqlType

JDBC's SQL data type

Return value

None.

Functional detail

This method sets the value of a specified object as a `?` parameter value.

If `targetSqlType` is `java.sql.Types.CHAR`, `java.sql.Types.VARCHAR`, or `java.sql.Types.LONGVARCHAR`, `x` is a `Boolean` object, and `x` is `true`, then the `?` parameter value is `1,0`.

If the `?` parameter's data type is `NCHAR` or `NVARCHAR` type (which are HiRDB data types) and its value is `1,0`, the JDBC driver throws an `SQLException`.

If `x` is a `byte` array, this method does not reference settings in the `byte` array; instead, the `byte` array's settings are referenced when an `execute`*XXX*`()` method is executed. Therefore, if settings in the `byte` array are changed between when this method executes and when the `execute`*XXX*`()` method executes, the new settings become the `?` parameter's values.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- `targetSqlType` is one of the following:

  `Types.ARRAY`, `Types.CLOB`, `Types.REF`, or `Types.STRUCT`
- The specified `?` parameter is the `OUT` parameter.

(be) setObject(String parameterName, Object x, int targetSqlType, int scale)

Function

Sets the value of a specified object as a ? parameter value.

Format

```
public synchronized void setObject(String parameterName,
Object x, int targetSqlType, int scale) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

Object x

Object that contains the value to be set in the ? parameter

int targetSqlType

JDBC's SQL data type

int scale

Scaling (ignored, if specified)

Return value

None.

Functional detail

This method sets the value of a specified object as a ? parameter value.

If targetSqlType is java.sql.Types.CHAR, java.sql.Types.VARCHAR, or java.sql.Types.LONGVARCHAR, x is a Boolean object, and x is true, then the ? parameter value is 1,0.

If the ? parameter's data type is NCHAR or NVARCHAR (which are HiRDB data types) and its value is 1,0, the JDBC driver throws an SQLException.

If x is a byte array, this method does not reference settings in the byte array; instead the byte array's settings are referenced when an execute*XXX*() method is executed. Therefore, if settings in the byte array are changed between when this method executes and when the execute*XXX*() method executes, the new settings become the ? parameter's values.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- `targetSqlType` is one of the following:

  `Types.ARRAY`, `Types.CLOB`, `Types.REF`, or `Types.STRUCT`
- The specified `?` parameter is the `OUT` parameter.

(bf) setShort(String parameterName, short x)

Function

Sets a specified `short` value as a `?` parameter value.

Format

```
public synchronized void setShort(String parameterName,
short x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

short x

Value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified `?` parameter is the `OUT` parameter.

(bg)setString(String parameterName, String x)

Function

Sets a specified `String` object as a `?` parameter value.

Format

```
public synchronized void setString(String parameterName,
String x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

String x

`String` object that contains the value to be set in the `?` parameter

Return value

None.

Functional detail

This method sets in the `?` parameter the `String` object specified in the `x` argument.

The following table shows the value of the `?` parameter when the `x` argument's value is a character string with a length of 0:

Data type of ? parameter	Value of ? parameter
[M|N][VAR]CHAR	• When the HiRDB_for_Java_DAB_CONVERT_NULL system property is set to TRUE null • Otherwise Character string with a length of 0
BINARY or BLOB	Character string with a length of 0
Other	null

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- Encoding failed.
- The specified ? parameter is the OUT parameter.

(bh)setTime(String parameterName, Time x)

Function

Sets a specified java.sql.Time object as a ? parameter value.

Format

```
public synchronized void setTime(String parameterName, Time
x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.sql.Time x

`java.sql.Time` object that contains the value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified `?` parameter is the `OUT` parameter.

(bi) setTime(String parameterName, java.sql.Time x,Calendar cal)

Function

Converts a `java.sql.Time` object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a `?` parameter value.

Format

```
public synchronized void setTime(String parameterName,
java.sql.Time x,Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

java.sql.Time x

`java.sql.Time` object that contains the value to be set in the `?` parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If `null` is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified `?` parameter is the `OUT` parameter.

(bj) setTimestamp(String parameterName, java.sql.Timestamp x)

Function

Sets a specified `java.sql.Timestamp` object as a `?` parameter value.

Format

```
public synchronized void setTimestamp(String parameterName,
java.sql.Timestamp x) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in `parameterName` is assumed to be the `?` parameter's name, any double-quotation mark (`"`) contained in the character string is treated as part of the `?` parameter's name.

java.sql.Timestamp x

`java.sql.Timestamp` object that contains the value to be set in the `?` parameter

Return value

None.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued to the Connection object that created this CallableStatement object.
- A nonexistent parameter name was specified (including the case where the specified parameterName value is null or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the ? parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified ? parameter is the OUT parameter.

(bk) setTimestamp(String parameterName, java.sql.Timestamp x,Calendar cal)

Function

Converts a java.sql.Timestamp object specified in local time to the equivalent time value in a specified calendar's time zone, and then sets it as a ? parameter value.

Format

```
public synchronized void setTimestamp(String parameterName,
java.sql.Timestamp x,Calendar cal) throws SQLException
```

Arguments

String parameterName

Parameter name (which is not case sensitive). Because the entire character string in parameterName is assumed to be the ? parameter's name, any double-quotation mark (") contained in the character string is treated as part of the ? parameter's name.

java.sql.Timestamp x

java.sql.Timestamp object that contains the value to be set in the ? parameter

Calendar cal

Calendar in which the time zone for the value to be stored in the database has been set. If `null` is specified, the JavaVM default calendar's time zone is assumed.

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` has already been issued for the `CallableStatement` object.
- `close()` has already been issued to the `Connection` object that created this `CallableStatement` object.
- A nonexistent parameter name was specified (including the case where the specified `parameterName` value is `null` or a character string with a length of 0).
- This method does not support the HiRDB data type specified in the `?` parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- The specified `?` parameter is the `OUT` parameter.

(bl) wasNull()

Function

Acquires a value indicating whether the value of the last `OUT` or `INOUT` parameter read was `NULL`.

Format

```
public synchronized boolean wasNull() throws SQLException
```

Arguments

None.

Return value

If the value of the last parameter read was `NULL`, the method returns `true`; if not, the method returns `false`.

The following table shows the return values.

Table 18-26: Return values

Condition	Return value
Last parameter read was NULL.	true
Last parameter read was not NULL.	false
Before an execute-related method has executed (ResultSet for OUT parameter is null).	false
No get operation (get*XXX*) has been performed.	false
After clearParameters has executed.	Same value as before clearParameters executed
After an execute-related method has re-executed (spanning a close of ResultSet for an OUT parameter).	false

Exceptions

The JDBC driver throws an SQLException in the following cases:

- close() has already been issued for the CallableStatement object.
- close() has already been issued for the Connection object that created this CallableStatement object.

(3) Package and class names

The following are the package and class names for installing this interface:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbCallableStatement

(4) Notes

(a) Attributes of OUT parameter in stored procedures

- The JDBC standards stipulate that you must register the attributes of OUT and INOUT parameters by executing the registerOutParameter method before you execute an execute*XXX* method. However, for a Type4 JDBC driver, execution of the registerOutParameter method is optional because the driver uses the attributes of OUT and INOUT parameters that were acquired by the CALL statement preprocessing.
- The information set by the registerOutParameter method takes effect when an execute*XXX* method is executed. This means that setting information with a registerOutParameter method after execution of an execute*XXX* method will not affect the execution results. Similarly, clearing the settings with clearParameters after execution of an execute*XXX* method will not affect the execution results.

(b) Notes about using the DECIMAL type

If you use a `registerOutParameter(int parameterIndex, int sqlType)` method that does not accept decimal places to set a `DECIMAL`-type `OUT` or `INOUT` parameter, the number of digits will be assumed to be 0.

(c) Specifying a value for a DECIMAL-type ? parameter

The following should be noted when a `set`*XXX* method is used to set a value in a `?` parameter of HiRDB's `DECIMAL` type and the precision and scaling acquired by the `CALL` statement's preprocessing do not match the precision and scaling of the value:

When the value's precision is higher than the precision acquired by the `CALL` statement's preprocessing: An `SQLException` is thrown.

When the value's precision is lower than the precision acquired by the `CALL` statement's preprocessing: The value is expanded.

When the value's scaling is greater than the scaling acquired by the `CALL` statement's preprocessing: The value is truncated based on the scaling acquired by the `CALL` statement's preprocessing.

When the value's scaling is less than the scaling acquired by the `CALL` statement's preprocessing: The value is padded with 0s.

18.4.6 ResultSet interface

(1) Overview

The `ResultSet` interface provides the following principal functions:

- Movement of data within a result set in units of rows
- Return of result data
- Notification of whether the retrieval result data is the null value

(2) Methods

The table below lists the methods of the `ResultSet` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 18-27: ResultSet interface methods

Subsection	Method	Function
(a)	absolute(int row)	Moves the cursor to a specified row in this `ResultSet` object.
(b)	afterLast()	Moves the cursor to the location immediately following the last row in this `ResultSet` object.

Subsection	Method	Function
(c)	beforeFirst()	Moves the cursor to the location immediately preceding the first row in this `ResultSet` object.
(d)	clearWarnings()	Clears all warnings reported about this `ResultSet` object.
(e)	close()	Closes the database cursor that has been opened for this `ResultSet` object and releases JDBC resources.
(f)	findColumn(String columnName)	Maps a specified column name to its column number.
(g)	first()	Moves the cursor to the first row in this `ResultSet` object.
(h)	getArray(int i)	Acquires as an `Array` object the elements of the repetition column with a specified column number located in the current row of this `ResultSet` object.
(i)	getArray(String colName)	Acquires as an `Array` object the elements of the repetition column with a specified column name located in the current row of the `ResultSet` object.
(j)	getAsciiStream(int columnIndex)	Acquires as a `PrdbDataStream` object the value of a specified column in the current row of this `ResultSet` object.
(k)	getAsciiStream(String columnName)	Acquires as a `PrdbDataStream` object the value of a specified column in the current row of this `ResultSet` object.
(l)	getBigDecimal(int columnIndex)	Acquires as a `java.math.BigDecimal` object the value of a specified column in the current row of this `ResultSet` object.
(m)	getBigDecimal(int columnIndex, int scale)	Acquires the value of a specified column in the current row of this `ResultSet` object as a `java.math.BigDecimal` object with the number of decimal places specified in `scale`.

Subsection	Method	Function
(n)	getBigDecimal(String columnName)	Acquires as a java.math.BigDecimal object the value of a specified column in the current row of this ResultSet object.
(o)	getBigDecimal(String columnName, int scale)	Acquires the value of a specified column in the current row of this ResultSet object as a java.math.BigDecimal object with the number of decimal places specified in scale.
(p)	getBinaryStream(int columnIndex)	Acquires as a binary stream the value of the column with a specified column number located in the current row of this ResultSet object.
(q)	getBinaryStream(String columnName)	Acquires as a binary stream the value of a specified column in the current row of this ResultSet object.
(r)	getBlob(int i)	Acquires as a java.sql.Blob object the value of the column with a specified column number located in the current row of this ResultSet object.
(s)	getBlob(String colName)	Acquires as a java.sql.Blob object the value of a specified column in the current row of this ResultSet object
(t)	getBoolean(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as boolean in Java programming language.
(u)	getBoolean(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as boolean in Java programming language.
(v)	getByte(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as byte in Java programming language.

Subsection	Method	Function
(w)	getByte(String columnName)	Acquires the value of a specified column in the current row of this `ResultSet` object as `byte` in Java programming language.
(x)	getBytes(int columnIndex)	Acquires the value of a specified column in the current row of this `ResultSet` object as a `byte` array in Java programming language.
(y)	getBytes(String columnName)	Acquires the value of a specified column in the current row of this `ResultSet` object as a `byte` array in Java programming language.
(z)	getCharacterStream(int columnIndex)	Acquires as a `java.io.Reader` object the value of a specified column in the current row of this `ResultSet` object.
(aa)	getCharacterStream(String columnName)	Acquires as a `java.io.Reader` object the value of a specified column in the current row of this `ResultSet` object.
(ab)	getConcurrency()	Returns the parallel processing mode of this `ResultSet` object.
(ac)	getCursorName()	Acquires the name of the SQL cursor used by this `ResultSet` object.
(ad)	getDate(int columnIndex)	Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object.
(ae)	getDate(int columnIndex,Calendar cal)	Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object.
(af)	getDate(String columnName)	Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object.
(ag)	getDate(String columnName,Calendar cal)	Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object.

Subsection	Method	Function
(ah)	getDouble(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as double in Java programming language.
(ai)	getDouble(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as double in Java programming language.
(aj)	getFetchDirection()	Acquires the fetch direction of this ResultSet object.
(ak)	getFetchSize()	Acquires the fetch size of this ResultSet object.
(al)	getFloat(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as float in Java programming language.
(am)	getFloat(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as float in Java programming language.
(an)	getInt(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as int in Java programming language.
(ao)	getInt(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as int in Java programming language.
(ap)	getLong(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as long in Java programming language.
(aq)	getLong(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as long in Java programming language.
(ar)	getMetaData()	Returns ResultSetMetaData that represents this ResultSet object's meta information.

Subsection	Method	Function
(as)	getObject(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as Object in Java programming language.
(at)	getObject(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as Object in Java programming language.
(au)	getRow()	Acquires the current row number.
(av)	getShort(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as short in Java programming language.
(aw)	getShort(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as short in Java programming language.
(ax)	getStatement()	Acquires the Statement object that created this ResultSet object.
(ay)	getString(int columnIndex)	Acquires the value of a specified column in the current row of this ResultSet object as String in Java programming language.
(az)	getString(String columnName)	Acquires the value of a specified column in the current row of this ResultSet object as String in Java programming language.
(ba)	getTime(int columnIndex)	Acquires as java.sql.Time the value of a specified column in the current row of this ResultSet object.
(bb)	getTime(int columnIndex,Calendar cal)	Acquires as a java.sql.Time object the value of a specified column in the current row of this ResultSet object.
(bc)	getTime(String columnName)	Acquires as a java.sql.Time object the value of a specified column in the current row of this ResultSet object.

Subsection	Method	Function
(bd)	getTime(String columnName,Calendar cal)	Acquires as a `java.sql.Time` object the value of a specified column in the current row of this `ResultSet` object.
(be)	getTimestamp(int columnIndex)	Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.
(bf)	getTimestamp(int columnIndex, Calendar cal)	Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.
(bg)	getTimestamp(String columnName)	Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.
(bh)	getTimestamp(String columnName, Calendar cal)	Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.
(bi)	getType()	Returns the type of this `ResultSet` object.
(bj)	getWarnings()	Acquires the first warning reported by a call related to this `ResultSet` object.
(bk)	isAfterLast()	Acquires a value indicating whether the cursor is located after the last row in this `ResultSet` object.
(bl)	isBeforeFirst()	Acquires a value indicating whether the cursor is located before the first row in this `ResultSet` object.
(bm)	isFirst()	Acquires a value indicating whether the cursor is located on the first row in this `ResultSet` object.
(bn)	isLast()	Acquires a value indicating whether the cursor is located on the last row in this `ResultSet` object.

Subsection	Method	Function
(bo)	last()	Moves the cursor to the last row of this ResultSet object.
(bp)	next()	Moves the cursor from the current position to the next row.
(bq)	previous()	Moves the cursor to the immediately preceding row in this ResultSet object.
(br)	relative(int rows)	Moves the cursor by the specified number of rows in the forward or reverse direction relative to the current position.
(bs)	setFetchDirection(int direction)	Sets the default fetch direction for the result set that is created from this Statement object.
(bt)	setFetchSize(int rows)	Sets the fetch size of this ResultSet object.
(bu)	wasNull()	Reports whether the last column value acquired is the NULL value.

(a) absolute(int row)

Function

Moves the cursor to a specified row in this ResultSet object.

Format

```
public synchronized boolean absolute(int row) throws
SQLException
```

Arguments

int row

Row number to which the cursor is to be moved. A positive number indicates that the rows in the result set are to be counted from the beginning, and a negative number indicates that the rows are to be counted from the end.

Return value

If the cursor position resulting from this method call is before the first row or after the last row, the method returns false; otherwise, the method returns true.

Functional detail

This method moves the cursor to a specified row in this `ResultSet` object.

The following table shows the cursor position after this method call and the return value.

Table 18-28: Destination of absolute and the return value

Number of rows in result set[#]	Specified row value	Destination	Return value
0	Non-zero value	Remains before the first row	false
n	*n* < `row`	After the last row	false
	1 ≤ `row` ≤ *n*	row	true
	-*n* ≤ `row` ≤ -1	(*n* + 1) + `row`	true
	`row` < -*n*	Before the first row	false

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `TYPE_FORWARD_ONLY`.
- `row=0` was specified.
- `ResultSet` has become invalid due to transaction settlement.
- A database access error occurred.

(b) afterLast()

Function

Moves the cursor to the location immediately following the last row in this `ResultSet` object.

Format

```
public synchronized void afterLast() throws SQLException
```

Arguments

None.

Return value

None.

Functional detail

This method moves the cursor to the location immediately following the last row in this `ResultSet` object.

The following table shows the location to which the cursor is moved when this method is called.

Table 18-29: Destination of afterLast

Number of rows in result set#	Current row	Row after afterLast() is called
0	Before the first row	Remains before the first row
n	Before the first row	After the last row
	1 ≤ current row ≤ *n*	After the last row
	After the last row	Remains after the last row

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `TYPE_FORWARD_ONLY`.
- `ResultSet` has become invalid due to transaction settlement.

- A database access error occurred.

(c) beforeFirst()

Function

Moves the cursor to the location immediately preceding the first row in this `ResultSet` object.

Format

```
public synchronized void beforeFirst() throws SQLException
```

Arguments

None.

Return value

None.

Functional detail

This method moves the cursor to the location immediately preceding the first row in this `ResultSet` object.

The following table shows the location to which the cursor is moved when this method is called.

Table 18-30: Destination of beforeFirst

Number of rows in result set[#]	Current row	Row after beforeFirst() is called
0	Before the first row	Remains before the first row
n	Before the first row	Remains before the first row
	1 ≤ current row ≤ *n*	Before the first row
	After the last row	Before the first row

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- The type of this ResultSet object is TYPE_FORWARD_ONLY.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(d) clearWarnings()

Function

Clears all warnings reported about this ResultSet object.

Format

```
public synchronized void clearWarnings() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

If ResultSet has become invalid due to transaction settlement, the JDBC driver throws an SQLException.

(e) close()

Function

Closes the database cursor that has been opened for this ResultSet object and releases JDBC resources.

Format

```
public synchronized void close() throws SQLException
```

Arguments

None.

Return value

None.

Exceptions

If a database access error occurs, the JDBC driver throws an `SQLException`.

(f) findColumn(String columnName)

Function

Maps a specified column name to its column number.

Format

```
public synchronized int findColumn(String columnName) throws
SQLException
```

Arguments

`String columnName`

Column name (not case sensitive). Because the entire character string in `columnName` is assumed to be the column name, any double-quotation mark (`"`) contained in the character string is treated as part of the column name. If the specified `columnName` value is `null` or a character string with a length of 0, a nonexistent column error occurs.

If multiple columns have the same `columnName` value, the column with the smallest column number takes precedence. If the column name is truncated based on the HiRDB server specifications because it is too long, but the truncated column name is specified, the method treats the column name as matching the value.

Return value

Column number corresponding to the specified column name

Functional detail

This method acquires and returns the column number that corresponds to the column name specified in the `columnName` argument.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.

- The specified column does not exist.
- A database access error occurred.

(g) first()

Function

Moves the cursor to the first row in this ResultSet object.

Format

```
public synchronized boolean first() throws SQLException
```

Arguments

None.

Return value

If the number of rows in the result set is 0, the method returns false; otherwise, the method returns true.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- The type of this ResultSet object is TYPE_FORWARD_ONLY.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(h) getArray(int i)

Function

Acquires as an Array object the elements of the repetition column with a specified column number located in the current row of this ResultSet object.

Format

```
public Array getArray(int i) throws SQLException
```

Arguments

int i

Column number

Return value

This method returns an `Array` object corresponding to the specified column. If all elements in the repetition column are the null value, the method returns `null`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object is closed by this driver because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` object used to create the `Statement` object that created this `ResultSet` object is closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number is specified in the `i` argument.
- The column indicated by the column number specified in the `i` argument is not a repetition column.
- An error occurred in the JDBC driver.

(i) getArray(String colName)

Function

Acquires as an `Array` object the elements of the repetition column with a specified column name located in the current row of the `ResultSet` object.

Format

```
public Array getArray(String colName) throws SQLException
```

Arguments

String colName

Column name

Return value

`Array` object that corresponds to the specified column name (if all elements in the repetition column are the null value, the method returns `null`)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection object used to create the Statement object that created this ResultSet object is closed.
- The ResultSet object has become invalid due to transaction settlement.
- The column specified in the colName argument does not exist or is null.
- An internal JDBC driver error occurred.

(j) getAsciiStream(int columnIndex)

Function

Acquires as a PrdbDataStream object the value of a specified column in the current row of this ResultSet object.

Format

```
public synchronized InputStream getAsciiStream (int
columnIndex) throws SQLException
```

Arguments

int columnIndex

Column number

Return value

PrdbDataStream object with the column value set (if the value is NULL, the method returns null)

Functional detail

This method acquires as a PrdbDataStream object the value of a specified column in the current row of this ResultSet object. The method does not convert the value to ASCII characters.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHARBINARYBLOB	NULL	null

HiRDB data type	Retrieval result	Return value
	Other than the above	InputStream object containing the retrieval result
Other	--	SQLException is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(k) getAsciiStream(String columnName)

Function

Acquires as a PrdbDataStream object the value of a specified column in the current row of this ResultSet object. The method does not convert the value to ASCII characters.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(j) getAsciiStream(int columnIndex)*.

Format

```
public synchronized InputStream getAsciiStream (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

PrdbDataStream object with the column value set (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(l) getBigDecimal(int columnIndex)

Function

Acquires as a java.math.BigDecimal object the value of a specified column in the current row of this ResultSet object.

Format

```
public synchronized BigDecimal getBigDecimal (int
columnIndex) throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value that corresponds to the specified column number (if the value is NULL, the method returns null)

Functional detail

This method acquires as a java.math.BigDecimal object the value of a specified column in the current row of this ResultSet object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*]	`BigDecimal` object containing the retrieval results. The value without the single-byte spaces at the beginning and end of the character string is used as the `BigDecimal` object.
	Other than the above	`SQLException` is thrown
SMALLINT	NULL	null
	Other than the above	`BigDecimal` object containing the retrieval results
INTEGER	NULL	null
	Other than the above	`BigDecimal` object containing the retrieval results
REAL	NULL	null
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown
	NaN	`SQLException` is thrown

HiRDB data type	Retrieval result	Return value
	Other than the above	BigDecimal object containing the retrieval results
FLOAT	NULL	null
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Other than the above	BigDecimal object containing the retrieval results
DECIMAL	NULL	null
	Other than the above	BigDecimal object containing the retrieval results
BOOLEAN#	NULL	null
	true	BigDecimal object obtained based on BigDecimal (1)
	false	BigDecimal object obtained based on BigDecimal (0)
Other	--	SQLException is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as BigDecimal.
- An error occurred in the JDBC driver.

(m) getBigDecimal(int columnIndex,int scale)

Function

Acquires the value of a specified column in the current row of this ResultSet object as a java.math.BigDecimal object with the number of decimal places specified in scale.

Format

```
public synchronized BigDecimal getBigDecimal(int
columnIndex,int scale) throws SQLException
```

Arguments

int columnIndex

Column number

int scale

Scaling

Return value

Column value corresponding to the specified column number with the number of decimal places specified in the scale argument (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified column in the current row of this ResultSet object as a java.math.BigDecimal object with the number of decimal places specified in scale.

If the number of decimal places in the column value is greater than the scale argument value, the method truncates the value based on the scale value. If the number of decimal places in the column value is less than the scale argument value, the method pads the value with 0s.

If the scale argument value is smaller than the number of decimal places in the retrieval result, the method discards the excess decimal places from the retrieval result. If the scale argument value is greater than the number of decimal places in the retrieval result, the method pads the retrieval result with 0s so that the number of decimal places matches the scale value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(l) getBigDecimal(int columnIndex)*.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as BigDecimal.
- A value less than 0 was specified for scale.
- An error occurred in the JDBC driver.

(n) getBigDecimal(String columnName)

Function

Acquires as a java.math.BigDecimal object the value of a specified column in the current row of this ResultSet object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(l) getBigDecimal(int columnIndex)*.

Format

```
public synchronized BigDecimal getBigDecimal (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value corresponding to the specified column name (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `BigDecimal`.
- An error occurred in the JDBC driver.

(o) getBigDecimal(String columnName,int scale)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as a `java.math.BigDecimal` object with the number of decimal places specified in `scale`.

Format

```
public synchronized BigDecimal getBigDecimal(String
columnName,int scale) throws SQLException
```

Arguments

String columnName

Column name

int scale

Scaling

Return value

Column value corresponding to the specified column name with the number of decimal places specified in the `scale` argument (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as a `java.math.BigDecimal` object with the number of decimal places specified in `scale`.

If the `scale` argument value is smaller than the number of decimal places in the retrieval result, the method discards the excess decimal places from the retrieval result. If the `scale` argument value is greater than the number of decimal places in the retrieval result, the method pads the retrieval result with 0s so that the number of decimal places matches the `scale` value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(l) getBigDecimal(int columnIndex)*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `BigDecimal`.
- A value less than `0` was specified for `scale`.

- An error occurred in the JDBC driver.

(p) getBinaryStream(int columnIndex)

Function

Acquires as a binary stream the value of the column with a specified column number located in the current row of this `ResultSet` object.

Format

```
public synchronized InputStream getBinaryStream(int
columnIndex) throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Java input stream in which the column value is sent as an uninterpreted byte stream (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires as a binary stream the value of the column with a specified column number located in the current row of this `ResultSet` object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
BINARYBLOB	NULL	null
	Other than the above	`InputStream` object containing the retrieval result
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the

Statement that created this ResultSet object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(q) getBinaryStream(String columnName)

Function

Acquires as a binary stream the value of a specified column in the current row of this ResultSet object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(p) getBinaryStream(int columnIndex)*.

Format

```
public synchronized InputStream getBinaryStream (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Java input stream in which the column value is sent as an uninterpreted byte stream (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.

- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(r) getBlob(int i)

Function

Acquires as a `java.sql.Blob` object the value of the column with a specified column number located in the current row of this `ResultSet` object.

Format

```
public synchronized Blob getBlob(int i) throws SQLException
```

Arguments

int i

Column number

Return value

`Blob` object representing the value of the specified column (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires as a `java.sql.Blob` object the value of the column with a specified column number located in the current row of this `ResultSet` object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
BINARYBLOB	NULL	null
	Other than the above	`java.sql.Blob` object containing the retrieval results
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.

- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(s) getBlob(String colName)

Function

Acquires as a `java.sql.Blob` object the value of a specified column in the current row of this `ResultSet` object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(r) getBlob(int i)*.

Format

```
public synchronized Blob getBlob(String colName) throws
SQLException
```

Arguments

String colName

Column name

Return value

`Blob` object representing the value of the specified column (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.

- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(t) getBoolean(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `boolean` in Java programming language.

Format

```
public synchronized boolean getBoolean(int columnIndex)
throws SQLException
```

Arguments

int columnIndex

Column number

Return value

`true` or `false` (if the value is `NULL`, the method returns `false`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `boolean` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M][VAR]CHARNVARCHAR	NULL	false
	`[`*single-byte-space*`]` `true` (single-byte characters; not case-sensitive) `[`*single-byte-space*`]`	true
	`[`*single-byte-space*`]` `1` (single-byte character) `[`*single-byte-space*`]`	true
	Other than the above	false
NCHAR	NULL	false
	Value beginning with `[`*single-byte-space*`]` `true` (single-byte characters; not case-sensitive)	true

HiRDB data type	Retrieval result	Return value
	Other than the above	false
SMALLINT	NULL	false
	0	false
	Non-zero value	true
INTEGER	NULL	false
	0	false
	Non-zero value	true
REAL	NULL	false
	Infinity	true
	-Infinity	true
	NaN	true
	`0.0` or `-0.0`	false
	Other than the above	true
FLOAT	NULL	false
	Infinity	true
	-Infinity	true
	NaN	true
	`0.0` or `-0.0`	false
	Other than the above	true
DECIMAL	NULL	false
	0[.00...0]	false
	Other than the above	true
BOOLEAN#	NULL	false
	Non-NULL	Retrieval result
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(u) getBoolean(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as boolean in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(t) getBoolean(int columnIndex)*.

Format

```
public synchronized boolean getBoolean(String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

true or false (if the value is NULL, the method returns false)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(v) getByte(int columnIndex)

Function

Acquires the value of a specified column in the current row of this ResultSet object as byte in Java programming language.

Format

```
public synchronized byte getByte(int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified column in the current row of this ResultSet object as byte in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and Byte.MIN_VALUE or greater, and Byte.MAX_VALUE or less	Retrieval result converted to a byte value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and greater than Byte.MAX_VALUE or less than Byte.MIN_VALUE	SQLException is thrown
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] -Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	SQLException is thrown
	Other than the above (double value cannot be obtained)	SQLException is thrown
SMALLINT	NULL	0
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Retrieval result converted to a byte value
	Other than the above	SQLException is thrown
INTEGER	NULL	0

HiRDB data type	Retrieval result	Return value
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Retrieval result converted to a byte value
	Other than the above	SQLException is thrown
REAL	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the retrieval result converted to a byte value
	Other than the above	SQLException is thrown
FLOAT	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown

HiRDB data type	Retrieval result	Return value
	`Byte.MIN_VALUE` or greater and `Byte.MAX_VALUE` or less	Integer part of the retrieval result converted to a `byte` value
	Other than the above	`SQLException` is thrown
DECIMAL	NULL	0
	`Byte.MIN_VALUE` or greater and `Byte.MAX_VALUE` or less	Integer part of the retrieval result converted to a `byte` value
	Other than the above	`SQLException` is thrown
BOOLEAN#	NULL	0
	true	1
	false	0
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

#: There is `BOOLEAN`-type data when the `Resultset` was created from `DatabaseMetadata`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as byte.
- An error occurred in the JDBC driver.

(w) getByte(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as byte in Java programming language. For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(v) getByte(int columnIndex)*.

Format

```
public synchronized byte getByte(String columnName) throws
SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as byte.

- An error occurred in the JDBC driver.

(x) getBytes(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` as a `byte` array in Java programming language.

Format

```
public synchronized byte[] getBytes(int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as a `byte` array in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHARBINARYBLOB	NULL	null
	Other than the above	Retrieval result converted to a `byte` array
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.

- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(y) getBytes(String columnName)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as a `byte` array in Java programming language. The byte value indicates the row value returned by the driver.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(x) getBytes(int columnIndex)*.

Format

```
public synchronized byte[] getBytes (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.

- An error occurred in the JDBC driver.

(z) getCharacterStream(int columnIndex)

Function

Acquires as a `java.io.Reader` object the value of a specified column in the current row of this `ResultSet` object.

Format

```
public synchronized Reader getCharacterStream (int
columnIndex) throws SQLException
```

Arguments

int columnIndex

Column number

Return value

`java.io.Reader` object containing the column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires as a `java.io.Reader` object the value of a specified column in the current row of this `ResultSet` object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHARBINARYBLOB	NULL	null
	Other than the above	`Reader` object containing the retrieval result
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the

Statement that created this ResultSet object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- Encoding failed.
- An error occurred in the JDBC driver.

(aa) getCharacterStream(String columnName)

Function

Acquires as a java.io.Reader object the value of a specified column in the current row of this ResultSet object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(z) getCharacterStream(int columnIndex)*.

Format

```
public synchronized Reader getCharacterStream (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Java input stream that sends the column value as a stream of one-byte ASCII characters (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- Encoding failed.
- An error occurred in the JDBC driver.

(ab) getConcurrency()

Function

Returns the parallel processing mode of this `ResultSet` object. The method always returns `ResultSet.CONCUR_READ_ONLY` because the JDBC driver does not support an update cursor.

Format

```
public synchronized int getConcurrency() throws SQLException
```

Arguments

None.

Return value

ResultSet.CONCUR_READ_ONLY

The column value cannot be updated.

ResultSet.CONCUR_UPDATABLE

The column value can be updated.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.

(ac) getCursorName()

Function

Acquires the name of the SQL cursor used by this `ResultSet` object.

In a HiRDB system, the JDBC driver always returns `null`.

Format

```
public synchronized String getCursorName() throws
SQLException
```

Arguments

None.

Return value

null

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(ad) getDate(int columnIndex)

Function

Acquires as a java.sql.Date object the value of a specified column in the current row of this ResultSet object.

Format

```
public synchronized java.sql.Date getDate(int columnIndex)
throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is NULL, the method returns null)

Functional detail

This method acquires as a java.sql.Date object the value of a specified column in the current row of this ResultSet object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *date-format* [*single-byte-space*]	Retrieval result converted to a `java.sql.Date` object without the single-byte spaces at the beginning and end of the retrieval result
	Other than the above	`SQLException` is thrown
DATE	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Date` object
TIMESTAMP	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Date` object
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.

- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.

- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

(ae) getDate(int columnIndex, Calendar cal)

Function

Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object. This method uses a specified calendar to create the appropriate date as a millisecond value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ad) getDate(int columnIndex)*.

Format

```
public synchronized java.sql.Date getDate(int columnIndex,
Calendar cal) throws SQLException
```

Arguments

int columnIndex

Column number

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.

- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

(af) getDate(String columnName)

Function

Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object as a `java.sql.Date` object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ad) getDate(int columnIndex)*.

Format

```
public synchronized java.sql.Date getDate (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

(ag) getDate(String columnName, Calendar cal)

Function

Acquires as a `java.sql.Date` object the value of a specified column in the current row of this `ResultSet` object. This method uses a specified calendar to create the appropriate date as a millisecond value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ad) getDate(int columnIndex)*.

Format

```
public synchronized java.sql.Date getDate(String columnName,
Calendar cal) throws SQLException
```

Arguments

String columnName

Column name

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

(ah) getDouble(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `double` in Java programming language.

Format

```
public synchronized double getDouble (int columnIndex)
throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `double` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0.0
	`[`*single-byte-space*`]` *integer-in-character-string-representation* `[`*single-byte-space*`]`, `[`*single-byte-space*`]` *decimal-number-in-character-string-representation* `[`*single-byte-space*`]`, or `[`*single-byte-space*`]`*floating-point-number-in-character-string-representation* `[`*single-byte-space*`]`, and `-Double.MAX_VALUE` or greater, and `-Double.MAX_VALUE` or greater, and `Double.MIN_VALUE` or less, and `Double.MIN_VALUE` or greater, and `Double.MAX_VALUE` or less	Retrieval result converted to a `double` value
	`[`*single-byte-space*`]` *integer-in-character-string-representation* `[`*single-byte-space*`]`, `[`*single-byte-space*`]` *decimal-number-in-character-string-representation* `[`*single-byte-space*`]`, or `[`*single-byte-space*`]`*floating-point-number-in-character-string-representation* `[`*single-byte-space*`]`, and greater than `Double.MAX_VALUE`	Infinity

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and less than `-Double.MAX_VALUE`	-Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and less than `Double.MIN_VALUE`, and greater than `0`	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] , [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] , or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*] , and greater than `-Double.MIN_VALUE`, and less than 0	-0.0
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	-Infinity
	[*single-byte-space*] `[+] Infinity` [*single-byte-space*]	Infinity
	[*single-byte-space*] `[+ | -] NaN` [*single-byte-space*]	NaN
	Other than the above (`double` value cannot be obtained)	`SQLException` is thrown
SMALLINT	NULL	0.0
	Other than the above	Retrieval result converted to a `double` value
INTEGER	NULL	0
	Other than the above	Retrieval result converted to a `double` value
REAL	NULL	0.0
	Infinity	Infinity

HiRDB data type	Retrieval result	Return value
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	Retrieval result converted to a `double` value
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	Retrieval result converted to a `double` value
DECIMAL	NULL	0
	Other than the above	Retrieval result converted to a `double` value
BOOLEAN#	NULL	0.0
	true	1.0
	false	0.0
Other	--	`SQLException` is thrown

Legend:

--: Not applicable

#: There is `BOOLEAN`-type data when the `Resultset` was created from `DatabaseMetadata`.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as double.
- An error occurred in the JDBC driver.

(ai) getDouble(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as double in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ah) getDouble(int columnIndex)*.

Format

```
public synchronized double getDouble (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this

ResultSet object has been closed.

- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as double.
- An error occurred in the JDBC driver.

(aj) getFetchDirection()

Function

Acquires the fetch direction of this ResultSet object. In HiRDB, this method always returns ResultSet.FETCH_FORWARD.

Format

```
public synchronized int getFetchDirection() throws
SQLException
```

Arguments

None.

Return value

ResultSet.FETCH_FORWARD

The result set is processed in the forward direction.

ResultSet.FETCH_REVERSE

The result set is processed in the reverse direction.

ResultSet.FETCH_UNKNOWN

The direction in which the result set is processed is unknown.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(ak) getFetchSize()

Function

Acquires the fetch size of this `ResultSet` object.

The method returns the value set by `setFetchSize`.

If no value has been set by `setFetchSize`, the method returns `0`.

Format

```
public synchronized int getFetchSize() throws SQLException
```

Arguments

None.

Return value

Current fetch size for this `ResultSet` object

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.

(al) getFloat(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `float` in Java programming language.

Format

```
public synchronized float getFloat (int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `float` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0.0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and one of the following: • `-Float.MAX_VALUE` or greater and `-Float.MIN_VALUE` or less • `Float.MIN_VALUE` or greater and `Float.MAX_VALUE` or less	Retrieval result converted to a `float` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `Float.MAX_VALUE`	Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `-Float.MAX_VALUE`	-Infinity
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and less than `Float.MIN_VALUE`, and greater than `0`	0.0

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `-Float.MIN_VALUE`, and less than 0	-0.0
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	-Infinity
	[*single-byte-space*] `[+] Infinity` [*single-byte-space*]	Infinity
	[*single-byte-space*] `[+|-] NaN` [*single-byte-space*]	NaN
	Other than the above (cannot be converted to a `float` value)	`SQLException` is thrown
SMALLINT	NULL	0.0
	Other than the above	Retrieval result converted to a `float` value
INTEGER	NULL	0.0
	Other than the above	Retrieval result converted to a `float` value
REAL	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN
	Other than the above	Retrieval result
FLOAT	NULL	0.0
	Infinity	Infinity
	-Infinity	-Infinity
	NaN	NaN

HiRDB data type	Retrieval result	Return value
	-Float.MAX_VALUE or greater and -Float.MIN_VALUE or less, or Float.MIN_VALUE or greater and Float.MAX_VALUE or less	Retrieval result converted to a float value
	Greater than Float.MAX_VALUE	Infinity
	Less than -Float.MAX_VALUE	-Infinity
	Less than Float.MIN_VALUE and greater than 0	0.0
	Greater than -Float.MIN_VALUE and less than 0	-0.0
DECIMAL	NULL	0.0
	Other than the above	Retrieval result converted to a float value
BOOLEAN#	NULL	0.0
	true	1.0
	false	0.0
Other	--	SQLException is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.

- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as float.
- An error occurred in the JDBC driver.

(am)getFloat(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as float in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(al) getFloat(int columnIndex)*.

Format

```
public synchronized float getFloat (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as float.

- An error occurred in the JDBC driver.

(an) getInt(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `int` in Java programming language.

Format

```
public synchronized int getInt(int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `int` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and `Integer.MIN_VALUE` or greater, and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `Integer.MAX_VALUE` or less than `Integer.MIN_VALUE`	`SQLException` is thrown

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] `[+]` `Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] `[+|-]` `NaN` [*single-byte-space*]	`SQLException` is thrown
	Other than the above (`double` value cannot be obtained)	`SQLException` is thrown
SMALLINT	NULL	0
	Other than the above	Retrieval result converted to an `int` value
INTEGER	NULL	0
	Other than the above	Retrieval result converted to an `int` value
REAL	NULL	0
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown
	NaN	`SQLException` is thrown

HiRDB data type	Retrieval result	Return value
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	Other than the above	`SQLException` is thrown
FLOAT	NULL	0
	Infinity	`SQLException` is thrown
	-Infinity	`SQLException` is thrown
	NaN	`SQLException` is thrown
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	Other than the above	`SQLException` is thrown
DECIMAL	NULL	0
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value

HiRDB data type	Retrieval result	Return value
	Other than the above	SQLException is thrown
BOOLEAN#	NULL	0
	true	1
	false	0
Other	--	SQLException is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as int.
- An error occurred in the JDBC driver.

(ao) getInt(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as int in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(an) getInt(int columnIndex)*.

Format

```
public synchronized int getInt(String columnName) throws
SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `int`.
- An error occurred in the JDBC driver.

(ap) getLong(int columnIndex)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `long` in Java programming language.

Format

```
public synchronized double getLong (int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `long` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 15 characters or less, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and 15 characters or less	Integer part of the retrieval result converted to a `long` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], and `Long.MIN_VALUE` or greater, and `Long.MAX_VALUE` or less	Integer part of the retrieval result converted to a `long` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*] and 16 characters or more, or [*single-byte-space*] *floating-point-number-in-character-string-representation* [*single-byte-space*], greater than `Long.MAX_VALUE` or less than `Long.MIN_VALUE`	`SQLException` is thrown
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	`SQLException` is thrown
	[*single-byte-space*] `[+]Infinity` [*single-byte-space*]	`SQLException` is thrown

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	SQLException is thrown
	Other than the above (cannot be converted to a double value or BigDecimal object)	SQLException is thrown
SMALLINT	NULL	0
	Other than the above	Retrieval result converted to a long value
INTEGER	NULL	0
	Other than the above	Retrieval result converted to a long value
REAL	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the retrieval result converted to a long value
	Other than the above	SQLException is thrown

HiRDB data type	Retrieval result	Return value
FLOAT	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the retrieval result converted to a long value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Long.MIN_VALUE and greater and Long.MAX_VALUE and less	Integer part of the retrieval result converted to a long value
BINARYBLOB	NULL	0
	Data with a length of 0	0
	1 byte or more	Maximum of 8 bytes converted to a long value in little endian format
BOOLEAN#	NULL	0

HiRDB data type	Retrieval result	Return value
	true	1
	false	0
Other	--	SQLException is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as long.
- An error occurred in the JDBC driver.

(aq) getLong(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as long in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ap) getLong(int columnIndex)*.

Format

```
public synchronized double getLong (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as long.
- An error occurred in the JDBC driver.

(ar) getMetaData()

Function

Returns ResultSetMetaData that represents this ResultSet object's meta information.

Format

```
public synchronized ResultSetMetaData getMetaData() throws
SQLException
```

Arguments

None.

Return value

This method returns meta information for this ResultSet object as a ResultSetMetaData object.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(as) getObject(int columnIndex)

Function

Acquires the value of a specified column in the current row of this ResultSet object as Object in Java programming language.

Format

```
public synchronized Object getObject (int columnIndex)
throws SQLException
```

Arguments

int columnIndex

Column number

Return value

This method returns a column value as a Java object.

This Java object has the default Java object type that corresponds to the column's SQL type based on the mapping of built-in types specified in the JDBC specifications. If the value is NULL, the method returns null.

Functional detail

This method acquires the value of a specified column in the current row of this ResultSet object as Object in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	Other than the above	Retrieval result

HiRDB data type	Retrieval result	Return value
SMALLINT	NULL	null
	Other than the above	Integer object created by using the retrieval result
INTEGER	NULL	null
	Other than the above	Integer object created by using the retrieval result
REAL	NULL	null
	Other than the above	Float object created by using the retrieval result
FLOAT	NULL	null
	Other than the above	Double object created by using the retrieval result
DECIMAL	NULL	null
	Other than the above	Retrieval result
DATE	NULL	null
	Other than the above	java.sql.Date object created by using the retrieval result
TIME	NULL	null
	Other than the above	java.sql.Time object created by using the retrieval result
TIMESTAMP	NULL	null
	Other than the above	java.sql.Timestamp object created by using the retrieval result
BINARYBLOB	NULL	null
	Other than the above	Retrieval result
BOOLEAN#	NULL	null
	Non-NULL	Boolean object created by using the retrieval result

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- An error occurred in the JDBC driver.

(at) getObject(String columnName)

Function

Acquires the value of a specified column in the current row of this `ResultSet` object as `Object` in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(as) getObject(int columnIndex)*.

Format

```
public synchronized Object getObject (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

This method returns a column value as a Java object.

This Java object has the default Java object type that corresponds to the column's SQL type based on the mapping of built-in types specified in the JDBC specifications. If the value is `NULL`, the method returns `null`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the

Statement that created this ResultSet object was closed.

- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- An error occurred in the JDBC driver.

(au) getRow()

Function

Acquires the current row number, such as 1 for the first row, 2 for the second row, etc. If the row is before the first row or after the last row, the row number is 0.

If the number of retrieved rows exceeds 2147483647, the method returns 2147483647.

Format

```
public synchronized int getRow() throws SQLException
```

Arguments

None.

Return value

Current row number (if the current row number is greater than Integer.MAX_VALUE, the method returns Integer.MAX_VALUE)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(av) getShort(int columnIndex)

Function

Acquires the value of a specified column in the current row of this ResultSet object as `short` in Java programming language.

Format

```
public synchronized short getShort(int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires the value of a specified column in the current row of this `ResultSet` object as `short` in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR] CHAR	NULL	0
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and `Short.MIN_VALUE` or greater, and `Short.MAX_VALUE` or less	Integer part of the retrieval result converted to a `short` value
	[*single-byte-space*] *integer-in-character-string-representation* [*single-byte-space*], [*single-byte-space*] *decimal-number-in-character-string-representation* [*single-byte-space*], or [*single-byte-space*]*floating-point-number-in-character-string-representation* [*single-byte-space*], and greater than `Short.MAX_VALUE` or less than `Short.MIN_VALUE`	`SQLException` is thrown
	[*single-byte-space*] `-Infinity` [*single-byte-space*]	`SQLException` is thrown

HiRDB data type	Retrieval result	Return value
	[*single-byte-space*] [+] Infinity [*single-byte-space*]	SQLException is thrown
	[*single-byte-space*] [+|-] NaN [*single-byte-space*]	SQLException is thrown
	Other than the above (double value cannot be obtained)	SQLException is thrown
SMALLINT	NULL	0
	Other than the above	Retrieval result
INTEGER	NULL	0
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Retrieval result converted to a short value
	Other than the above	SQLException is thrown
REAL	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value

HiRDB data type	Retrieval result	Return value
	Other than the above	SQLException is thrown
FLOAT	NULL	0
	Infinity	SQLException is thrown
	-Infinity	SQLException is thrown
	NaN	SQLException is thrown
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value
	Other than the above	SQLException is thrown
DECIMAL	NULL	0
	Short.MIN_VALUE and greater and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value
	Other than the above	SQLException is thrown
BOOLEAN#	NULL	0
	true	1
	false	0

HiRDB data type	Retrieval result	Return value
Other	--	SQLException is thrown

Legend:

--: Not applicable

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as short.
- An error occurred in the JDBC driver.

(aw)getShort(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as short in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(av) getShort(int columnIndex)*.

Format

```
public synchronized short getShort (String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as short.
- An error occurred in the JDBC driver.

(ax) getStatement()

Function

Acquires the Statement object that created this ResultSet object.

If the result set was created by a method of DatabaseMetaData, this method returns null.

Format

```
public synchronized Statement getStatement() throws
SQLException
```

Arguments

None.

Return value

Statement object that created this ResultSet object (if the result set was created by a method of DatabaseMetaData, this method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(ay) getString(int columnIndex)

Function

Acquires the value of a specified column in the current row of this ResultSet object as String in Java programming language.

Format

```
public synchronized String getString(int columnIndex) throws
SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is NULL, the method returns null)

Functional detail

This method acquires the value of a specified column in the current row of this ResultSet object as String in Java programming language.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	Other than the above	Retrieval result
SMALLINT	NULL	null

HiRDB data type	Retrieval result	Return value
	Other than the above	`String` object containing the retrieval result in character string representation
INTEGER	NULL	null
	Other than the above	`String` object containing the retrieval result in character string representation
REAL	NULL	null
	Infinity	`String` object of the character string `"Infinity"`
	-Infinity	`String` object of the character string `"-Infinity"`
	NaN	`String` object of the character string `"NaN"`
	Other than the above	`String` object containing the retrieval result in character string representation
FLOAT	NULL	null
	Infinity	`String` object of the character string `"Infinity"`
	-Infinity	`String` object of the character string `"-Infinity"`
	NaN	`String` object of the character string `"NaN"`
	Other than the above	`String` object containing the retrieval result in character string representation
DECIMAL	NULL	null
	Other than the above	`String` object containing the retrieval result in character string representation
DATE	NULL	null
	Other than the above	`String` object of a character string in *yyyy*-*MM*-*DD* format acquired by `JdbConvert.convertDate()`
TIME	NULL	null
	Other than the above	`String` object of a character string in *hh*:*mm*:*ss* format
TIMESTAMP	NULL	null
	Other than the above	`String` object of a character string in *yyyy*-*MM*-*DD* Δ *hh*:*mm*:*ss*[.*ffffff*] format
BINARYBLOB	NULL	null

HiRDB data type	Retrieval result	Return value
	Other than the above	String object obtained as the retrieval result
BOOLEAN#	NULL	null
	true	String object of the character string "true"
	false	String object of the character string "false"

#: There is BOOLEAN-type data when the Resultset was created from DatabaseMetadata.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- Encoding failed.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(az) getString(String columnName)

Function

Acquires the value of a specified column in the current row of this ResultSet object as String in Java programming language.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ay) getString(int columnIndex)*.

Format

```
public synchronized String getString(String columnName)
throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- Encoding failed.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

(ba) getTime(int columnIndex)

Function

Acquires as java.sql.Time the value of a specified column in the current row of this ResultSet object.

Format

```
public synchronized java.sql.Time getTime(int columnIndex)
throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is NULL, the method returns null)

Functional detail

This method acquires as `java.sql.Time` the value of a specified column in the current row of this `ResultSet` object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *time-format* [*single-byte-space*]	Value obtained by removing single-byte spaces at the beginning and end of the retrieval result and then converting to a `java.sql.Time` object
	Other than the above	`SQLException`
TIME	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Time` object
TIMESTAMP	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Time` object
Other	--	`SQLException`

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.

- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.

- ResultSet has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as java.sql.Time.
- An error occurred in the JDBC driver.

(bb)getTime(int columnIndex, Calendar cal)

Function

Acquires as a java.sql.Time object the value of a specified column in the current row of this ResultSet object. This method uses a specified calendar to create the appropriate time as a millisecond value.

Format

```
public synchronized java.sql.Time getTime (int columnIndex,
Calendar cal) throws SQLException
```

Arguments

int columnIndex

Column number

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is NULL, the method returns null)

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ba) getTime(int columnIndex)*.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

(bc) getTime(String columnName)

Function

Acquires as a `java.sql.Time` object the value of a specified column in the current row of this `ResultSet` object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ba) getTime(int columnIndex)*.

Format

```
public synchronized java.sql.Time getTime (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

(bd)getTime(String columnName, Calendar cal)

Function

Acquires as a `java.sql.Time` object the value of a specified column in the current row of this `ResultSet` object. This method uses a specified calendar to create the appropriate time as a millisecond value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(ba) getTime(int columnIndex)*.

Format

```
public synchronized java.sql.Time getTime (String
columnName, Calendar cal) throws SQLException
```

Arguments

String columnName

Column name

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

(be) getTimestamp(int columnIndex)

Function

Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.

Format

```
public synchronized java.sql.Timestamp getTimestamp (int
columnIndex) throws SQLException
```

Arguments

int columnIndex

Column number

Return value

Column value (if the value is `NULL`, the method returns `null`)

Functional detail

This method acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.

The following table shows the relationships among the HiRDB data type, the retrieval results, and the return value:

HiRDB data type	Retrieval result	Return value
[M|N][VAR]CHAR	NULL	null
	[*single-byte-space*] *timestamp-format* [*single-byte-space*]	Value obtained by removing single-byte spaces at the beginning and end of the retrieval result and then converting to a `java.sql.Timestamp` object
	Other than the above	`SQLException` is thrown
DATE	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Timestamp` object

HiRDB data type	Retrieval result	Return value
TIMESTAMP	NULL	null
	Other than the above	Retrieval result converted to a `java.sql.Timestamp` object
Other	--	`SQLException` is thrown

Legend:

-- Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

(bf) getTimestamp(int columnIndex, Calendar cal)

Function

Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object. This method uses a specified calendar to create the appropriate timestamp as a millisecond value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(be) getTimestamp(int columnIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (int
columnIndex, Calendar cal) throws SQLException
```

Arguments

int columnIndex

Column number

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column number is specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

(bg)getTimestamp(String columnName)

Function

Acquires as a `java.sql.Timestamp` object the value of a specified column in the current row of this `ResultSet` object.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(be) getTimestamp(int columnIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (String
columnName) throws SQLException
```

Arguments

String columnName

Column name

Return value

Column value (if the value is NULL, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as java.sql.Timestamp.
- An error occurred in the JDBC driver.

(bh) getTimestamp(String columnName, Calendar cal)

Function

Acquires as a java.sql.Timestamp object the value of a specified column in the current row of this ResultSet object. This method uses a specified calendar to create the appropriate timestamp as a millisecond value.

For the relationships among the HiRDB data type, the retrieval results, and the return value, see *Functional detail* in *(be) getTimestamp(int columnIndex)*.

Format

```
public synchronized java.sql.Timestamp getTimestamp (String
columnName, Calendar cal) throws SQLException
```

Arguments

String columnName

Column name

Calendar cal

Calendar in which the time zone for the values stored in the database has been set

Return value

Column value (if the value is `NULL`, the method returns `null`)

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

(bi) getType()

Function

Returns the type of this `ResultSet` object. This method returns `ResultSet.TYPE_FORWARD_ONLY` or `ResultSet.TYPE_SCROLL_INSENSITIVE`.

Format

```
public synchronized int getType() throws SQLException
```

Arguments

None.

Return value

ResultSet.TYPE_FORWARD_ONLY

The cursor can move only forward.

ResultSet.TYPE_SCROLL_INSENSITIVE

The cursor can be scrolled, but a value change does not take effect.

ResultSet.TYPE_SCROLL_SENSITIVE

The cursor can be scrolled, and a value change takes effect.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(bj) getWarnings()

Function

Acquires the first warning reported by a call related to this ResultSet object. If there is more than one warning, the subsequent warnings are chained to the first warning and can be acquired by calling the SQLWarning.getNextWarning method for the immediately preceding warning that was acquired.

Format

```
public synchronized SQLWarning getWarnings() throws
SQLException
```

Arguments

None.

Return value

First SQLWarning object (if there is no SQLWarning object, the method returns null)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(bk) isAfterLast()

Function

Acquires a value indicating whether the cursor is located after the last row in this `ResultSet` object.

Format

```
public synchronized boolean isAfterLast() throws
SQLException
```

Arguments

None.

Return value

If the cursor is located after the last row, the method returns `true`; if not, or the result set contains no rows, the method returns `false`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A database access error occurred.

(bl) isBeforeFirst()

Function

Acquires a value indicating whether the cursor is located before the first row in this `ResultSet` object.

Format

```
public synchronized boolean isBeforeFirst() throws
SQLException
```

Arguments

None.

Return value

If the cursor is located before the first row, the method returns `true`; if not, or the result set contains no rows, the method returns `false`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.
- A database access error occurred.

(bm)isFirst()

Function

Acquires a value indicating whether the cursor is located on the first row in this `ResultSet` object.

Format

```
public synchronized boolean isFirst() throws SQLException
```

Arguments

None.

Return value

If the cursor is located on the first row, the method returns `true`; if not, the method returns `false`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- `ResultSet` has become invalid due to transaction settlement.

- A database access error occurred.

(bn)isLast()

Function

Acquires a value indicating whether the cursor is located on the last row in this ResultSet object.

Format

```
public synchronized boolean isLast() throws SQLException
```

Arguments

None.

Return value

If the cursor is located on the last row, the method returns true; if not, the method returns false.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(bo)last()

Function

Moves the cursor to the last row of this ResultSet object.

Format

```
public synchronized boolean last() throws SQLException
```

Arguments

None.

Return value

If the cursor has moved to the last row, the method returns true; if the result set contains no rows, the method returns false.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- The type of this ResultSet object is TYPE_FORWARD_ONLY.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(bp) next()

Function

Moves the cursor from the current position to the next row. If the current cursor position is before the first row, the method moves the cursor to the first row; if it is on the last row, the method moves the cursor to a location after the last row.

The first time the next method is called, the cursor is opened.

Format

```
public synchronized boolean next() throws SQLException
```

Arguments

None.

Return value

If the cursor position resulting from this method call is before the first row or after the last row, the method returns false; otherwise, the method returns true.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this

ResultSet object has been closed.

- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(bq)previous()

Function

Moves the cursor to the immediately preceding row in this ResultSet object.

Format

```
public synchronized boolean previous() throws SQLException
```

Arguments

None.

Return value

If the cursor position resulting from this method call is before the first row, the method returns false; otherwise, the method returns true.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- The type of this ResultSet object is TYPE_FORWARD_ONLY.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(br) relative(int rows)

Function

Moves the cursor by the specified number of rows in the forward or reverse direction relative to the current position.

A positive number means that the cursor is to move in the forward direction, and a negative number that the cursor is to move in the reverse direction.

Format

```
public synchronized boolean relative(int rows) throws
SQLException
```

Arguments

int rows

Number of rows to be moved relative to the current row

Return value

If the cursor position resulting from this method call is before the first row or after the last row, the method returns `false`; otherwise, the method returns `true`.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where this driver closed `ResultSet` because the `Statement` that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `TYPE_FORWARD_ONLY`.
- The current position cannot be acquired.
- The current cursor position is not on a valid row.
- `ResultSet` has become invalid due to transaction settlement.
- A database access error occurred.

(bs) setFetchDirection(int direction)

Function

Sets the default fetch direction for the result set that is created from this `Statement` object.

Only `ResultSet.FETCH_FORWARD` can be specified.

Format

```
public synchronized void setFetchDirection(int direction)
throws SQLException
```

Arguments

int direction

Default fetch direction

Return value

None.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `Statement` object is closed.
- The `Connection` used to create this `Statement` object is closed.
- A value other than `ResultSet.FETCH_FORWARD` is specified for `direction`.

(bt) setFetchSize(int rows)

Function

Sets the fetch size of this `ResultSet` object. If `0` is specified, the fetch size depends on the client environment definition.

Format

```
public synchronized void setFetchSize(int rows) throws
SQLException
```

Arguments

int rows

Number of rows to be fetched, in the range from `0` to `4096`.

Return value

None.

Functional detail

This method sets the fetch size of this `ResultSet` object. When `0` is specified, the fetch size depends on the client environment definition.

If this method is omitted, the JDBC driver uses the number of rows specified in the `Statement` object for retrieval. If no number of rows is specified in the `Statement` object, or this `ResultSet` object has not been created from the `Statement` object, the JDBC driver uses the value of the `PDBLKF` client environment definition for retrieval. If `0` is specified in this method, the JDBC driver uses the value of the `PDBLKF`client environment definition for retrieval. For other notes, see *18.4.3(2)(z) setFetchSize(int rows)*.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- The type of this ResultSet object is TYPE_FORWARD_ONLY.
- The current position cannot be acquired.
- The current cursor position is not on a valid row.
- ResultSet has become invalid due to transaction settlement.
- A database access error occurred.

(bu)wasNull()

Function

Reports whether the last column value acquired is the NULL value.

This method returns false before any value has been acquired by a get*XXX* method.

Format

```
public synchronized boolean wasNull() throws SQLException
```

Arguments

None.

Return value

If the last column value acquired is NULL, the method returns true; otherwise, the method returns false.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where this driver closed ResultSet because the Statement that created this ResultSet object was closed.
- The Connection used to create the Statement that created this ResultSet object has been closed.
- ResultSet has become invalid due to transaction settlement.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbResultSet`

(4) Fields

The following table lists the fields supported by the `ResultSet` interface.

Table 18-31: Fields supported by the ResultSet interface

Field	Remarks
`public static final int FETCH_FORWARD`	--
`public static final int FETCH_REVERSE`	--
`public static final int FETCH_UNKNOWN`	--
`public static final int TYPE_FORWARD_ONLY`	--
`public static final int TYPE_SCROLL_INSENSITIVE`	--
`public static final int TYPE_SCROLL_SENSITIVE`	When this value is specified, the JDBC driver assumes that `TYPE_SCROLL_INSENSITIVE` was specified.
`public static final int CONCUR_READ_ONLY`	--
`public static final int CONCUR_UPDATABLE`	When this value is specified, the JDBC driver assumes that `CONCUR_READ_ONLY` was specified.
`public static final int HOLD_CURSORS_OVER_COMMIT`	--
`public static final int CLOSE_CURSORS_AT_COMMIT`	--

Legend:

--: None

(5) Notes

(a) Value acquisition using a getXXX method

- For details about whether mapping is possible with a `get`*XXX* method, see *18.8.2 Mapping during retrieval data acquisition*.
- If the column number or name specified in a `set`*XXX* method does not exist, the

JDBC driver throws an SQLException.

- If a value specified in a set*XXX* method exceeds the value range that can be represented by the data type of the corresponding ? parameter (for example, if getShort is used to get an INTEGER-type value of 40,000), an overflow occurs and results in an SQLException. For details about the combinations of set*XXX* methods for which overflow can occur and the HiRDB data types, see *18.8.5 Overflow handling*.

(b) Mapping (conversion)

For details about whether mapping is possible with a get*XXX* method to be used in getting retrieval data, see *18.8.2 Mapping during retrieval data acquisition*. If a get*XXX* method is called for a JDBC SQL type that cannot be mapped, the JDBC driver throws an SQLException.

(c) Using the block transfer facility by specifying the setFetchSize method

For details, see *18.4.3(4)(a) Using the block transfer facility by specifying the setFetchSize method*.

(d) Memory size used when the result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE

When the result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE, the JDBC driver allocates memory for accumulating the retrieval results when the following methods of the ResultSet interface are executed:

- ResultSet.next method
- ResultSet.last method
- ResultSet.absolute method
- ResultSet.relative method
- ResultSet.afterLast method

The JDBC driver assigns and accumulates memory objects to all values in the retrieval results. If a value has a variable length, the memory object is set to the actual size of the retrieved data.

(e) next, absolute, relative, last, and afterLast methods

When the next method is executed, the JDBC driver retrieves and accumulates data from the database as described in the following table.

Table 18-32: Data retrieved and accumulated from the database during execution of the next method

Condition	Result set type	
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE
The data of the current row, which was moved by the `next` method, has not been read into the JDBC driver.	The JDBC driver gets the moved current row from the connected database.	The JDBC driver gets the moved current row from the connected database, then reads and stores the row in its memory.
The data of the current row, which was moved by the `next` method, has been read into the JDBC driver.		The JDBC driver does not retrieve data from the connected database.

When the `absolute`, `relative`, `last`, or `afterLast` method is executed, the JDBC driver retrieves and accumulates data from the database as described in the following table.

Table 18-33: Data retrieved and accumulated from the database during execution of the absolute, relative, last, or afterLast method

Condition	Result set type is TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE
The first row to the specified row# of the retrieval results contain data that the JDBC driver has not read.	The JDBC driver retrieves the rows that were not read from the connected database and stores them in its memory.
The first row to the specified row# of the retrieval results do not contain data that the JDBC driver has not read.	The JDBC driver does not retrieve data from the connected database.

Note

If the data type of the result set is `TYPE_FORWARD_ONLY`, the JDBC driver throws an `SQLException`.

#

If the `last` or `afterLast` method is used, the range is from the first row to the last row.

(f) getAsciiStream, getBinaryStream, getCharacterStream, and getUnicodeStream methods

The JDBC driver does not implicitly close objects returned by the `getAsciiStream`, `getBinaryStream`, `getCharacterStream`, and `getUnicodeStream` methods. You must make provision for the method-calling side to execute the `close` method.

(g) Number of retrieved rows

The following table shows the number of retrieved rows that `ResultSet` objects can obtain from the HiRDB server. The JDBC driver discards retrieval results that exceed the applicable number of rows shown in the following table.

Table 18-34: Number of retrieved rows that ResultSet objects can obtain from the HiRDB server

ResultSet object	Result set type	
	TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE	Other type
`ResultSet` object generated by `Statement` object that executed `setMaxRows` method	The number of retrieved rows is the number of rows specified by `setMaxRows` method.	
Other `ResultSet` object	The number of retrieved rows is the upper limit for `setMaxRows` (2,147,483,647).	No upper limit

(h) About the default null value

Tables 18-35 and *18-36* show the return values of a `get`*XXXX* method for HiRDB data types when the default null value setting facility is used (by specifying `USE` in the `PDDFLNVAL` client environment definition) and the retrieval result is the default null value.

For details about the default null value setting facility, see the manual *HiRDB Version 9 SQL Reference*.

When the default null value setting facility is used, the `wasNull` method returns `FALSE`.

Note that the default null value setting facility is not applicable to a `ResultSet` object that has been acquired by a method of the `DatabaseMetaData` class. Therefore, the JDBC driver sets `NULL` as the `get`*XXXX* method's return value even if the default null value setting facility is used.

Table 18-35: Return value of a get*XXXX* method for HiRDB data types when the retrieval result is the default null value (1/2)

getXXX method	HiRDB data type				
	[M|N][VAR] CHAR	BINARY BLOB	DATE	TIME	TIMESTAMP
getByte	--	--	--	--	--
getShort	--	--	--	--	--
getInt	--	--	--	--	--
getLong	--	--	--	--	--
getFloat	--	--	--	--	--
getDouble	--	--	--	--	--
getBigDecimal	--	--	--	--	--
getBoolean	FALSE	--	--	--	--
getString	Default NULL value	`String` object of the default NULL value	`String` object of a character string in *yyyy-MM-DD* format obtained by converting the default NULL value	`String` object of a character string in *hh*:*mm*:*ss* format obtained by converting the default NULL value	`String` object of a character string in *yyyy-MM-DD* Δ*hh*:*mm*:*ss*[.*ffffff*] format obtained by converting the default NULL value
getBytes	Default NULL value converted to a `byte` array	Same as left	--	--	--
getDate	--	--	Default NULL value converted to a `java.sql.Date` object	--	Default NULL value converted to a `java.sql.Date` object
getTime	--	--	--	Default NULL value converted to a `java.sql.Time` object	Same as left

getXXX method	HiRDB data type				
	[M|N][VAR] CHAR	BINARY BLOB	DATE	TIME	TIMESTAMP
getTimestamp	--	--	Default NULL value converted to a java.sql.Timestamp object	--	Default NULL value converted to a java.sql.Timestamp object
getAsciiStream	InputStream object containing the default NULL value	Same as left	--	--	--
getBinaryStream	--	InputStream object containing the default NULL value	--	--	--
getObject	Default NULL value	Same as left	java.sql.Date object created by the default NULL value	java.sql.Time object created by the default NULL value	java.sql.Timestamp object created by the default NULL value
getCharacterStream	Reader object containing the default NULL value	Same as left	--	--	--
getArray	--	--	--	--	--
getBlob	--	java.sql.Blob object that has the default NULL value	--	--	--

Legend:

-- An SQLException is thrown.

Table 18-36: Return value of a get*XXXX* method for HiRDB data types when the retrieval result is the default null value (2/2)

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	REAL	FLOAT	DECIMAL	ARRAY
getByte	Default NULL value converted to a `byte` value	Same as left	Same as left	Same as left	Same as left	--
getShort	Default NULL value	Default NULL value converted to a `short` value	Integer part of the default NULL value converted to a `short` value	Same as left	Same as left	--
getInt	Default NULL value converted to an `int` value	Same as left	Integer part of the default NULL value converted to an `int` value	Same as left	Same as left	--
getLong	Default NULL value converted to a `long` value	Same as left	Integer part of the default NULL value converted to a `long` value	Same as left	Same as left	--
getFloat	Default NULL value converted to a `float` value	Same as left	Same as left	Same as left	Same as left	--

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	REAL	FLOAT	DECIMAL	ARRAY
getDouble	Default NULL value converted to a `double` value	Same as left	Same as left	Same as left	Same as left	--
getBigDecimal	`BigDecimal` object that has the default NULL value	Same as left	Same as left	Same as left	Same as left	--
getBoolean	FALSE	Same as left	Same as left	Same as left	Same as left	--
getString	`String` object containing the default NULL value in character string representation	Same as left	Same as left	Same as left	Same as left	--
getBytes	--	--	--	--	--	--
getDate	--	--	--	--	--	--
getTime	--	--	--	--	--	--
getTimestamp	--	--	--	--	--	--
getAsciiStream	--	--	--	--	--	--
getBinaryStream	--	--	--	--	--	--
getObject	`Integer` object created by the default NULL value	Same as left	`Float` object created by the default NULL value	`Double` object created by the default NULL value	Default NULL value	`Array` object created by the default NULL value
getCharacterStream	--	--	--	--	--	--

getXXX method	JDBC SQL type					
	SMALLINT	INTEGE R	REAL	FLOAT	DECIMA L	ARRAY
getArray	--	--	--	--	--	java.sql.Array object containing no element
getBlob	--	--	--	--	--	--

Legend:

-- An SQLException is thrown.

18.4.7 DatabaseMetaData interface

(1) Overview

The DatabaseMetaData interface provides the following principal functions:

- Return of various information related to the connected database
- Return of listing information, such as a list of tables or columns (the information is stored in a result set)

Some methods of the DatabaseMetaData class use a String pattern character string as an argument. The following table shows the special characters that can be specified in such a pattern character string.

Special character	Description
_ (underscore)	Any single character
%	A character string of any length, including no characters (as well as any number of characters)
\	Escape character (enables a special character immediately following this escape character in a pattern character string to be treated as a regular character)

(2) Methods

The table below lists the methods of the DatabaseMetaData interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-37: DatabaseMetaData interface methods

Subsection	Method	Function
(a)	allProceduresAreCallable()	Returns a value indicating whether all the procedures returned by the `getProcedures()` method can be called by the current user.
(b)	allTablesAreSelectable()	Returns a value indicating whether all the tables returned by the `getTables()` method can be used by the current user.
(c)	dataDefinitionCausesTransactionCommit()	Returns a value indicating whether a data definition statement in a transaction is to forcibly commit the transaction.
(d)	dataDefinitionIgnoredInTransactions()	Returns a value indicating whether data definition statements are ignored in transactions.
(e)	deletesAreDetected(int type)	Returns a value indicating whether deletions of visible rows can be detected by calling the `ResultSet.rowDeleted()` method.
(f)	doesMaxRowSizeIncludeBlobs()	Returns a value indicating whether the `getMaxRowSize()` method contains the `LONGVARCHAR` and `LONGVARBINARY` SQL data types.
(g)	getAttributes(String catalog,String schemaPattern,String typeNamePattern,String attributeNamePattern)	Returns attribute-related information for user-defined types that can be used in catalogs and specified schemas.
(h)	getBestRowIdentifier(String catalog,String schema,String table,int scope,boolean nullable)	Returns information about the optimum column set for a table in which rows are identified uniquely.
(i)	getCatalogs()	Returns the available catalog names.
(j)	getCatalogSeparator()	Returns the separator between the catalog name and the table name.
(k)	getCatalogTerm()	Returns a word recommended for `catalog`.
(l)	getColumnPrivileges (String catalog,String schema,String table,String columnNamePattern)	Returns information about table column access permissions.
(m)	getColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)	Returns information about specified table columns.
(n)	getConnection()	Returns the `Connection` instance that created this `DatabaseMetaData` instance.

Subsection	Method	Function
(o)	getCrossReference (String primaryCatalog,String primarySchema,String primaryTable,String foreignCatalog,1String foreignSchema)	Returns cross-reference information between a specified referencing table and referenced table.
(p)	getDatabaseMajorVersion()	Returns the database's major version information.
(q)	getDatabaseMinorVersion()	Returns the database's minor version information.
(r)	getDatabaseProductName()	Returns the connected database's product name.
(s)	getDatabaseProductVersion()	Returns the connected database's version.
(t)	getDefaultTransactionIsolation()	Returns the default transaction cut-off level.
(u)	getDriverMajorVersion()	Returns this JDBC driver's major version as `int` type.
(v)	getDriverMinorVersion()	Returns this JDBC driver's minor version as `int` type.
(w)	getDriverName()	Returns the JDBC driver name `HiRDB_Type4_JDBC_Driver`.
(x)	getDriverVersion()	Returns the version of this JDBC driver as `String`.
(y)	getExportedKeys (String catalog,String schema,String table)	Returns information about a specified table's foreign keys.
(z)	getExtraNameCharacters()	Returns the special characters (characters other than `a` to `z`, `A` to `Z`, `0` to `9`, and the underscore (_)) that can be used in an SQL ID name that is not enclosed in double quotation marks.
(aa)	getIdentifierQuoteString()	Returns the character string used to enclose SQL identifiers.
(ab)	getImportedKeys (String catalog,String schema,String table)	Returns information about a specified table's primary key.
(ac)	getIndexInfo(String catalog,String schema,String table,boolean unique,boolean approximate)	Returns information about the indexes of a specified table.
(ad)	getJDBCMajorVersion()	Returns the driver's JDBC major version.
(ae)	getJDBCMinorVersion()	Returns the driver's JDBC minor version.
(af)	getMaxBinaryLiteralLength()	Returns the maximum number of hexadecimal characters that can be used in a binary literal.

Subsection	Method	Function
(ag)	getMaxCatalogNameLength()	Returns the maximum length of a catalog name (number of characters).
(ah)	getMaxCharLiteralLength()	Returns the maximum length of a character literal (number of characters).
(ai)	getMaxColumnNameLength()	Returns the maximum length of a column name (number of characters).
(aj)	getMaxColumnsInGroupBy()	Returns the maximum number of columns in a `GROUP BY` clause.
(ak)	getMaxColumnsInIndex()	Returns the maximum number of columns permitted for an index.
(al)	getMaxColumnsInOrderBy()	Returns the maximum number of columns in an `ORDER BY` clause.
(am)	getMaxColumnsInSelect()	Returns the maximum number of columns in a `SELECT` list.
(an)	getMaxColumnsInTable()	Returns the maximum number of columns in a table.
(ao)	getMaxConnections()	Returns the maximum number of concurrent connections.
(ap)	getMaxCursorNameLength()	Returns the maximum length of a cursor name (number of characters).
(aq)	getMaxIndexLength()	Returns the maximum length of an index, including all parts of the index.
(ar)	getMaxProcedureNameLength()	Returns the maximum length of a procedure name (number of characters).
(as)	getMaxRowSize()	Returns the maximum length of a row (in bytes).
(at)	getMaxSchemaNameLength()	Returns the maximum length of a schema name (number of characters).
(au)	getMaxStatementLength()	Returns the maximum length of an SQL statement.
(av)	getMaxStatements()	Returns the maximum number of SQL statements that can be active.
(aw)	getMaxTableNameLength()	Returns the maximum length of a table name (number of characters).

Subsection	Method	Function
(ax)	getMaxTablesInSelect()	Returns the maximum number of tables in a SELECT statement.
(ay)	getMaxUserNameLength()	Returns the maximum length of a user name (number of characters).
(az)	getNumericFunctions()	Returns a list of the available mathematical functions (delimited by a comma).
(ba)	getPrimaryKeys(String catalog, String schema, String table)	Returns information about a specified table's primary key columns.
(bb)	getProcedureColumns(String catalog,String schemaPattern,String procedureNamePattern, String columnNamePattern)	Returns information about stored procedure parameters.
(bc)	getProcedures(String catalog,String schemaPattern,String procedureNamePattern)	Returns information about stored procedures.
(bd)	getProcedureTerm()	Returns a word recommended for procedure.
(be)	getResultSetHoldability()	Returns the holding facility for ResultSet objects.
(bf)	getSchemas()	Returns the available schema names.
(bg)	getSchemaTerm()	Returns a word recommended for schema.
(bh)	getSearchStringEscape()	Returns the character string used as the escape sequence for wildcard characters.
(bi)	getSQLKeywords()	Returns a list (delimited by a comma) of all database-specific SQL keywords that are not SQL92 keywords.
(bj)	getSQLStateType()	Returns a value indicating whether SQLSTATE returned by SQLException.getSQLState is an X/Open SQL CLI or SQL99.
(bk)	getStringFunctions()	Returns a list of string functions (delimited by a comma).
(bl)	getSuperTables(String catalog,String schemaPattern,String tableNamePattern)	Returns a description of table hierarchies defined in a specified schema.
(bm)	getSuperTypes(String catalog,String schemaPattern,String typeNamePattern)	Returns a description of user-defined-type hierarchies that are defined in a specified schema.
(bn)	getSystemFunctions()	Returns the available system functions.

Subsection	Method	Function
(bo)	getTablePrivileges(String catalog,String schemaPattern,String tableNamePattern)	Returns information about table access privileges.
(bp)	getTables(String catalog,String schemaPattern,String tableNamePattern,String[] types)	Returns information about tables.
(bq)	getTableTypes()	Returns the available table types.
(br)	getTimeDateFunctions()	Returns a list of the available time and date functions (delimited by a comma).
(bs)	getTypeInfo()	Returns information about the default SQL types.
(bt)	getUDTs(String catalog,String schemaPattern,String typeNamePattern,int[] types)	Returns information about the user-defined types.
(bu)	getURL()	Returns the URL used for the connection to HiRDB or XDM/RD E2.
(bv)	getUserName()	Returns the user name used to connect to HiRDB or XDM/RD E2.
(bw)	getVersionColumns(String catalog,String schema,String table)	Returns information about the table columns that are updated automatically when rows in the table are updated.
(bx)	insertsAreDetected(int type)	Returns a value indicating whether insertion of a visible row can be detected by calling the `ResultSet.rowInserted()` method.
(by)	isCatalogAtStart()	Returns a value indicating whether a catalog appears at the leading (or trailing) end of a fully qualified table name.
(bz)	isReadOnly()	Returns a value indicating whether the database is in read-only mode.
(ca)	locatorsUpdateCopy()	Indicates whether a change was made to a LOB copy or directly to the LOB.
(cb)	nullPlusNonNullIsNull()	Returns a value indicating whether a join of a `NULL` value and a non-`NULL` value is treated as being `NULL`.
(cc)	nullsAreSortedAtEnd()	Returns a value indicating whether the `NULL` value is sorted during termination processing (regardless of the sort order).

Subs ectio n	Method	Function
(cd)	nullsAreSortedAtStart()	Returns a value indicating whether the NULL value is sorted during startup processing (regardless of the sort order).
(ce)	nullsAreSortedHigh()	Returns a value indicating whether the NULL value is sorted in ascending order.
(cf)	nullsAreSortedLow()	Returns a value indicating whether the NULL value is sorted in descending order.
(cg)	othersDeletesAreVisible(int type)	Returns a value indicating whether a deletion performed externally is visible.
(ch)	othersInsertsAreVisible(int type)	Returns a value indicating whether an insertion performed externally is visible.
(ci)	othersUpdatesAreVisible(int type)	Returns a value indicating whether a deletion performed externally is visible.
(cj)	ownDeletesAreVisible(int type)	Returns a value indicating whether a deletion of a result set itself is visible.
(ck)	ownInsertsAreVisible(int type)	Returns a value indicating whether an insertion of a result set itself is visible.
(cl)	ownUpdatesAreVisible(int type)	Returns a value indicating whether an updating of a result set itself is visible.
(cm)	storesLowerCaseIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all lower-case letters.
(cn)	storesLowerCaseQuotedIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all lower-case letters.
(co)	storesMixedCaseIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in upper-case and lower-case letters.

Subs ectio n	Method	Function
(cp)	storesMixedCaseQuotedIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in upper-case and lower-case letters.
(cq)	storesUpperCaseIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all upper-case letters.
(cr)	storesUpperCaseQuotedIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all upper-case letters.
(cs)	supportsAlterTableWithAddColumn()	Returns a value indicating whether ALTER TABLE with added columns is supported.
(ct)	supportsAlterTableWithDropColumn()	Returns a value indicating whether ALTER TABLE with dropped columns is supported.
(cu)	supportsANSI92EntryLevelSQL()	Returns a value indicating whether the ANSI92 entry-level SQL grammar is supported.
(cv)	supportsANSI92FullSQL()	Returns a value indicating whether the ANSI92 full-level SQL grammar is supported.
(cw)	supportsANSI92IntermediateSQL()	Returns a value indicating whether the ANSI92 intermediate-level SQL grammar is supported.
(cx)	supportsBatchUpdates()	Returns a value indicating whether batch updating is supported.
(cy)	supportsCatalogsInDataManipulation()	Returns a value indicating whether catalog names can be used in data manipulation statements.
(cz)	supportsCatalogsInIndexDefinitions()	Returns a value indicating whether catalog names can be used in index definition statements.
(da)	supportsCatalogsInPrivilegeDefinitions()	Returns a value indicating whether catalog names can be used in privilege definition statements.
(db)	supportsCatalogsInProcedureCalls()	Returns a value indicating whether catalog names can be used in procedure call statements.

Subsection	Method	Function
(dc)	supportsCatalogsInTableDefinitions()	Returns a value indicating whether catalog names can be used in table definition statements.
(dd)	supportsColumnAliasing()	Returns a value indicating whether aliases are supported for columns.
(de)	supportsConvert()	Returns a value indicating whether the CONVERT function is supported for SQL types.
(df)	supportsConvert(int fromType,int toType)	Returns a value indicating whether the CONVERT function is supported for given SQL types.
(dg)	supportsCoreSQLGrammar()	Returns a value indicating whether the ODBC Core SQL grammar is supported.
(dh)	supportsCorrelatedSubqueries()	Returns a value indicating whether correlated subqueries are supported.
(di)	supportsDataDefinitionAndDataManipulationTransactions()	Returns a value indicating whether data definition statements and data manipulation statements are both supported in transactions.
(dj)	supportsDataManipulationTransactionsOnly()	Returns a value indicating whether only data manipulation statements are supported in transactions.
(dk)	supportsDifferentTableCorrelationNames()	Returns a value indicating whether the table names must be different from the correlation names when table correlation names are supported.
(dl)	supportsExpressionsInOrderBy()	Returns a value indicating whether expressions are supported in an ORDER BY list.
(dm)	supportsExtendedSQLGrammar()	Returns a value indicating whether the ODBC Extended SQL grammar is supported.
(dn)	supportsFullOuterJoins()	Returns a value indicating whether full outer joins are supported.
(do)	supportsGetGeneratedKeys()	Returns a value indicating whether automatic generation keys can be acquired after statements have executed.
(dp)	supportsGroupBy()	Returns a value indicating whether the GROUP BY clause form is supported.

Subsection	Method	Function
(dq)	supportsGroupByBeyondSelect()	Returns a value indicating whether a column for which the GROUP BY clause is not specified in SELECT can be used when all columns in SELECT must be specified.
(dr)	supportsGroupByUnrelated()	Returns a value indicating whether a column for which the GROUP BY clause is not specified in SELECT can be used.
(ds)	supportsIntegrityEnhancementFacility()	Returns a value indicating whether the SQL Integrity Enhancement Facility is supported.
(dt)	supportsLikeEscapeClause()	Returns a value indicating whether escape characters are supported in the LIKE clause.
(du)	supportsLimitedOuterJoins()	Returns a value indicating whether limited support is provided for outer joins.
(dv)	supportsMinimumSQLGrammar()	Returns a value indicating whether the ODBC Minimum SQL grammar is supported.
(dw)	supportsMixedCaseIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being case sensitive, and then the results are stored in upper-case and lower-case letters.
(dx)	supportsMixedCaseQuotedIdentifiers()	Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being case sensitive, and then the results are stored in upper-case and lower-case letters.
(dy)	supportsMultipleOpenResults()	Returns a value indicating whether it is possible for multiple ResultSet objects to be returned simultaneously by a CallableStatement object.
(dz)	supportsMultipleResultSets()	Returns a value indicating whether multiple ResultSet objects can be acquired from execution of a single execute method.
(ea)	supportsMultipleTransactions()	Returns a value indicating whether multiple transactions can be open at the same time (for different connections).

Subsection	Method	Function
(eb)	supportsNamedParameters()	Returns a value indicating whether named parameters are supported for the `callable` statement.
(ec)	supportsNonNullableColumns()	Returns a value indicating whether columns can be defined as non-null columns.
(ed)	supportsOpenCursorsAcrossCommit()	Returns a value indicating whether the cursor can remain open between commit operations.
(ee)	supportsOpenCursorsAcrossRollback()	Returns a value indicating whether the cursor can remain open between rollback operations.
(ef)	supportsOpenStatementsAcrossCommit()	Returns a value indicating whether statements can remain open between commit operations.
(eg)	supportsOpenStatementsAcrossRollback()	Returns a value indicating whether statements can remain open between rollback operations.
(eh)	supportsOrderByUnrelated()	Returns a value indicating whether a column for which the `ORDER BY` clause is not in `SELECT` can be used.
(ei)	supportsOuterJoins()	Returns a value indicating whether some form of outer join is supported.
(ej)	supportsPositionedDelete()	Returns a value indicating whether positioned `DELETE` is supported.
(ek)	supportsPositionedUpdate()	Returns a value indicating whether positioned `UPDATE` is supported.
(el)	supportsResultSetConcurrency(int type, int concurrency)	Returns a value indicating whether the combination of a specified type of `ResultSet` and a specified parallel processing type is supported.
(em)	supportsResultSetHoldability(int holdability)	Returns a value indicating whether the holding facility is supported for a specified `ResultSet` object.
(en)	supportsResultSetType(int type)	Returns a value indicating whether a specified type of `ResultSet` is supported.
(eo)	supportsSavepoints()	Returns a value indicating whether save points are supported.
(ep)	supportsStatementPooling()	Returns a value indicating whether statement pooling is supported.

Subsection	Method	Function
(eq)	supportsSchemasInDataManipulation()	Returns a value indicating whether schema names can be used in data manipulation statements.
(er)	supportsSchemasInIndexDefinitions()	Returns a value indicating whether schema names can be used in index definition statements.
(es)	supportsSchemasInPrivilegeDefinitions()	Returns a value indicating whether schema names can be used in privilege definition statements.
(et)	supportsSchemasInProcedureCalls()	Returns a value indicating whether schema names can be used in procedure calls.
(eu)	supportsSchemasInTableDefinitions()	Returns a value indicating whether schema names can be used in table definition statements.
(ev)	supportsSelectForUpdate()	Returns a value indicating whether SELECT is supported for updating.
(ew)	supportsStoredProcedures()	Returns a value indicating whether stored procedure calls are supported.
(ex)	supportsSubqueriesInComparisons()	Returns a value indicating whether subqueries are supported in comparison expressions.
(ey)	supportsSubqueriesInExists()	Returns a value indicating whether subqueries are supported in exists expressions.
(ez)	supportsSubqueriesInIns()	Returns a value indicating whether subqueries are supported in in statements.
(fa)	supportsSubqueriesInQuantifieds()	Returns a value indicating whether subqueries are supported in quantified expressions.
(fb)	supportsTableCorrelationNames()	Returns a value indicating whether table correlation names are supported.
(fc)	supportsTransactionIsolationLevel(int level)	Returns a value indicating whether a specified transaction isolation level is supported.
(fd)	supportsTransactions()	Returns a value indicating whether transactions are supported.
(fe)	supportsUnion()	Returns a value indicating whether SQL UNION is supported.
(ff)	supportsUnionAll()	Returns a value indicating whether SQL UNION ALL is supported.

Subsection	Method	Function
(fg)	updatesAreDetected(int type)	Returns a value indicating whether updating performed on a `ResultSet` of a specified `ResultSet` type can be detected by the `ResultSet.rowUpdated` method.
(fh)	usesLocalFilePerTable()	Returns a value indicating whether a file is to be used for each table.
(fi)	usesLocalFiles()	Returns a value indicating whether tables are to be stored in local files.

(a) allProceduresAreCallable()

Function

Returns a value indicating whether all the procedures returned by the `getProcedures()` method can be called by the current user.

Format

```
public boolean allProceduresAreCallable() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Can be called.

false: Cannot be called.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(b) allTablesAreSelectable()

Function

Returns a value indicating whether all the tables returned by the `getTables()`

method can be used by the current user.

Format

```
public boolean allTablesAreSelectable() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Can be used.

false: Cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(c) dataDefinitionCausesTransactionCommit()

Function

Returns a value indicating whether a data definition statement in a transaction is to forcibly commit the transaction.

Format

```
public boolean dataDefinitionCausesTransactionCommit()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Commit

false: Not commit

Functional detail

This method always returns `true`.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(d) dataDefinitionIgnoredInTransactions()

Function

Returns a value indicating whether data definition statements are ignored in transactions.

Format

```
public boolean dataDefinitionIgnoredInTransactions() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Ignored

false: Not ignored

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(e) deletesAreDetected(int type)

Function

Returns a value indicating whether deletions of visible rows can be detected by calling the ResultSet.rowDeleted() method.

Format

```
public boolean deletesAreDetected(int type) throws
SQLException
```

Arguments

int type

One of the following ResultSet types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

boolean type:

true: Deletions are detected.

false: Deletions are not detected.

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(f) doesMaxRowSizeIncludeBlobs()

Function

Returns a value indicating whether the getMaxRowSize() method contains the LONGVARCHAR and LONGVARBINARY SQL data types.

Format

```
public boolean doesMaxRowSizeIncludeBlobs() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Contains the LONGVARCHAR and LONGVARBINARY SQL data types.

false: Does not contain the LONGVARCHAR and LONGVARBINARY SQL data types.

Functional detail

This method always returns false.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(g) getAttributes (String catalog,String schemaPattern,String typeNamePattern,String attributeNamePattern)

Function

Returns attribute-related information for user-defined types that can be used in catalogs and specified schemas.

Format

```
public ResultSet getAttributes(String catalog, String
schemaPattern, String typeNamePattern, String
attributeNamePattern) throws SQLException
```

Arguments

String catalog

Catalog name

String schemaPattern

Schema name pattern

String typeNamePattern

Type name pattern

String attributeNamePattern

Attribute name pattern

Return value

`ResultSet` object

Functional detail

This method always returns a `ResultSet` in which the number of rows resulting from the retrieval is 0 because the Type4 JDBC driver does not support user-defined types. The method does not perform a validity check on any of the arguments. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Description
1	String	CHAR	TYPE_CAT	Catalog name
2	String	VARCHAR	TYPE_SCHEM	Authorization identifier

Column No.	Type	SQL type (Types)	Column name	Description
3	String	VARCHAR	TYPE_NAME	Type name
4	String	VARCHAR	ATTR_NAME	Attribute name
5	int	INTEGER	DATA_TYPE	Attribute type
6	String	VARCHAR	ATTR_TYPE_NAME	Type name
7	int	INTEGER	ATTR_SIZE	Column size
8	int	INTEGER	DECIMAL_DIGITS	Decimal places
9	int	INTEGER	NUM_PREC_RADIX	Radix
10	int	INTEGER	NULLABLE	Whether the `NULL` value is permitted
11	String	VARCHAR	REMARKS	Comments
12	String	VARCHAR	ATTR_DEF	Default value
13	int	INTEGER	SQL_DATA_TYPE	Not used
14	int	INTEGER	SQL_DATETIME_SUB	Not used
15	int	INTEGER	CHAR_OCTET_LENGTH	Maximum length (in bytes) of a `char`-type column
16	int	INTEGER	ORDINAL_POSITION	Column index in table
17	String	VARCHAR	IS_NULLABLE	Whether the `NULL` value is permitted
18	String	VARCHAR	SCOPE_CATALOG	Catalog for a table in the scope of reference attribute
19	String	VARCHAR	SCOPE_SCHEMA	Schema for a table in the scope of reference attribute
20	String	VARCHAR	SCOPE_TABLE	Table name in the scope of reference attribute
21	short	SMALLINT	SOURCE_DATA_TYPE	Source data type for individual types, user-defined `Ref` type, or `java.sql.Types` SQL type

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(h) getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)

Function

Returns information about the optimum column set for a table in which rows are identified uniquely.

Format

```
public ResultSet getBestRowIdentifier(String catalog, String
schema, String table, int scope, boolean nullable) throws
SQLException
```

Arguments

String catalog

Catalog name

String schema

Schema name

String table

Table name

int scope

Target scale

boolean nullable

`NULL` value specification

Return value

`ResultSet` object

Functional detail

This method does not perform a validity check on any of the arguments. It always returns a `ResultSet` in which the number of rows resulting from retrieval is 0.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(i) getCatalogs()

Function

Returns the available catalog names.

Format

```
public ResultSet getCatalogs() throws SQLException
```

Arguments

None.

Return value

`ResultSet` object

Functional detail

This method always returns a `ResultSet` in which the number of rows resulting from the retrieval is 0.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(j) getCatalogSeparator()

Function

Returns the separator between the catalog name and the table name.

Format

```
public String getCatalogSeparator() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `null`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(k) getCatalogTerm()

Function

Returns a word recommended for `catalog`.

Format

```
public String getCatalogTerm() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `null`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(l) getColumnPrivileges (String catalog,String schema,String table,String columnNamePattern)

Function

Returns information about table column access permissions.

Format

```
public ResultSet getColumnPrivileges(String catalog, String
schema, String table, String columnNamePattern) throws
SQLException
```

Arguments

String catalog

Catalog name (this driver ignores this argument)

String schema

Schema name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String table

Table name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a table name is not used to narrow down the search.

String columnNamePattern

Column name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a column name is not used to narrow down the search.

Return value

`ResultSet` object

Functional detail

This method returns information about the access permissions for the table to which the specified table column belongs. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	TABLE_CAT	--	Catalog name (`NULL` value is always returned)
2	String	VARCHAR	TABLE_SCHEM	1 (ascending order)	Authorization identifier
3	String	VARCHAR	TABLE_NAME	2 (ascending order)	Table name
4	String	VARCHAR	COLUMN_NAME	3 (ascending order)	Column name
5	String	VARCHAR	GRANTOR	--	User who grants access privileges
6	String	VARCHAR	GRANTEE	--	User who receives access privileges
7	String	VARCHAR	PRIVILEGE	4 (ascending order)	Names of granted access privileges (multiple access privilege names are delimited by the comma): `SELECT`: `SELECT` privilege `INSERT`: `INSERT` privilege `UPDATE`: `UPDATE` privilege `DELETE`: `DELETE` privilege

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
8	String	VARCHAR	IS_GRANTABLE	--	Whether a user who has received access privileges can grant access privileges to other users: `YES`: Can grant privileges to other users. `NO`: Cannot grant privileges to other users. `null`: Whether the user can grant privileges to other users is unknown. If multiple access privileges are set in `PRIVILEGE` and at least one of the privileges can be granted to other users, the method sets `YES`. `null` is not returned in either HiRDB or XDM/RD E2.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(m) getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)

Function

Returns information about specified table columns.

Format

```
public ResultSet getColumns(String catalog, String
schemaPattern, String tableNamePattern, String
columnNamePattern) throws SQLException
```

Arguments

String catalog

Catalog name (this driver ignores this argument)

String schemaPattern

Schema name pattern (case sensitive)

String tableNamePattern

Table name pattern (case sensitive)

String columnNamePattern

Column name pattern (case sensitive)

Return value

`ResultSet` object

Functional detail

Returns information about the specified table columns. The following table shows the format of the `ResultSet` that is returned.

Table 18-38: Format of `ResultSet` returned by getColumns

Column No.	Type	SQL type (Types)	Column name	Description
1	String	CHAR	TABLE_CAT	`NULL` value is always returned
2	String	VARCHAR	TABLE_SCHEM	Authorization identifier
3	String	VARCHAR	TABLE_NAME	Table name
4	String	VARCHAR	COLUMN_NAME	Column name
5	int	INTEGER	DATA_TYPE	SQL type
6	String	CHAR	TYPE_NAME	Type name
7	int	INTEGER	COLUMN_SIZE	Column size
8	int	INTEGER	BUFFER_LENGTH	`NULL` value is always returned
9	int	INTEGER	DECIMAL_DIGITS	Decimal places
10	int	INTEGER	NUM_PREC_RADIX	Radix • Approximate value: `2` • Exact value: `10` • Non-numeric value: `0`

Colum n No.	Type	SQL type (Types)	Column name	Description
11	int	INTEGER	NULLABLE	Whether the NULL value can be used for this type: • columnNoNulls: NULL value might not be permitted. • columnNullable: NULL value can be used. • columnNullableUnknown: Whether the NULL value can be used is unknown.
12	String	VARCHAR	REMARKS	Comment column
13	String	VARCHAR	COLUMN_DEF	Default value
14	int	INTEGER	SQL_DATA_TYPE	NULL value is always returned
15	int	INTEGER	SQL_DATETIME_SUB	NULL value is always returned
16	int	INTEGER	CHAR_OCTET_LENGTH	Maximum length in bytes of char-type column (same as COLUMN_SIZE)
17	int	SMALLINT	ORDINAL_POSITION	Column number (beginning with 1)
18	String	CHAR	IS_NULLABLE	Whether the NULL value can be used for this type: • NO: NULL value cannot be used. • YES: NULL value might be permitted.
19	String	CHAR	SCOPE_CATLOG	NULL value is always returned
20	String	CHAR	SCOPE_SCHEMA	NULL value is always returned
21	String	CHAR	SCOPE_TABLE	NULL value is always returned
22	short	SMALLINT	SOURCE_DATA_TYPE	NULL value is always returned

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(n) getConnection()

Function

Returns the Connection instance that created this DatabaseMetaData instance.

Format

```
public Connection getConnection() throws SQLException
```

Arguments

None.

Return value

`Connection` object

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(o) getCrossReference (String primaryCatalog,String primarySchema,String primaryTable,String foreignCatalog,1String foreignSchema)

Function

Returns cross-reference information between a specified referencing table and referenced table.

Format

```
public ResultSet getCrossReference(String primaryCatalog,
String primarySchema, String primaryTable, String
foreignCatalog, String foreignSchema) throws SQLException
```

Arguments

String primaryCatalog

Catalog name of the referenced table (this driver ignores this argument)

String primarySchema

Schema name pattern of the referenced table (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String primaryTable

Table name pattern of the referenced table (case sensitive). If `null` or a character string with a length of 0 is specified, a table name is not used to narrow down the search.

String foreignCatalog

Catalog name of the referencing table (this driver ignores this argument)

String foreignSchema

Schema name pattern of the referencing table (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

Return value

`ResultSet` object

Functional detail

This method returns cross-reference information between a specified referencing table and referenced table. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	PKTABLE_CAT	--	Catalog name of the referenced table (`NULL` value is always returned)
2	String	VARCHAR	PKTABLE_SCHEM	--	Authorization identifier of the referenced table
3	String	VARCHAR	PKTABLE_NAME	--	Table name of the referenced table
4	String	VARCHAR	PKCOLUMN_NAME	--	Column name of primary key
5	String	CHAR	FKTABLE_CAT	--	Catalog name of the referencing table (`NULL` value is always returned)
6	String	VARCHAR	FKTABLE_SCHEM	1 (ascending order)	Authorization identifier of the referencing table
7	String	VARCHAR	FKTABLE_NAME	2 (ascending order)	Table name of the referencing table
8	String	VARCHAR	FKCOLUMN_NAME	--	Column name of foreign key
9	short	SMALLINT	KEY_SEQ	3 (ascending order)	Sequence number of foreign key

C ol u m n N o.	Type	SQL type (Types)	Column name	Sorting#	Description
1 0	short	SMALLINT	UPDATE_RULE	--	Updating rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is updated. • `importedKeyNoAction` The primary key cannot be updated. • `importedKeyCascade` The foreign key is updated by the same value as for the primary key. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be updated.
11	short	SMALLINT	DELETE_RULE	--	Deletion rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is deleted. • `importedKeyNoAction` The primary key cannot be deleted. • `importedKeyCascade` The row containing the foreign key is deleted. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be deleted.
1 2	String	VARCHAR	FK_NAME	--	Constraint name of referential constraints

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
13	String	VARCHAR	PK_NAME	--	Index name of the primary key
14	short	SMALLINT	DEFERRABILITY	--	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed: • `importedKeyInitiallyDeferred` Can be postponed. • `importedKeyInitiallyImmediate` Currently set for immediate evaluation, but the setting can be changed to postpone evaluation. • `importedKeyNotDeferrable` Cannot be postponed.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(p) getDatabaseMajorVersion()

Function

Returns the database's major version information.

Format

```
public int getDatabaseMajorVersion() throws SQLException
```

Arguments

None.

Return value

`int` type:

Database's major version

Functional detail

When a HiRDB server is used, this method returns the HiRDB server's major version. For example, if the HiRDB server's version is 08-02, the method returns 8 of the `int` type.

When an XDM/RD E2 server is used, this method returns the value obtained by adding 20 to the XDM/RD E2 server's major version. This return value is the same as the server's major version displayed in the header of SQL traces output by the JDBC driver.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(q) getDatabaseMinorVersion()

Function

Returns the database's minor version information.

Format

```
public int getDatabaseMinorVersion() throws SQLException
```

Arguments

None.

Return value

`int` type:

Database's minor version

Functional detail

When a HiRDB server is used, this method returns the HiRDB server's minor version. For example, if the HiRDB server's version is 08-02, the method returns 2 of the `int` type.

When an XDM/RD E2 server is used, this method returns the XDM/RD E2 server's minor version. For example, if the XDM/RD E2 server's version is 11-03, the method returns 3 of the `int` type.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(r) getDatabaseProductName()

Function

Returns the connected database's product name.

When a HiRDB server is used, this method returns `HiRDB`. When an XDM/RD E2 server is used, this method returns `XDM/RD E2`.

Format

```
public String getDatabaseProductName() throws SQLException
```

Arguments

None.

Return value

`String` object

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(s) getDatabaseProductVersion()

Function

Returns the connected database's version.

Format

```
public String getDatabaseProductVersion() throws
SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

When a HiRDB server is used, this method returns the HiRDB server's version in the format *vv*-*rr* (for example: 08-00).

When an XDM/RD E2 server is used, this method returns the value obtained by adding 20 to the XDM/RD E2 server's version, in the format *vv*-*rr*. For example, if the XDM/RD E2 server's version is 11-03, the method returns 31-03.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(t) getDefaultTransactionIsolation()

Function

Returns the default transaction cut-off level.

Format

```
public int getDefaultTransactionIsolation() throws
SQLException
```

Arguments

None.

Return value

int type:

Cut-off level

Functional detail

This method always returns TRANSACTION_REPEATABLE_READ.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(u) getDriverMajorVersion()

Function

Returns this JDBC driver's major version as int type. For example, if the version is 08-00, the method returns 8 of the int type.

Format

```
public int getDriverMajorVersion() throws SQLException
```

Arguments

None.

Return value

int type:

JDBC driver's major version

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(v) getDriverMinorVersion()

Function

Returns this JDBC driver's minor version as int type. For example, if the version is 08-00, the method returns 0 of the int type.

Format

```
public int getDriverMinorVersion() throws SQLException
```

Arguments

None.

Return value

int type:

JDBC driver's minor version

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(w) getDriverName()

Function

Returns the JDBC driver name HiRDB_Type4_JDBC_Driver.

Format

```
public String getDriverName() throws SQLException
```

Arguments

None.

Return value

`String` object

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(x) getDriverVersion()

Function

Returns the version of this JDBC driver as `String`. For example, if the version is 08-00, the method returns `08-00` of the `String` type.

Format

```
public String getDriverVersion() throws SQLException
```

Arguments

None.

Return value

`String` object

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(y) getExportedKeys (String catalog,String schema,String table)

Function

Returns information about a specified table's foreign keys.

Format

```
public ResultSet getExportedKeys(String catalog,String
schema,String table) throws SQLException
```

Arguments

String catalog

Catalog name of the referencing table (this driver ignores this argument)

String schema

Schema name pattern of the referencing table (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String table

Table name pattern of the referencing table (case sensitive). If `null` or a character string with a length of 0 is specified, a table name is not used to narrow down the search.

Return value

`ResultSet` object

Functional detail

This method returns information about the specified table's foreign keys. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting #	Description
1	String	CHAR	PKTABLE_CAT	--	Catalog name of the referenced table (`NULL` value is always returned)
2	String	VARCHAR	PKTABLE_SCHEM	--	Authorization identifier of the referenced table
3	String	VARCHAR	PKTABLE_NAME	--	Table name of the referenced table
4	String	VARCHAR	PKCOLUMN_NAME	--	Column name of primary key
5	String	CHAR	FKTABLE_CAT	--	Catalog name of the referencing table (`NULL` value is always returned)
6	String	VARCHAR	FKTABLE_SCHEM	1 (ascending order)	Authorization identifier of the referencing table
7	String	VARCHAR	FKTABLE_NAME	2 (ascending order)	Table name of the referencing table
8	String	VARCHAR	FKCOLUMN_NAME	--	Column name of foreign key

Column No.	Type	SQL type (Types)	Column name	Sorting #	Description
9	short	SMALLINT	KEY_SEQ	3 (ascending order)	Sequence number of foreign key
10	short	SMALLINT	UPDATE_RULE	--	Updating rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is updated. • `importedKeyNoAction` The primary key cannot be updated. • `importedKeyCascade` The foreign key is updated by the same value as for the primary key. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be updated.
11	short	SMALLINT	DELETE_RULE	--	Deletion rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is deleted. • `importedKeyNoAction` The primary key cannot be deleted. • `importedKeyCascade` The row containing the foreign key is deleted. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be deleted.

Column No.	Type	SQL type (Types)	Column name	Sorting #	Description
12	String	VARCHAR	FK_NAME	--	Constraint name of referential constraints
13	String	VARCHAR	PK_NAME	--	Index name of the primary key
14	short	SMALLINT	DEFERRABILITY	--	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed: • `importedKeyInitiallyDeferred` Can be postponed. • `importedKeyInitiallyImmediate` Currently set for immediate evaluation, but the setting can be changed to postpone the evaluation. • `importedKeyNotDeferrable` Cannot be postponed.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(z) getExtraNameCharacters()

Function

Returns the special characters that can be used in an SQL ID name that is not enclosed in double quotation marks. The characters `a` to `z`, `A` to `Z`, `0` to `9`, and the underscore (_) are not included.

Format

```
public String getExtraNameCharacters() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `\`, `@`, and `#`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(aa) getIdentifierQuoteString()

Function

Returns the character string used to enclose SQL identifiers.

Format

```
public String getIdentifierQuoteString() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns a double-quotation mark (`"`).

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ab) getImportedKeys (String catalog,String schema,String table)

Function

Returns information about a specified table's primary key.

Format

```
public ResultSet getImportedKeys(String catalog,String
schema,String table) throws SQLException
```

Arguments

String catalog

Catalog name of the referenced table (this driver ignores this argument)

String schema

Schema name pattern of the referenced table (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String table

Table name pattern of the referenced table (case sensitive). If `null` or a character string with a length of 0 is specified, a table name is not used to narrow down the search.

Return value

`ResultSet` object

Functional detail

This method returns information about the primary key of a specified referenced table. The following table shows the format of `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	PKTABLE_CAT	--	Catalog name of the referenced table (`NULL` value is always returned)
2	String	VARCHAR	PKTABLE_SCHEM	1 (ascending order)	Authorization identifier of the referenced table
3	String	VARCHAR	PKTABLE_NAME	2 (ascending order)	Table name of the referenced table
4	String	VARCHAR	PKCOLUMN_NAME	--	Column name of primary key
5	String	CHAR	FKTABLE_CAT	--	Catalog name of the referencing table (`NULL` value is always returned)

Column No.	Type	SQL type (Types)	Column name	Sorting$^{\#}$	Description
6	String	VARCHAR	FKTABLE_SCHEM	--	Authorization identifier of the referencing table
7	String	VARCHAR	FKTABLE_NAME	--	Table name of the referencing table
8	String	VARCHAR	FKCOLUMN_NAME	--	Column name of foreign key
9	short	SMALLINT	KEY_SEQ	3 (ascending order)	Sequence number of foreign key
10	short	SMALLINT	UPDATE_RULE	--	Updating rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is updated. • `importedKeyNoAction` The primary key cannot be updated. • `importedKeyCascade` The foreign key is updated by the same value as for the primary key. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be updated.

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
11	short	SMALLINT	DELETE_RULE	--	Deletion rules. This indicates how a foreign key is affected when the foreign key is referencing the primary key and the primary key is deleted. • `importedKeyNoAction` The primary key cannot be deleted. • `importedKeyCascade` The row containing the foreign key is deleted. • `importedKeySetNull` The value of the foreign key is set to `NULL`. • `importedKeySetDefault` The value of the foreign key is set to the default value. • `importedKeyRestrict` The primary key cannot be deleted.
12	String	VARCHAR	FK_NAME	--	Constraint name of referential constraints
13	String	VARCHAR	PK_NAME	--	Index name of the primary key
14	short	SMALLINT	DEFERRABILITY	--	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed. • `importedKeyInitiallyDeferred` Can be postponed. • `importedKeyInitiallyImmediate` Currently set for immediate evaluation, but the setting can be changed to postpone the evaluation. • `importedKeyNotDeferrable` Cannot be postponed.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(ac) getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)

Function

Returns information about the indexes of a specified table.

Format

```
public ResultSet getIndexInfo(String catalog, String schema,
String table, boolean unique, boolean approximate) throws
SQLException
```

Arguments

String catalog

Catalog name (this value is not used)

String schema

Schema name pattern (case sensitive)

String table

Table name pattern (case sensitive)

boolean unique

Uniqueness attribute:

`true`: Returns information for unique indexes only.

`false`: Returns all index information, whether or not the indexes are unique.

boolean approximate

Result attribute (this value is not used)

Return value

`ResultSet` object

Functional detail

This method returns information about the indexes of a specified table. The following table shows the format of the `ResultSet` that is returned.

Table 18-39: Format of `ResultSet` returned by getIndexInf

Colu mn No.	Type	SQL type	Column name	Description
1	Strin g	CHAR	TABLE_CAT	Catalog name (`NULL` value is always returned)
2	Strin g	VARCHA R	TABLE_SCHEM	Authorization identifier
3	Strin g	VARCHA R	TABLE_NAME	Table name
4	boole an	BIT	NON_UNIQUE	If the key values for which the index is defined (total value of a single or multiple columns defined as index columns) are different in every row, the method returns `false`; otherwise, the method returns `true`.
5	Strin g	CHAR	INDEX_QUALIFIE R	Index's catalog name (`NULL` value is always returned)
6	Strin g	VARCHA R	INDEX_NAME	Index identifier
7	short	SMALLI NT	TYPE	Index type: • For cluster keys: `DatabaseMetaData.tableIndexClustered(1)` • For hash indexes: `DatabaseMetaData.tableIndexHashed(2)` • For other indexes: `DatabaseMetaData.tableIndexOther(3)`
8	short	SMALLI NT	ORDINAL_POSITI ON	For a single-column index, the method returns `1`. For a multicolumn index, the method returns the sequence number of a column that composes the index (integers beginning with 1 that identify in sequence the column names that constitute the index).
9	Strin g	VARCHA R	COLUMN_NAME	Column name
10	Strin g	VARCHA R	ASC_OR_DESC	For an index defined in ascending order, the method returns `A`; for an index defined in descending order, the method returns `D`. For a plug-in index, the method returns the `NULL` value.
11	int	INTEGER	CARDINALITY	Number of unique values in the index (`NULL` value is always returned)
12	int	INTEGER	PAGES	Number of pages used for the index (`NULL` value is always returned)

Column No.	Type	SQL type	Column name	Description
13	String	CHAR	FILTER_CONDITION	Filter condition (NULL value is always returned.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ad) getJDBCMajorVersion ()

Function

Returns the driver's JDBC major version.

Format

```
public int getJDBCMajorVersion() throws SQLException
```

Arguments

None.

Return value

int type:

JDBC driver's major version

Functional detail

This method returns the driver's JDBC major version, which is 2.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ae) getJDBCMinorVersion ()

Function

Returns the driver's JDBC minor version.

Format

```
public int getJDBCMinorVersion () throws SQLException
```

Arguments

None.

Return value

`int` type:

JDBC minor version

Functional detail

This method returns the driver's JDBC minor version, which is `1`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(af) getMaxBinaryLiteralLength()

Function

Returns the maximum number of hexadecimal characters that can be used in a binary literal.

Format

```
public int getMaxBinaryLiteralLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of characters

Functional detail

This method always returns `64000`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ag) getMaxCatalogNameLength()

Function

Returns the maximum length of a catalog name (number of characters).

Format

```
public int getMaxCatalogNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of characters permitted for a catalog name. A value of `0` means there is no limit or the limit is not known.

Functional detail

This method always returns `0`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ah) getMaxCharLiteralLength()

Function

Returns the maximum length of a character literal (number of characters).

Format

```
public int getMaxCharLiteralLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of characters permitted for a character literal. A value of `0` means there is no limit or the limit is not known.

Functional detail

This method always returns `32000`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ai) getMaxColumnNameLength()

Function

Returns the maximum length of a column name (number of characters).

Format

```
public int getMaxColumnNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of characters permitted for a column name

Functional detail

This method always returns `30`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(aj) getMaxColumnsInGroupBy()

Function

Returns the maximum number of columns in a `GROUP BY` clause.

Format

```
public int getMaxColumnsInGroupBy() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum number of columns

Functional detail

This method always returns `255`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ak) getMaxColumnsInIndex()

Function

Returns the maximum number of columns permitted for an index.

Format

```
public int getMaxColumnsInIndex() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum number of columns

Functional detail

This method always returns `16`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(al) getMaxColumnsInOrderBy()

Function

Returns the maximum number of columns in an `ORDER BY` clause.

Format

```
public int getMaxColumnsInOrderBy() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum number of columns

Functional detail

This method always returns 255.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(am)getMaxColumnsInSelect()

Function

Returns the maximum number of columns in a SELECT list.

Format

```
public int getMaxColumnsInSelect() throws SQLException
```

Arguments

None.

Return value

int type:

Permitted maximum number of columns

Functional detail

This method always returns 30000.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(an) getMaxColumnsInTable()

Function

Returns the maximum number of columns in a table.

Format

```
public int getMaxColumnsInTable() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum number of columns

Functional detail

This method always returns `30000`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ao) getMaxConnections()

Function

Returns the maximum number of concurrent connections.

Format

```
public int getMaxConnections() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of connections that can be active at the same time. A value of `0` means there is no limit or the limit is not known.

Functional detail

This method always returns `0`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ap) getMaxCursorNameLength()

Function

Returns the maximum length of a cursor name (number of characters).

Format

```
public int getMaxCursorNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of characters permitted for a cursor name

Functional detail

This method always returns `30`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(aq) getMaxIndexLength()

Function

Returns the maximum length of an index, including all parts of the index.

Format

```
public int getMaxIndexLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum length of an index

Functional detail

This method always returns `4036`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ar) getMaxProcedureNameLength()

Function

Returns the maximum length of a procedure name (number of characters).

Format

```
public int getMaxProcedureNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum length of a procedure name

Functional detail

This method always returns `30`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(as) getMaxRowSize()

Function

Returns the maximum length of a row (in bytes).

Format

```
public int getMaxRowSize() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of bytes permitted for one row. A value of `0` means there is no limit or the limit is not known.

Functional detail

This method always returns `0`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(at) getMaxSchemaNameLength()

Function

Returns the maximum length of a schema name (number of characters).

Format

```
public int getMaxSchemaNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum length of a schema name

Functional detail

When a HiRDB server is used, this method returns `8`.

When an XDM/RD E2 server is used, this method returns `30`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(au) getMaxStatementLength()

Function

Returns the maximum length of an SQL statement.

Format

```
public int getMaxStatementLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum length of an SQL statement

Functional detail

When a HiRDB server is used, this method returns `2,000,000`.

When an XDM/RD E2 server is used, this method returns `30,000`.

Note

When you are using an XDM/RD E2 server and you specify MAX SQL LENGTH in the RDB clause of a server space start control statement for Database Connection Server, you can extend the maximum SQL statement length to up to 2,000,000 bytes. However, this method will still return the default value (30,000), as if MAX SQL LENGTH had not been specified; this is because the JDBC driver does not acquire the MAX SQL LENGTH specification information from the server. However, the JDBC driver will be able to process the set maximum SQL statement length (up to 2,000,000 bytes).

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(av) getMaxStatements()

Function

Returns the maximum number of SQL statements that can be active.

Format

```
public int getMaxStatements() throws SQLException
```

Arguments

None.

Return value

int type:

Maximum number of SQL statements

Functional detail

This method always returns 4095.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(aw)getMaxTableNameLength()

Function

Returns the maximum length of a table name (number of characters).

Format

```
public int getMaxTableNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum length of a table name

Functional detail

This method always returns `30`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ax) getMaxTablesInSelect()

Function

Returns the maximum number of tables in a `SELECT` statement.

Format

```
public int getMaxTablesInSelect() throws SQLException
```

Arguments

None.

Return value

`int` type:

Maximum number of tables permitted in a `SELECT` statement

Functional detail

When a HiRDB server is used, this method returns `64`.

When an XDM/RD E2 server is used, this method returns `256`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ay) getMaxUserNameLength()

Function

Returns the maximum length of a user name (number of characters).

Format

```
public int getMaxUserNameLength() throws SQLException
```

Arguments

None.

Return value

`int` type:

Permitted maximum length of a user name

Functional detail

When a HiRDB server is used, this method returns `8`.

When an XDM/RD E2 server is used, this method returns `7`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(az) getNumericFunctions()

Function

Returns a list of the available mathematical functions (delimited by a comma).

Format

```
public String getNumericFunctions() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

When a HiRDB server is used, this method returns the following list:

`ABS,ACOS,ASIN,ATAN,ATAN2,CEILING,COS,DEGREES,EXP,FLOOR,LOG,LOG10,MOD,PI,POWER,RADIANS,ROUND,SIGN,SIN,SQRT,TAN,TRUNCATE`

When an XDM/RD E2 server is used, this method returns the following list:

ABS,CEILING,EXP,FLOOR,LOG,MOD,POWER,SQRT

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ba) getPrimaryKeys(String catalog, String schema, String table)

Function

Returns information about a specified table's primary key columns.

Format

```
public ResultSet getPrimaryKeys(String catalog, String
schema, String table) throws SQLException
```

Arguments

String catalog

Catalog name (this value is not used)

String schema

Schema name pattern (case sensitive)

String table

Table name pattern (case sensitive)

Return value

ResultSet object

Functional detail

This method returns information about a specified table's primary key columns. The following table shows the format of the ResultSet that is returned.

Table 18-40: Format of ResultSet returned by getPrimaryKeys

Column No.	Type	SQL type	Column name	Description
1	String	CHAR	TABLE_CAT	Catalog name (NULL value is always returned)
2	String	VARCHAR	TABLE_SCHEM	Authorization identifier
3	String	VARCHAR	TABLE_NAME	Table name
4	String	VARCHAR	COLUMN_NAME	Column name

Column No.	Type	SQL type	Column name	Description
5	short	SMALLINT	KEY_SEQ	Sequence number of a column that composes the index (an integer beginning with 1 that identifies in sequence the column names that constitute the index)
6	String	VARCHAR	PK_NAME	Primary index identifier

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bb) getProcedureColumns (String catalog,String schemaPattern,String procedureNamePattern, String columnNamePattern)

Function

Returns information about stored procedure parameters.

Format

```
public ResultSet getProcedureColumns(String catalog, String
schemaPattern, String procedureNamePattern, String
columnNamePattern) throws SQLException
```

Arguments

String catalog

Catalog name (this driver ignores this argument)

String schemaPattern

Schema name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String procedureNamePattern

Procedure name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a procedure name is not used to narrow down the search.

String columnNamePattern

Parameter name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a parameter name is not used to narrow down the search.

Return value

ResultSet object

Functional detail

This method returns information about the parameters of a specified stored procedure. The following table shows the format of the ResultSet that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	PROCEDURE_CAT	--	Catalog name (NULL value is always returned)
2	String	VARCHAR	PROCEDURE_SCHEM	1 (ascending order)	Authorization identifier
3	String	VARCHAR	PROCEDURE_NAME	2 (ascending order)	Procedure name
4	String	VARCHAR	COLUMN_NAME	3 (ascending order)	Parameter name

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
5	short	SMALLINT	COLUMN_TYPE	--	Parameter type: • procedureColumnUnknown Unknown • procedureColumnIn IN parameter • procedureColumnInOut INOUT parameter • procedureColumnOut OUT parameter • procedureColumnReturn Procedure return value • procedureColumnResult Result column of ResultSet In HiRDB and XDM/RD E2, neither procedureColumnReturn nor procedureColumnResult will be set. In HiRDB, procedureColumnUnknown will not be set.
6	int	INTEGER	DATA_TYPE	--	Parameter's SQL type (value defined in java.sql.Types)
7	String	VARCHAR	TYPE_NAME	--	Parameter's SQL type name (type name expressed as a character string)
8	int	INTEGER	PRECISION	--	Parameter's precision If the parameter's SQL type name is not DECIMAL, the method returns 0.
9	int	INTEGER	LENGTH	--	Parameter size
10	short	SMALLINT	SCALE	--	Parameter scaling (number of digits in the fraction part) If the parameter's SQL type name is neither DECIMAL nor TIMESTAMP, the method returns 0.
11	short	SMALLINT	RADIX	--	Radix of parameter • Approximate value: 2 • Exact value: 10 • Non-numeric value: 0

Col umn No.	Type	SQL type (Types)	Column name	Sorting#	Description
12	short	SMALLINT	NULLABLE	--	Whether the NULL value is permitted: • procedureNoNulls NULL value is not permitted. • procedureNullable NULL value is permitted. • procedureNullableUnknown Whether NULL value is permitted is unknown. The method always returns procedureNullable.
13	String	VARCHAR	REMARKS	--	Comment related to the parameter (NULL value is always returned)

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(bc) getProcedures (String catalog,String schemaPattern,String procedureNamePattern)

Function

Returns information about stored procedures.

Format

```
public ResultSet getProcedures(String catalog, String
schemaPattern, String procedureNamePattern) throws
SQLException
```

Arguments

String catalog

Catalog name (this driver ignores this argument)

String schemaPattern

Schema name pattern (case sensitive). If null or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String procedureNamePattern

Procedure name pattern (case sensitive). If null or a character string with a length of 0 is specified, a procedure name is not used to narrow down the search.

Return value

ResultSet object

Functional detail

This method returns information about stored procedures. The following table shows the format of the ResultSet that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	PROCEDURE_CAT	--	Catalog name (NULL value is always returned)
2	String	VARCHAR	PROCEDURE_SCHEM	1 (ascending order)	Authorization identifier
3	String	VARCHAR	PROCEDURE_NAME	2 (ascending order)	Procedure name
4	String	VARCHAR	RESERVE1	--	Reserved
5	String	VARCHAR	RESERVE2	--	Reserved
6	String	VARCHAR	RESERVE3	--	Reserved
7	String	VARCHAR	REMARKS	--	Description of procedure

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
8	short	SMALLINT	PROCEDURE_TYPE	--	Procedure type: • procedureResultUnknown Results might be returned. • procedureNoResult Results are not returned. • procedureReturnsResult Results are returned.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- A database access error occurred.

(bd)getProcedureTerm()

Function

Returns a word recommended for `procedure`.

Format

```
public String getProcedureTerm() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `procedure`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(be) getResultSetHoldability ()

Function

Returns the holding facility for `ResultSet` objects.

Format

```
public int getResultSetHoldability() throws SQLException
```

Arguments

None.

Return value

This method returns the holding facility for `ResultSet` objects. This indicates whether a `ResultSet` object is to be closed when the `Connection.commit` method is called:

`ResultSet.HOLD_CURSORS_OVER_COMMIT`: `ResultSet` object is not to be closed.

`ResultSet.CLOSE_CURSORS_AT_COMMIT`: `ResultSet` object is to be closed.

Functional detail

This method returns the holding facility for valid `ResultSet` objects.

If holdable cursor is enabled in the cursor operation mode (`HIRDB_CURSOR`) when a `Connection` instance is created, the method returns `ResultSet.HOLD_CURSORS_OVER_COMMIT`; if holdable cursor is disabled, the method returns `ResultSet.CLOSE_CURSORS_AT_COMMIT`.

For details about how to specify the cursor operation mode, see *18.2.2(2) User properties*.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bf) getSchemas()

Function

Returns the available schema names.

Format

```
public ResultSet getSchemas() throws SQLException
```

Arguments

None.

Return value

`ResultSet` object

Functional detail

This method returns only the names of schemas that own tables. The following table shows the format of the `ResultSet` that is returned.

Table 18-41: Format of ResultSet returned by getSchemas

Column No.	Type	SQL type	Column name	Description
1	String	VARCHAR	TABLE_SCHEM	Authorization identifier
2	String	CHAR	TABLE_CATALOG	`NULL` value is always returned.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bg)getSchemaTerm()

Function

Returns a word recommended for `schema`.

Format

```
public String getSchemaTerm() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `schema`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bh) getSearchStringEscape()

Function

Returns the character string used as the escape sequence for wildcard characters.

Format

```
public String getSearchStringEscape() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns a backslash (\).

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bi) getSQLKeywords()

Function

Returns a list (delimited by a comma) of all database-specific SQL keywords that are not SQL92 keywords.

Format

```
public String getSQLKeywords() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

When a HiRDB server is used, this method returns the following list:

ABS,ACCESS,AFTER,ALIAS,AMOUNT,ANDNOT,ANSI,ARRAY,ASSIGN,

ASYNC,AUTO,BASE,BEFORE,BINARY,BIT_AND_TEST,BLOB,BOOLEAN,

BREADTH,BTREE,BUFFER,BYTE,CALL,CHANGE,CLUSTER,COLUMNS,

COMMENT,COMPLETION,CONDITION,CONFIGURATION,CONST,

CONSTRUCTOR,CONTIGUOUS,CURAID,CYCLE,DATA,DATABASE,DAYS,

DBA,DEFER,DEMOTING,DEPTH,DEVICE,DICTIONARY,DIGITS,DIRECT,DO,

DOUBLE_PRECISION,EACH,EDIT,ELSEIF,EQUALS,ESTIMATED,

EXCLUSIVE,EXIT,EXTERN,FILE,FIX,FIXED,FLAT,FORCE,FREE,FUNCTION,

GENERAL,GET_JAVA_STORED_ROUTINE_SOURCE,HANDLER,HASH,HELP,

HEX,HiRDB,HOURS,HUGE,IDENTIFIED,IF,IGNORE,INDEX,INOUT,

IS_USER_CONTAINED_IN_HDS_GROUP,LARGE,LEAVE,LENGTH,LESS,

LIMIT,LINES,LINK,LIST,LOCATOR,LOCK,LOCKS,LOGID,LOGNAME,LONG,

LOOP,MASTER,MAXUSAGES,MCHAR,MINUTES,MOD,MODE,MODIFY,

MONTHS,MOVE,MVARCHAR,NEW,NONE,NOWAIT,NULLABLE,NVARCHAR,

OBJECT,OFF,OFFSET,OID,OLD,OPERATION,OPERATORS,OPTIMIZE,

OTHERS,OUT,OVER,OVERFLOW,OWN,PAGE,PARAMETERS,PARTITIONED,

PCTFREE,PENDANT,PIC,PICTURE,PREALLOCATED,PREFERRED,

PREORDER,PRIVATE,PROGRAM,PROTECTED,PURGE,RANDOM,RD,

RDAREA,RECOMPILE,RECOVERABLE,RECOVERY,RECURSIVE,REF,

REFERENCING,REGLIKE,RELEASE,RELEASING,RENAME,RESIGNAL,

RESTART,RETURN,RETURNS,ROLE,ROOT,ROUTINE,ROW,ROWID,

SAVEPOINT,SCALE,SCAN,SCHEMAS,SCOPE,SD,SEARCH,SECONDS,

SEGMENT,SENSITIVE,SEPARATE,SEPARATOR,SEQUENCE,SFLIKE,SH

ARE,

SHORT,SIGNAL,SIMILAR,SLOCK,SMALLFLT,SPLIT,SQLCODE_TYPE,

SQLCOUNT,SQLDA,SQLERRM,SQLERRMC,SQLERRML,SQLEXCEPTIO
N,

SQLNAME,SQL_STANDARD,SQLWARN,SQLWARNING,START,STATIC,

STOP,STOPPING,STRUCTURE,SUBSTR,SUPPRESS,SYNONYM,TEST,TE
XT,

THERE,TIMESTAMP_FORMAT,TREAT,TRIGGER,TYPE,UAMT,UBINBUF,

UCHAR,UDATE,UHANT,UHDATE,UNDER,UNIFY_2000,UNIONALL,

UNLIMITED,UNLOCK,UNTIL,USE,UTIME,UTXTBUF,VARCHAR_FORM
AT,

VARIABLE,VIRTUAL,VISIBLE,VOLATILE,VOLUME,VOLUMES,WAIT,

WHILE,WITHOUT,XLIKE,XLOCK,YEARS

When an XDM/RD E2 server is used, this method returns the following list:

ABS,ANDNOT,ARRAY,ASSIGN,AUDIT,BINARY,BLOB,BOOLEAN,CALL,

CARDINALITY,CHANGE,CLOB,CLUSTER,COMMENT,COMPRESSED,C
ORR,

COUNT,COVAR_POP,COVAR_SAMP,CUBE,CUME_DIST,CURRENT_ROL
E,

DATA,DAYS,DBA,DENSE_RANK,DIGITS,DO,EACH,ELSEIF,EVERY,

EXCLUSIVE,FALSE,FILTER,FIX,FLAT,FORCE,FUNCTION,GROUPING,H
EX,

HOURS,IDENTIFIED,IF,INDEX,INOUT,ITERATE,LABEL,LARGE,LEAVE,

LENGTH,LIMIT,LIST,LOCK,LONG,LOOP,MCHAR,MICROSECOND,

MICROSECONDS,MINUTES,MOD,MODE,MONTHS,MVARCHAR,NEW,

NONLOCAL,NOWAIT,NVARCHAR,OLD,OPTIMIZE,OUT,OVER,OVERLA
Y,

OWN,PARTITION,PCTFREE,PERCENT_RANK,PERCENTILE_CONT,

PERCENTILE_DISC,PRIVATE,PROGRAM,PROTECTED,PURGE,RANGE,

RANK,RDAREA,RDNODE,RECURSIVE,REFERENCING,REGR_AVGX,

REGR_AVGY,REGR_COUNT,REGR_INTERCEPT,REGR_R2,REGR_SLOP
E,

REGR_SXX,REGR_SXY,REGR_SYY,RELEASE,REPEAT,RESERVED,RETURN,

RETURNS,ROLLUP,ROUTINE,ROW,ROW_NUMBER,ROWID,SECONDS,

SHARE,SIGNAL,SMALLFLT,SPECIFIC,SQLCOUNT,SQLDA,SQLERRM,

SQLERRMC,SQLERRML,SQLNAME,SQLWARN,STDDEV_POP,

STDDEV_SAMP,STOPPING,SUBSTR,TIMESTAMP_FORMAT,TRIGGER,TYPE,

UNBOUNDED,UNDER,UNTIL,USER_AUDIT,USER_GROUP,USER_LEVEL,

VAR_POP,VAR_SAMP,VARCHAR_FORMAT,WAIT,WHILE,WINDOW,WITHIN,

WITHOUT,YEARS

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bj) getSQLStateType ()

Function

Returns a value indicating whether SQLSTATE returned by SQLException.getSQLState is an X/Open SQL CLI or SQL99.

Format

```
public int getSQLStateType() throws SQLException
```

Arguments

None.

Return value

int type:

SQLSTATE type, which is sqlStateXOpen or sqlStateSQL99

Functional detail

- For a HiRDB server

  If you specified YES in the PDSTANDARDSQLSTATE client environment variable (or you omitted the PDSTANDARDSQLSTATE client environment variable and you specified Y in the pd_standard_sqlstate system common definition), the method sets sqlStateSQL99 as the return value because the value for SQL2003 is set in SQLSTATE.

For other than the above, the method sets 0 as the return value because an HiRDB-specific value is set in SQLSTATE.

- For an XDM/RD E2 server

  The method sets sqlStateSQL99 as the return value.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bk) getStringFunctions()

Function

Returns a list of string functions (delimited by a comma).

Format

```
public String getStringFunctions() throws SQLException
```

Arguments

None.

Return value

String object

Functional detail

When a HiRDB server is used, this method returns the following list:

ASCII,CHAR,INSERT,LCASE,LEFT,LENGTH,LOCATE,LTRIM,REPLACE,RIGHT,RTRIM,SUBSTRING,UCASE

When an XDM/RD E2 server is used, this method returns the following list:

LCASE,LENGTH,LOCATE,LTRIM,POSITION,REPLACE,RTRIM,SUBSTRING,UCASE

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bl) getSuperTables (String catalog,String schemaPattern,String tableNamePattern)

Function

Returns a description of table hierarchies defined in a specified schema.

Format

```
public ResultSet getSuperTables(String catalog,String
schemaPattern,String tableNamePattern) throws SQLException
```

Arguments

String catalog

Catalog name

String schemaPattern

Schema name pattern

String tableNamePattern

Table name pattern

Return value

`ResultSet` object

Functional detail

This method always returns a `ResultSet` in which the number of rows resulting from the retrieval is 0 because HiRDB and XDM/RD E2 do not support table hierarchies. The method does not perform a validity check on any of the arguments. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Description
1	String	CHAR	TABLE_CAT	Catalog name
2	String	VARCHAR	TABLE_SCHEM	Authorization identifier
3	String	VARCHAR	TABLE_NAME	Type name
4	String	VARCHAR	SUPERTABLE_NAME	Super type name

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bm)getSuperTypes (String catalog,String schemaPattern,String typeNamePattern)

Function

Returns a description of user-defined-type hierarchies that are defined in a specified schema.

Format

```
public ResultSet getSuperTypes(String catalog,String
schemaPattern,String typeNamePattern) throws SQLException
```

Arguments

String catalog

Catalog name

String schemaPattern

Schema name pattern

String typeNamePattern

User-defined type name pattern

Return value

`ResultSet` object

Functional detail

This method always returns a `ResultSet` in which the number of rows resulting from the retrieval is 0 because the Type4 JDBC driver does not support user-defined types. The method does not perform a validity check on any of the arguments. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Description
1	String	CHAR	TYPE_CAT	Catalog name of the user-defined type
2	String	VARCHAR	TYPE_SCHEM	Schema name of the user-defined type
3	String	VARCHAR	TYPE_NAME	Type name of the user-defined type
4	String	CHAR	SUPERTYPE_CAT	Catalog name of the direct super type
5	String	VARCHAR	SUPERTYPE_SCHEM	Schema name of the direct super type
6	String	VARCHAR	SUPERTYPE_NAME	Type name of the direct super type

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bn)getSystemFunctions()

Function

Returns the available system functions.

Format

```
public String getSystemFunctions() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

This method always returns `USER`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bo)getTablePrivileges (String catalog,String schemaPattern,String tableNamePattern)

Function

Returns information about table access privileges.

Format

```
public ResultSet getTablePrivileges(String catalog, String
schemaPattern, String tableNamePattern)throws SQLException
```

Arguments

String catalog

Catalog name (this driver ignores this argument)

String schemaPattern

Schema name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a schema name is not used to narrow down the search.

String tableNamePattern

Table name pattern (case sensitive). If `null` or a character string with a length of 0 is specified, a table name is not used to narrow down the search.

Return value

`ResultSet` object

Functional detail

This method returns information about the access privileges for tables. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
1	String	CHAR	TABLE_CAT	--	Catalog name (`NULL` value is always returned)
2	String	VARCHAR	TABLE_SCHEM	1 (ascending order)	Authorization identifier
3	String	VARCHAR	TABLE_NAME	2 (ascending order)	Table name
4	String	VARCHAR	GRANTOR	--	User who grants access privileges
5	String	VARCHAR	GRANTEE	--	User who receives access privileges
6	String	VARCHAR	PRIVILEGE	3 (ascending order)	Names of granted access privileges (multiple access privilege names are separated by a comma): `SELECT`: `SELECT` privilege `INSERT`: `INSERT` privilege `UPDATE`: `UPDATE` privilege `DELETE`: `DELETE` privilege

Column No.	Type	SQL type (Types)	Column name	Sorting#	Description
7	String	VARCHAR	IS_GRANTABLE	--	Whether a user who has received access privileges can grant access privileges to other users: YES: User can grant privileges to other users. NO: User cannot grant privileges to other users. null: Whether the user can grant privileges to other users is unknown. In both HiRDB and XDM/RD E2, null will not be returned. If multiple access privileges are set in PRIVILEGE and at least one of the privileges can be granted to other users, the method sets YES.

Legend:

--: Not applicable

#

Priority level of the sort key column

Exceptions

If one of the following is true, the JDBC driver throws an SQLException:

- close() was executed on the Connection object before this method executed.
- A database access error occurred.

(bp)getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)

Function

Returns information about tables.

Format

```
public ResultSet getTables(String catalog, String
schemaPattern, String tableNamePattern, String[] types)
throws SQLException
```

Arguments

String catalog

Not used

String schemaPattern

Schema name pattern (case sensitive)

String tableNamePattern

Table name pattern (case sensitive)

String[] types

List of table types (case sensitive). You can specify any table types that are returned by the `getTableTypes()` method. If `null` is specified, the method assumes that `TABLE`, `VIEW`, and `SYSTEM TABLE` are all specified.

Return value

`ResultSet` object

Functional detail

This method returns information about tables. The following table shows the format of the `ResultSet` that is returned.

Table 18-42: Format of `ResultSet` returned by getTables

Column No.	Type	SQL type	Column name	Description
1	String	CHAR	TABLE_CAT	`NULL` value is always returned
2	String	VARCHAR	TABLE_SCHEM	Authorization identifier
3	String	VARCHAR	TABLE_NAME	Table name
4	String	VARCHAR	TABLE_TYPE	Table type: • `TABLE`: Base table (if `BASE TABLE` is specified in the DABroker for Java-compatible mode, the method assumes that `TABLE` is specified) • `VIEW`: View table • `SYSTEM TABLE`: Data dictionary table
5	String	VARCHAR	REMARKS	Comment
6	String	CHAR	TYPE_CAT	`NULL` value is always returned

Column No.	Type	SQL type	Column name	Description
7	String	CHAR	TYPE_SCHEM	NULL value is always returned
8	String	CHAR	TYPE_NAME	NULL value is always returned
9	String	CHAR	SELF_REFERENCING_COL_NAME	NULL value is always returned
10	String	CHAR	REF_GENERATION	NULL value is always returned

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- `close()` was executed on the `Connection` object before this method executed.
- At lease one element in the `String[] type` argument is `null`.
- At least one element in the `String[] type` argument is not one of the following character strings:

  `TABLE`, `VIEW`, `SYSTEM TABLE`, `BASE TABLE`

(bq)getTableTypes ()

Function

Returns the available table types.

Format

```
public ResultSet getTableTypes()throws SQLException
```

Arguments

None.

Return value

`ResultSet` object

Functional detail

This method returns the available table types. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type	Column name	Description
1	String	VARCHAR	TABLE_TYPE	Table type: • TABLE: Base table • VIEW: View table • SYSTEM TABLE: Data dictionary table

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(br) getTimeDateFunctions()

Function

Returns a list of the available time and date functions (delimited by a comma).

Format

```
public String getTimeDateFunctions() throws SQLException
```

Arguments

None.

Return value

`String` object

Functional detail

When a HiRDB server is used, this method returns the following list:

ADD_INTERVAL,CENTURY,CHARACTER,CURRENT_DATE,CURRENT_TIME,CURRENT_TIMESTAMP,

DATE,DATE_TIME,DAYNAME,DAYOFWEEK,DAYOFYEAR,DAY,DAYS,INTERVAL_DATETIMES,

HALF,HOUR,LAST_DAY,MIDNIGHTSECONDS,MINUTE,MONTH,MONTHNAME,MONTHS_BETWEEN,

NEXT_DAY,QUARTER,ROUNDMONTH,SECOND,TIME,TIMESTAMP,TIMESTAMP_FORMAT,TRUNCYEAR,

VARCHAR_FORMAT,WEEK,WEEKOFMONTH,YEAR,YEARS_BETWEEN

When an XDM/RD E2 server is used, this method returns the following list:

CURDATE,CURRENT_DATE,CURTIME,CURRENT_TIME,CURRENT_TI

MESTAMP,HOUR,MINUTE,MONTH,NOW,SECOND,YEAR

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bs) getTypeInfo()

Function

Returns information about the default SQL types.

Format

```
public ResultSet getTypeInfo() throws SQLException
```

Arguments

None.

Return value

`ResultSet` object

Functional detail

This method returns information about the default SQL types. The following table shows the format of the `ResultSet` that is returned.

Table 18-43: Format of `ResultSet` returned by getTypeInfo

Column No.	Type	SQL type	Column name	Description
1	String	VARCHAR	TYPE_NAME	Type name
2	short	SMALLINT	DATA_TYPE	SQL data type of `java.sql.Types`
3	int	INTEGER	PRECISION	Maximum precision
4	String	VARCHAR	LITERAL_PREFIX	Prefix used to quote literals
5	String	VARCHAR	LITERAL_SUFFIX	Suffix used to quote literals
6	String	VARCHAR	CREATE_PARAMS	Parameter used to create types

Column No.	Type	SQL type	Column name	Description
7	short	SMALLINT	NULLABLE	Whether the NULL value can be used for this type: • typeNoNulls: NULL value cannot be used. • typeNullable: NULL value can be used. • typeNullableUnknown: Whether the NULL value can be used is unknown. The method always returns typeNullableUnknown.
8	boolean	BIT	CASE_SENSITIVE	Whether the value is case sensitive: • true: Character string data types • false: Other data types
9	short	SMALLINT	SEARCHABLE	Whether WHERE can be used for this type: • typePredNone: Cannot be used. • typePredChar: Only WHERE..LIKE can be used. • typePredBasic: Values other than WHERE..LIKE can be used. • typeSearchable: All WHERE.. can be used The method always returns typeSearchable.
10	boolean	BIT	UNSIGNED_ATTRIBUTE	Whether the attribute is unsigned. The method returns false for numeric data (because it is signed) and true for other data types (because they are unsigned).
11	boolean	BIT	FIXED_PREC_SCALE	Whether this can be a currency value. The method always returns false.
12	boolean	BIT	AUTO_INCREMENT	Whether this can be used as an automatic increment value. The method always returns false.
13	String	VARCHAR	LOCAL_TYPE_NAME	The type name's localized version The method returns the same value as the type name.

Column No.	Type	SQL type	Column name	Description
14	short	SMALLINT	MINIMUM_SCALE	Supported minimum scale
15	short	SMALLINT	MAXIMUM_SCALE	Supported maximum scale
16	int	INTEGER	SQL_DATA_TYPE	NULL value is always returned
17	int	INTEGER	SQL_DATETIME_SUB	NULL value is always returned
18	int	INTEGER	NUM_PREC_RADIX	2 or 10 for numeric data, 0 for all other data.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bt) getUDTs (String catalog,String schemaPattern,String typeNamePattern,int[] types)

Function

Returns information about the user-defined types.

Format

```
public ResultSet getUDTs(String catalog, String
schemaPattern, String typeNamePattern, int[] types) throws
SQLException
```

Arguments

String catalog

Catalog name

String schemaPattern

Schema name pattern

String typeNamePattern

Type name pattern

int[] types

List of user-defined types

Return value

ResultSet object

Functional detail

This method always returns a ResultSet in which the number of rows resulting from the retrieval is 0 because the Type4 JDBC driver does not support user-defined types. The method does not perform a validity check on any of the arguments. The following table shows the format of the ResultSet that is returned:

Column No.	Type	SQL type (Types)	Column name	Description
1	String	CHAR	TYPE_CAT	Catalog name of the user-defined type
2	String	VARCHAR	TYPE_SCHEM	Schema name of the user-defined type
3	String	VARCHAR	TYPE_NAME	Type name of the user-defined type
4	String	VARCHAR	CLASS_NAME	Java class name
5	int	INTEGER	DATA_TYPE	Type value defined by java.sql.Types
6	String	VARCHAR	REMARKS	Description of the type
7	short	SMALLINT	BASE_TYPE	Type code of DISTINCT-type source or type code of an implementation of SELF_REFERENCING_COLUMN user-defined reference type

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bu)getURL()

Function

Returns the URL used for the connection to HiRDB or XDM/RD E2. If there is no URL, the method returns NULL.

Format

```
public String getURL() throws SQLException
```

Arguments

None.

Return value

String object

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bv) getUserName()

Function

Returns the user name used to connect to HiRDB or XDM/RD E2.

Format

```
public String getUserName() throws SQLException
```

Arguments

None.

Return value

String object

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(bw)getVersionColumns (String catalog,String schema,String table)

Function

Returns information about the table columns that are updated automatically when rows in the table are updated.

Format

```
public ResultSet getVersionColumns(String catalog,String
schema,String table) throws SQLException
```

Arguments

String catalog

Catalog name

String schema

Schema name

String table

Table name

Return value

`ResultSet` object

Functional detail

This method always returns a `ResultSet` in which the number of rows resulting from the retrieval is 0 because HiRDB and XDM/RD E2 do not support a function for updating columns automatically. The method does not perform a validity check on any of the arguments. The following table shows the format of the `ResultSet` that is returned:

Column No.	Type	SQL type (Types)	Column name	Description
1	short	SMALLINT	SCOPE	Not used
2	String	VARCHAR	COLUMN_NAME	Column name
3	int	INTEGER	DATA_TYPE	SQL type
4	String	VARCHAR	TYPE_NAME	Data source-dependent type name
5	int	INTEGER	COLUMN_SIZE	Data precision
6	int	INTEGER	BUFFER_LENGTH	Data length in bytes
7	short	SMALLINT	DECIMAL_DIGITS	Scale
8	short	SMALLINT	PSEUDO_COLUMN	Whether this is a pseudo column

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bx) insertsAreDetected(int type)

Function

Returns a value indicating whether insertion of a visible row can be detected by calling the `ResultSet.rowInserted()` method.

Format

```
public boolean insertsAreDetected(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Changes are detected by the specified type of result set.

false: Changes are not detected by the specified type of result set.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(by) isCatalogAtStart()

Function

Returns a value indicating whether a catalog appears at the leading (or trailing) end of a fully qualified table name.

Format

```
public boolean isCatalogAtStart() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: The catalog name appears at the beginning of a fully qualified table name.

false: The catalog name does not appear at the beginning of a fully qualified table name.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(bz) isReadOnly()

Function

Returns a value indicating whether the database is in read-only mode.

Format

```
public boolean isReadOnly() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: The database is in read-only mode.

false: The database is not in read-only mode.

Functional detail

This method always returns `false`.

Note

When you are using an XDM/RD E2 server and have used the Database Connection Server start control statement or an operation command to set the database access mode to `READ`, the database is placed in read-only mode. However, the JDBC driver assumes that the access mode is `UPDATE` because it does not acquire the access mode information from the server.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ca) locatorsUpdateCopy ()

Function

Indicates whether a change was made to a LOB copy or directly to the LOB.

Format

```
public boolean locatorsUpdateCopy() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true : Change was made to a LOB copy.

false: Change was made directly to the LOB.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cb) nullPlusNonNullIsNull()

Function

Returns a value indicating whether a join of a `NULL` value and a non-`NULL` value is treated as being `NULL`.

Format

```
public boolean nullPlusNonNullIsNull() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: A join of a `NULL` and a non-`NULL` values is treated as `NULL`.

false: A join of a `NULL` and a non-`NULL` values is not treated as `NULL`.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cc) nullsAreSortedAtEnd()

Function

Returns a value indicating whether the NULL value is sorted during termination processing (regardless of the sort order).

Format

```
public boolean nullsAreSortedAtEnd() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Sorted

false: Not sorted

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(cd) nullsAreSortedAtStart()

Function

Returns a value indicating whether the NULL value is sorted during startup processing (regardless of the sort order).

Format

```
public boolean nullsAreSortedAtStart() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Sorted

false: Not sorted

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ce) nullsAreSortedHigh()

Function

Returns a value indicating whether the `NULL` value is sorted in ascending order.

Format

```
public boolean nullsAreSortedHigh() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Sorted

false: Not sorted

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cf) nullsAreSortedLow()

Function

Returns a value indicating whether the `NULL` value is sorted in descending order.

Format

```
public boolean nullsAreSortedLow() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Sorted

false: Not sorted

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(cg) othersDeletesAreVisible(int type)

Function

Returns a value indicating whether a deletion performed externally is visible.

Format

```
public boolean othersDeletesAreVisible(int type) throws
SQLException
```

Arguments

int type

One of the following ResultSet types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

boolean type:

true: Visible

false: Not visible

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ch) othersInsertsAreVisible(int type)

Function

Returns a value indicating whether an insertion performed externally is visible.

Format

```
public boolean othersInsertsAreVisible(int type) throws
SQLException
```

Arguments

int type:

One of the following ResultSet types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

boolean type:

true: Visible

false: Not visible

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ci) othersUpdatesAreVisible(int type)

Function

Returns a value indicating whether a deletion performed externally is visible.

Format

```
public boolean othersUpdatesAreVisible(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Visible

false: Not visible

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cj) ownDeletesAreVisible(int type)

Function

Returns a value indicating whether a deletion of a result set itself is visible.

Format

```
public boolean ownDeletesAreVisible(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Visible

false: Not visible

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ck) ownInsertsAreVisible(int type)

Function

Returns a value indicating whether an insertion of a result set itself is visible.

Format

```
public boolean ownInsertsAreVisible(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Visible

false: Not visible

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cl) ownUpdatesAreVisible(int type)

Function

Returns a value indicating whether an updating of a result set itself is visible.

Format

```
public boolean ownUpdatesAreVisible(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Visible

false: Not visible

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cm)storesLowerCaseIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all lower-case letters.

Format

```
public boolean storesLowerCaseIdentifiers() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in all lower-case letters.

false: Such SQL identifiers are stored as being case sensitive.

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(cn) storesLowerCaseQuotedIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all lower-case letters.

Format

```
public boolean storesLowerCaseQuotedIdentifiers() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in all lower-case letters.

false: Other than the above.

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(co) storesMixedCaseIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in upper-case and lower-case letters.

Format

```
public boolean storesMixedCaseIdentifiers() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in upper-case and lower-case letters.

false: Other than the above.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cp) storesMixedCaseQuotedIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in upper-case and lower-case letters.

Format

```
public boolean storesMixedCaseQuotedIdentifiers() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in upper-case and lower-case letters.

false: Other than the above.

Functional detail

This method always returns true.

Note

1. Such SQL identifiers are actually processed as being case sensitive, but the method returns true because the results are stored as upper-case and lower-case letters. You can use the supportsMixedCaseQuotedIdentifiers method to determine whether such SQL identifiers are processed as being case sensitive.
2. When you use an XDM/RD E2 server and you specify YES (the default is NO) in the DELIMITED ID UPPER operand in the RD environment definition for XDM/RD E2, the method stores the SQL identifiers in upper-case letters. However, the method assumes that NO is specified because the JDBC driver does not acquire information about the DELIMITED ID UPPER operand from the server.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(cq) storesUpperCaseIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all upper-case letters.

Format

```
public boolean storesUpperCaseIdentifiers() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in all upper-case letters.

false: Other than the above.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cr) storesUpperCaseQuotedIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being not case sensitive, and then the results are stored in all upper-case letters.

Format

```
public boolean storesUpperCaseQuotedIdentifiers() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Such SQL identifiers are processed as being not case sensitive and are stored in all upper-case letters.

false: Other than the above.

Functional detail

This method always returns `false`.

Note

When you are using an XDM/RD E2 server and you specify `YES` (the default is `NO`) in the `DELIMITED ID UPPER` operand in the RD environment definition for XDM/RD E2, the method stores the SQL identifiers in upper-case letters. However, the JDBC driver assumes that `NO` (the default value) is specified because the driver cannot acquire information about the

DELIMITED ID UPPER operand.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(cs) supportsAlterTableWithAddColumn()

Function

Returns a value indicating whether ALTER TABLE with added columns is supported.

Format

```
public boolean supportsAlterTableWithAddColumn() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Supported

false: Not supported

Functional detail

This method always returns true.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ct) supportsAlterTableWithDropColumn()

Function

Returns a value indicating whether ALTER TABLE with dropped columns is supported.

Format

```
public boolean supportsAlterTableWithDropColumn() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cu) supportsANSI92EntryLevelSQL()

Function

Returns a value indicating whether the ANSI92 entry-level SQL grammar is supported.

Format

```
public boolean supportsANSI92EntryLevelSQL() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cv) supportsANSI92FullSQL()

Function

Returns a value indicating whether the ANSI92 full-level SQL grammar is supported.

Format

```
public boolean supportsANSI92FullSQL() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cw)supportsANSI92IntermediateSQL()

Function

Returns a value indicating whether the ANSI92 intermediate-level SQL grammar is supported.

Format

```
public boolean supportsANSI92IntermediateSQL() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cx) supportsBatchUpdates()

Function

Returns a value indicating whether batch updating is supported.

Format

```
public boolean supportsBatchUpdates() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cy) supportsCatalogsInDataManipulation()

Function

Returns a value indicating whether catalog names can be used in data manipulation statements.

Format

```
public boolean supportsCatalogsInDataManipulation() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Catalog names can be used.

false: Catalog names cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(cz) supportsCatalogsInIndexDefinitions()

Function

Returns a value indicating whether catalog names can be used in index definition statements.

Format

```
public boolean supportsCatalogsInIndexDefinitions() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Catalog names can be used.

false: Catalog names cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(da) supportsCatalogsInPrivilegeDefinitions()

Function

Returns a value indicating whether catalog names can be used in privilege definition statements.

Format

```
public boolean supportsCatalogsInPrivilegeDefinitions()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Catalog names can be used.

false: Catalog names cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(db)supportsCatalogsInProcedureCalls()

Function

Returns a value indicating whether catalog names can be used in procedure call statements.

Format

```
public boolean supportsCatalogsInProcedureCalls() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Catalog names can be used.

false: Catalog names cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dc) supportsCatalogsInTableDefinitions()

Function

Returns a value indicating whether catalog names can be used in table definition statements.

Format

```
public boolean supportsCatalogsInTableDefinitions() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Catalog names can be used.

false: Catalog names cannot be used.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dd) supportsColumnAliasing()

Function

Returns a value indicating whether aliases are supported for columns.

Format

```
public boolean supportsColumnAliasing() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Supported

false: Not supported

Functional detail

This method always returns true.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(de) supportsConvert()

Function

Returns a value indicating whether the CONVERT function is supported for SQL types.

Format

```
public boolean supportsConvert() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Supported

false: Not supported

Functional detail

This method always returns true.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(df) supportsConvert(int fromType, int toType)

Function

Returns a value indicating whether the CONVERT function is supported for a specified combination of SQL types.

Format

```
public boolean supportsConvert(int fromType, int toType)
throws SQLException
```

Arguments

int fromType

Conversion source type. This is a type code of the `java.sql.Types` class.

int toType

Conversion target type. This is a type code of the `java.sql.Types` class.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method returns a value that depends on the combination of the conversion source type and the conversion target type. The following tables show the return value depending on the combination of the conversion source and target types.

Table 18-44: Return value depending on the combination of the conversion source and target types (part 1)

Conversion source Type of java.sql.Types	**Conversion target Type of java.sql.Types**												
	BIT	**TINYINT**	**SMALLINT**	**INTEGER**	**BIGINT**	**FLOAT**	**REAL**	**DOUBLE**	**NUMERIC**	**DECIMAL**	**CHAR**	**VARCHAR**	**LONGVARCHAR**
BIT	1	1	1	1	1	1	1	1	1	1	1	1	1
TINYINT	1	1	1	1	1	1	1	1	1	1	1	1	1
SMALLINT	1	1	1	1	1	1	1	1	1	1	1	1	1
INTEGER	1	1	1	1	1	1	1	1	1	1	1	1	1
BIGINT	1	1	1	1	1	1	1	1	1	1	1	1	1

Conversion source Type of java.sql.Types	Conversion target Type of java.sql.Types												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
FLOAT	1	1	1	1	1	1	1	1	1	1	1	1	1
REAL	1	1	1	1	1	1	1	1	1	1	1	1	1
DOUBLE	1	1	1	1	1	1	1	1	1	1	1	1	1
NUMERIC	1	1	1	1	1	1	1	1	1	1	1	1	1
DECIMAL	1	1	1	1	1	1	1	1	1	1	1	1	1
CHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
VARCHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
LONGVARCHAR	1	1	1	1	1	1	1	1	1	1	1	1	1
DATE	0	0	0	0	0	0	0	0	0	0	1	1	1
TIME	0	0	0	0	0	0	0	0	0	0	1	1	1
TIMESTAMP	0	0	0	0	0	0	0	0	0	0	1	1	1
BINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
VARBINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
LONGVARBINARY	0	0	0	0	0	0	0	0	0	0	0	0	0
JAVA_OBJECT	0	0	0	0	0	0	0	0	0	0	0	0	0
STRUCT	0	0	0	0	0	0	0	0	0	0	0	0	0
ARRAY	0	0	0	0	0	0	0	0	0	0	0	0	0
BLOB	0	0	0	0	0	0	0	0	0	0	0	0	0
CLOB	0	0	0	0	0	0	0	0	0	0	0	0	0

Conversion source Type of java.sql.Types	Conversion target Type of java.sql.Types												
	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
REF	0	0	0	0	0	0	0	0	0	0	0	0	0

Legend:

0: false

1: true

Table 18-45: Return value depending on the combination of the conversion source and target types (part 2)

Conversion source Type of java.sql.Types	Conversion target Type of java.sql.Types											
	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	JAVA_OBJECT	STRUCT	ARRAY	BLOB	CLOB	REF
BIT	0	0	0	0	0	0	0	0	0	0	0	0
TINYINT	0	0	0	0	0	0	0	0	0	0	0	0
SMALLINT	0	0	0	0	0	0	0	0	0	0	0	0
INTEGER	0	0	0	0	0	0	0	0	0	0	0	0
BIGINT	0	0	0	0	0	0	0	0	0	0	0	0
FLOAT	0	0	0	0	0	0	0	0	0	0	0	0
REAL	0	0	0	0	0	0	0	0	0	0	0	0
DOUBLE	0	0	0	0	0	0	0	0	0	0	0	0
NUMERIC	0	0	0	0	0	0	0	0	0	0	0	0
DECIMAL	0	0	0	0	0	0	0	0	0	0	0	0

Conversion source **Type of java.sql.Types**	**Conversion target** **Type of java.sql.Types**											
	DATE	**TIME**	**TIMESTAMP**	**BINARY**	**VARBINARY**	**LONGVARBINARY**	**JAVA_OBJECT**	**STRUCT**	**ARRAY**	**BLOB**	**CLOB**	**REF**
CHAR	1	1	1	0	0	0	0	0	0	0	0	0
VARCHAR	1	1	1	0	0	0	0	0	0	0	0	0
LONGVARCHAR	1	1	1	0	0	0	0	0	0	0	0	0
DATE	1	0	1	0	0	0	0	0	0	0	0	0
TIME	0	1	0	0	0	0	0	0	0	0	0	0
TIMESTAMP	1	0	1	0	0	0	0	0	0	0	0	0
BINARY	0	0	0	1	1	1	0	0	0	0	0	0
VARBINARY	0	0	0	1	1	1	0	0	0	0	0	0
LONGVARBINARY	0	0	0	1	1	1	0	0	0	0	0	0
JAVA_OBJECT	0	0	0	0	0	0	0	0	0	0	0	0
STRUCT	0	0	0	0	0	0	0	0	0	0	0	0
ARRAY	0	0	0	0	0	0	0	0	0	0	0	0
BLOB	0	0	0	0	0	0	0	0	0	0	0	0
CLOB	0	0	0	0	0	0	0	0	0	0	0	0
REF	0	0	0	0	0	0	0	0	0	0	0	0

Legend:

`0`: `false`

`1`: `true`

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dg)supportsCoreSQLGrammar()

Function

Returns a value indicating whether the ODBC Core SQL grammar is supported.

Format

```
public boolean supportsCoreSQLGrammar() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dh)supportsCorrelatedSubqueries()

Function

Returns a value indicating whether correlated subqueries are supported.

Format

```
public boolean supportsCorrelatedSubqueries() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(di) supportsDataDefinitionAndDataManipulationTransactions()

Function

Returns a value indicating whether data definition statements and data manipulation statements are both supported in transactions.

Format

```
public boolean
supportsDataDefinitionAndDataManipulationTransactions()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

When a HiRDB server is used, this method returns `false`.

When an XDM/RD E2 server is used, this method returns `true`.

Note

When you are using an XDM/RD E2 server, you can execute a data definition statement before a transaction that executes a data manipulation statement has been completed. In such a case, the data definition statement forcibly commits the transaction.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dj) supportsDataManipulationTransactionsOnly()

Function

Returns a value indicating whether only data manipulation statements are supported in transactions.

Format

```
public boolean supportsDataManipulationTransactionsOnly()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dk) supportsDifferentTableCorrelationNames()

Function

Returns a value indicating whether the table names must be different from the correlation names when table correlation names are supported.

Format

```
public boolean supportsDifferentTableCorrelationNames()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Must be different

false: May be the same

Functional detail

When a HiRDB server is used, this method returns `false`.

When an XDM/RD E2 server is used, this method returns `true`.

Note

When you are using an XDM/RD E2 server and you specify `USE` (the default is `NOUSE`) in the `RANGE VARIABLE` operand in the RD environment definition for XDM/RD E2, you can use the same correlation name as the table name. However, this method assumes that `NOUSE` (the default value) is specified and returns `true` because the JDBC driver has not acquired information about the `RANGE VARIABLE` operand from the server. The JDBC driver itself does not require table names to be different from the correlation names.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dl) supportsExpressionsInOrderBy()

Function

Returns a value indicating whether expressions are supported in an `ORDER BY` list.

Format

```
public boolean supportsExpressionsInOrderBy() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

When a HiRDB server is used, this method returns `false`.

When an XDM/RD E2 server is used, this method returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dm)supportsExtendedSQLGrammar()

Function

Returns a value indicating whether the ODBC Extended SQL grammar is supported.

Format

```
public boolean supportsExtendedSQLGrammar() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dn)supportsFullOuterJoins()

Function

Returns a value indicating whether full outer joins are supported.

Format

```
public boolean supportsFullOuterJoins() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Supported

false: Not supported

Functional detail

This method always returns true.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(do) supportsGetGeneratedKeys ()

Function

Returns a value indicating whether automatic generation keys can be acquired after statements have executed.

Format

```
public boolean supportsGetGeneratedKeys() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Automatic generation keys can be acquired.

false: Automatic generation keys cannot be acquired.

Functional detail

This method always returns false because the Type4 JDBC driver does not support automatic generation keys.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(dp) supportsGroupBy()

Function

Returns a value indicating whether the GROUP BY clause form is supported.

Format

```
public boolean supportsGroupBy() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dq) supportsGroupByBeyondSelect()

Function

Returns a value indicating whether a column for which the `GROUP BY` clause is not specified in `SELECT` can be used when all columns in `SELECT` must be specified.

Format

```
public boolean supportsGroupByBeyondSelect() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Can be used

false: Cannot be used

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dr) supportsGroupByUnrelated()

Function

Returns a value indicating whether a column for which the `GROUP BY` clause is not specified in `SELECT` can be used.

Format

```
public boolean supportsGroupByUnrelated() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Can be used

false: Cannot be used

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ds) supportsIntegrityEnhancementFacility()

Function

Returns a value indicating whether the SQL Integrity Enhancement Facility is supported.

Format

```
public boolean supportsIntegrityEnhancementFacility()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dt) supportsLikeEscapeClause()

Function

Returns a value indicating whether escape characters are supported in the `LIKE` clause.

Format

```
public boolean supportsLikeEscapeClause() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(du)supportsLimitedOuterJoins()

Function

Returns a value indicating whether limited support is provided for outer joins.

Format

```
public boolean supportsLimitedOuterJoins() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Limited support is provided.

false: Limited support is not provided.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dv) supportsMinimumSQLGrammar()

Function

Returns a value indicating whether the ODBC Minimum SQL grammar is supported.

Format

```
public boolean supportsMinimumSQLGrammar() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dw)supportsMixedCaseIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is not enclosed in quotation marks is processed as being case sensitive, and then the results are stored in upper-case and lower-case letters.

Format

```
public boolean supportsMixedCaseIdentifiers() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Results are stored in upper-case and lower-case letters.

false: Results are not stored using both upper-case and lower-case letters.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dx) supportsMixedCaseQuotedIdentifiers()

Function

Returns a value indicating whether an SQL identifier containing upper-case and lower-case letters that is enclosed in quotation marks is processed as being case sensitive, and then the results are stored in upper-case and lower-case letters.

Format

```
public boolean supportsMixedCaseQuotedIdentifiers() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Results are stored in upper-case and lower-case letters.

false: Results are not stored using both upper-case and lower-case letters.

Functional detail

This method always returns true.

Note

When you are using an XDM/RD E2 server and you specify YES (the default is NO) in the DELIMITED ID UPPER operand in the RD environment definition for XDM/RD E2, the method stores SQL identifiers in upper-case letters. However, the JDBC driver assumes that NO (the default value) is specified because the JDBC driver has not acquired information about the DELIMITED ID UPPER operand from the server.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(dy) supportsMultipleOpenResults ()

Function

Returns a value indicating whether it is possible for multiple ResultSet objects to be returned simultaneously by a CallableStatement object.

Format

```
public boolean supportsMultipleOpenResults() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: A CallableStatement object can return multiple ResultSet objects simultaneously.

false: A CallableStatement object cannot return multiple ResultSet objects simultaneously.

Functional detail

When a HiRDB server is used, this method returns `true`.

When an XDM/RD E2 server is used, this method returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(dz) supportsMultipleResultSets()

Function

Returns a value indicating whether multiple `ResultSet` objects can be acquired from execution of a single `execute` method.

Format

```
public boolean supportsMultipleResultSets() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Multiple `ResultSet` objects can be acquired.

false: Multiple `ResultSet` objects cannot be acquired.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ea) supportsMultipleTransactions()

Function

Returns a value indicating whether multiple transactions can be open at the same time (for different connections).

Format

```
public boolean supportsMultipleTransactions() throws
```

```
SQLException
```

Arguments

None.

Return value

`boolean` type:

true Multiple transactions can be open at the same time.

false: Multiple transactions cannot be open at the same time.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(eb) supportsNamedParameters ()

Function

Returns a value indicating whether named parameters are supported for the `callable` statement.

Format

```
public boolean supportsNamedParameters() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ec) supportsNonNullableColumns()

Function

Returns a value indicating whether columns can be defined as non-null columns.

Format

```
public boolean supportsNonNullableColumns() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Columns can be defined as non-null columns.

false: Columns cannot be defined as non-null columns.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ed) supportsOpenCursorsAcrossCommit()

Function

Returns a value indicating whether the cursor can remain open between commit operations.

Format

```
public boolean supportsOpenCursorsAcrossCommit() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: The cursor can remain open between commit operations.

false: The cursor cannot remain open between commit operations.

Functional detail

If a holdable cursor is enabled in the cursor operation mode (`HIRDB_CURSOR`) when a `Connection` instance is created, the method returns `true`; if a holdable cursor is disabled, the method returns `false`. For details about how to specify the cursor operation mode, see *18.2.2(2) User properties*.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ee) supportsOpenCursorsAcrossRollback()

Function

Returns a value indicating whether the cursor can remain open between rollback operations.

Format

```
public boolean supportsOpenCursorsAcrossRollback() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: The cursor can remain open between rollback operations.

false: The cursor cannot remain open between rollback operations.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ef) supportsOpenStatementsAcrossCommit()

Function

Returns a value indicating whether statements can remain open between commit operations.

Format

```
public boolean supportsOpenStatementsAcrossCommit() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Statements can remain open between commit operations.

false: Statements cannot remain open between commit operations.

Functional detail

If you specify `TRUE` in `STATEMENT_COMMIT_BEHAVIOR` in the URL specification items during `Connection` instance creation or if you omit `STATEMENT_COMMIT_BEHAVIOR`, the method returns `true`. If you specify `FALSE` in `STATEMENT_COMMIT_BEHAVIOR`, the method returns `false`. For details about how to specify the `STATEMENT_COMMIT_BEHAVIOR` URL specification item, see *18.2.2(2) User properties*.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(eg) supportsOpenStatementsAcrossRollback()

Function

Returns a value indicating whether statements can remain open between rollback operations.

Format

```
public boolean supportsOpenStatementsAcrossRollback()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Statements can remain open between rollback operations.

false: Statements cannot remain open between rollback operations.

Functional detail

If you specify TRUE in STATEMENT_COMMIT_BEHAVIOR in the URL specification items during Connection instance creation or if you omit STATEMENT_COMMIT_BEHAVIOR, the method returns true. If you specify FALSE in STATEMENT_COMMIT_BEHAVIOR, the method returns false. For details about how to specify the STATEMENT_COMMIT_BEHAVIOR URL specification item, see *18.2.2(2) User properties*.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(eh) supportsOrderByUnrelated()

Function

Returns a value indicating whether a column for which the ORDER BY clause is not in SELECT can be used.

Format

```
public boolean supportsOrderByUnrelated() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Can be used

false: Cannot be used

Functional detail

This method always returns true.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ei) supportsOuterJoins()

Function

Returns a value indicating whether some form of outer join is supported.

Format

```
public boolean supportsOuterJoins() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ej) supportsPositionedDelete()

Function

Returns a value indicating whether positioned `DELETE` is supported.

Format

```
public boolean supportsPositionedDelete() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(ek) supportsPositionedUpdate()

Function

Returns a value indicating whether positioned UPDATE is supported.

Format

```
public boolean supportsPositionedUpdate() throws
SQLException
```

Arguments

None.

Return value

boolean type:

true: Supported

false: Not supported

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(el) supportsResultSetConcurrency(int type, int concurrency)

Function

Returns a value indicating whether the combination of a specified type of ResultSet and a specified parallel processing type is supported.

Format

```
public boolean supportsResultSetConcurrency(int type, int
concurrency) throws SQLException
```

Arguments

int type

Type defined in java.sql.ResultSet for the result set type int concurrency

Type defined by `java.sql.ResultSet` as the parallel processing type

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

When `type` is `TYPE_FORWARD_ONLY` or `TYPE_SCROLL_INSENSITIVE` and `concurrency` is `CONCUR_READ_ONLY`, the method returns `true`; otherwise, the method returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(em)supportsResultSetHoldability (int holdability)

Function

Returns a value indicating whether the holding facility is supported for a specified `ResultSet` object.

Format

```
public boolean supportsResultSetHoldability(int
holdability) throws SQLException
```

Arguments

int holdability

Literal

ResultSet.HOLD_CURSORS_OVER_COMMIT: The `ResultSet` object is not closed when the `Connection.commit` method is called.

ResultSet.CLOSE_CURSORS_AT_COMMIT: The `ResultSet` object is closed when the `Connection.commit` method is called.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method returns a value indicating whether the holding facility is supported for a specified `ResultSet` object. The method returns the following values:

holdability argument	**HIRDB_CURSOR**[#]	
	Enabled	**Disabled**
HOLD_CURSORS_OVER_COMMIT	Returns `true`	Returns `false`
CLOSE_CURSORS_AT_COMMIT	Returns `false`	Returns `true`

#

Cursor operation mode (`HIRDB_CURSOR`) when a `Connection` instance is created.

For details about how to specify the cursor operation mode, see *18.2.2(2) User properties*.

Exceptions

If one of the following is true, the JDBC driver throws an `SQLException`:

- `close()` was executed on the `Connection` object before this method executed.
- The `holdability` argument value is invalid.

(en) supportsResultSetType(int type)

Function

Returns a value indicating whether a specified type of `ResultSet` is supported.

Format

```
public boolean supportsResultSetType(int type) throws
SQLException
```

Arguments

int type

Type of result set defined in `java.sql.ResultSet`

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

When `type` is `TYPE_FORWARD_ONLY` or `TYPE_SCROLL_INSENSITIVE`, the method returns `true`; otherwise, the method returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(eo) supportsSavepoints ()

Function

Returns a value indicating whether save points are supported.

Format

```
public boolean supportsSavepoints() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false` because the JDBC driver does not support save points in HiRDB or XDM/RD E2.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ep) supportsStatementPooling ()

Function

Returns a value indicating whether statement pooling is supported.

Format

```
public boolean supportsStatementPooling() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false` because the Type4 JDBC driver does not support statement pooling.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(eq) supportsSchemasInDataManipulation()

Function

Returns a value indicating whether schema names can be used in data manipulation statements.

Format

```
public boolean supportsSchemasInDataManipulation() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Schema names can be used.

false: Schema names cannot be used.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(er) supportsSchemasInIndexDefinitions()

Function

Returns a value indicating whether schema names can be used in index definition statements.

Format

```
public boolean supportsSchemasInIndexDefinitions() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Schema names can be used.

false: Schema names cannot be used.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(es) supportsSchemasInPrivilegeDefinitions()

Function

Returns a value indicating whether schema names can be used in privilege definition statements.

Format

```
public boolean supportsSchemasInPrivilegeDefinitions()
throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Schema names can be used.

false: Schema names cannot be used.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(et) supportsSchemasInProcedureCalls()

Function

Returns a value indicating whether schema names can be used in procedure calls.

Format

```
public boolean supportsSchemasInProcedureCalls() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Schema names can be used.

false: Schema names cannot be used.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(eu) supportsSchemasInTableDefinitions()

Function

Returns a value indicating whether schema names can be used in table definition statements.

Format

```
public boolean supportsSchemasInTableDefinitions() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Schema names can be used.

false: Schema names cannot be used.

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ev) supportsSelectForUpdate()

Function

Returns a value indicating whether `SELECT` is supported for updating.

Format

```
public boolean supportsSelectForUpdate() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ew)supportsStoredProcedures()

Function

Returns a value indicating whether stored procedure calls are supported.

Format

```
public boolean supportsStoredProcedures() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ex) supportsSubqueriesInComparisons()

Function

Returns a value indicating whether subqueries are supported in comparison expressions.

Format

```
public boolean supportsSubqueriesInComparisons() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ey) supportsSubqueriesInExists()

Function

Returns a value indicating whether subqueries are supported in `exists` expressions.

Format

```
public boolean supportsSubqueriesInExists() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ez) supportsSubqueriesInIns()

Function

Returns a value indicating whether subqueries are supported in `in` statements.

Format

```
public boolean supportsSubqueriesInIns() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fa) supportsSubqueriesInQuantifieds()

Function

Returns a value indicating whether subqueries are supported in quantified expressions.

Format

```
public boolean supportsSubqueriesInQuantifieds() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fb) supportsTableCorrelationNames()

Function

Returns a value indicating whether table correlation names are supported.

Format

```
public boolean supportsTableCorrelationNames() throws
SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fc) supportsTransactionIsolationLevel(int level)

Function

Returns a value indicating whether a specified transaction isolation level is supported.

Format

```
public boolean supportsTransactionIsolationLevel(int level)
throws SQLException
```

Arguments

`int level`

Transaction cut-off level defined in `java.sql.Connection`

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

When `level` is `TRANSACTION_REPEATABLE_READ`, `TRANSACTION_READ_COMMITTED`, or `TRANSACTION_READ_UNCOMMITTED`, the method returns `true`; otherwise, the method returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fd) supportsTransactions()

Function

Returns a value indicating whether transactions are supported.

Format

```
public boolean supportsTransactions() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fe) supportsUnion()

Function

Returns a value indicating whether `SQL UNION` is supported.

Format

```
public boolean supportsUnion() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(ff) supportsUnionAll()

Function

Returns a value indicating whether `SQL UNION ALL` is supported.

Format

```
public boolean supportsUnionAll() throws SQLException
```

Arguments

None.

Return value

`boolean` type:

true: Supported

false: Not supported

Functional detail

This method always returns `true`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fg) updatesAreDetected(int type)

Function

Returns a value indicating whether updating performed on a `ResultSet` of a specified `ResultSet` type can be detected by the `ResultSet.rowUpdated` method.

Format

```
public boolean updatesAreDetected(int type) throws
SQLException
```

Arguments

int type

One of the following `ResultSet` types:

- ResultSet.TYPE_FORWARD_ONLY
- ResultSet.TYPE_SCROLL_INSENSITIVE
- ResultSet.TYPE_SCROLL_SENSITIVE

Return value

`boolean` type:

true: Can be detected.

false: Cannot be detected.

Functional detail

This method always returns `false`.

Exceptions

If `close()` has been executed on the `Connection` object before this method is executed, the JDBC driver throws an `SQLException`.

(fh) usesLocalFilePerTable()

Function

Returns a value indicating whether a file is to be used for each table.

Format

```
public boolean usesLocalFilePerTable() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Used

false: Not used

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(fi) usesLocalFiles()

Function

Returns a value indicating whether tables are to be stored in local files.

Format

```
public boolean usesLocalFiles() throws SQLException
```

Arguments

None.

Return value

boolean type:

true: Tables are to be stored in local files.

false: Tables are not to be stored in local files.

Functional detail

This method always returns false.

Exceptions

If close() has been executed on the Connection object before this method is executed, the JDBC driver throws an SQLException.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: `PrdbDatabaseMetaData`

18.4.8 ResultSetMetaData interface

(1) Overview

The `ResultSetMetaData` interface provides the following principal function:

- Return of meta information, such as the data type and the data length, for each column in the result set.

(2) Methods

The table below lists the methods of the `ResultSetMetaData` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 18-46: ResultSetMetaData interface methods

Subsection	Method	Function
(a)	getCatalogName(int column)	Returns the catalog name for a specified column number of a table.
(b)	getColumnClassName(int column)	Returns the fully specified Java class name for a specified column.
(c)	getColumnCount()	Returns the number of columns in this `ResultSet` object.
(d)	getColumnDisplaySize(int column)	Returns a specified column's maximum width (in characters).
(e)	getColumnLabel(int column)	Returns a recommended print or display title for a specified column.
(f)	getColumnName(int column)	Returns a specified column's column name.
(g)	getColumnType(int column)	Returns a specified column's SQL data type.
(h)	getColumnTypeName(int column)	Returns a specified column's database-specific format name.
(i)	getPrecision(int column)	Returns a specified column's length in decimal digits.
(j)	getScale(int column)	Returns the number of decimal places in a specified column.
(k)	getSchemaName(int column)	Returns a specified column's table schema.
(l)	getTableName(int column)	Returns the table name for a specified column.

Subsection	Method	Function
(m)	isAutoIncrement(int column)	Returns a value indicating whether a specified column is both numbered automatically and treated as being read-only.
(n)	isCaseSensitive(int column)	Returns a value indicating whether a specified column is case sensitive.
(o)	isCurrency(int column)	Returns a value indicating whether a specified column is for currency values.
(p)	isDefinitelyWritable(int column)	Returns a value indicating whether write operations on a specified column will be successful.
(q)	isNullable(int column)	Returns a value indicating whether the NULL value can be set in a specified column.
(r)	isReadOnly(int column)	Returns a value indicating whether a specified column's value is read-only.
(s)	isSearchable(int column)	Returns a value indicating whether a specified column can be used in a where section.
(t)	isSigned(int column)	Returns a value indicating whether a specified column is for signed numeric values.
(u)	isWritable(int column)	Returns a value indicating whether write operations on a specified column can be successful.

(a) getCatalogName(int column)

Function

Returns the catalog name for a specified column number of a table.

Format

```
public synchronized String getCatalogName(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`String` object

Functional detail

This method always returns `null`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(b) getColumnClassName(int column)

Function

Returns the fully specified Java class name for a specified column.

Format

```
public synchronized String getColumnClassName(int column)
throws SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`String` object

Functional detail

This method returns the result of executing the `ResultSet#getObject` method on a column as the `String` Java class type. The following table shows the return value depending on the column's data type.

Table 18-47: Character strings returned by getColumnClassName

Column's HiRDB data type	Character string returned
BLOB	"java.lang.Object"
BINARY	"java.lang.Object"
INTEGER	"java.lang.Integer"
SMALLINT	"java.lang.Integer"
FLOAT DOUBLE PRECISION	"java.lang.Double"

Column's HiRDB data type	Character string returned
SMALLFLT REAL	"java.lang.Float"
DECIMAL NUMERIC	"java.math.BigDecimal"
CHAR	"java.lang.String"
MCHAR	"java.lang.String"
NCHAR	"java.lang.String"
VARCHAR	"java.lang.String"
MVARCHAR	"java.lang.String"
NVARCHAR	"java.lang.String"
DATE	"java.sql.Date"
TIME	"java.sql.Time"
TIMESTAMP	"java.sql.Timestamp"
BOOLEAN (column found only in a ResultSet created from DatabaseMetaData)	"java.lang.Boolean"

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(c) getColumnCount()

Function

Returns the number of columns in this ResultSet object.

Format

```
public synchronized int getColumnCount()
```

Arguments

None.

Return value

int type:

Number of columns

Exceptions

None.

(d) getColumnDisplaySize(int column)

Function

Returns a specified column's maximum width (in characters)

Format

```
public synchronized int getColumnDisplaySize(int column)
throws SQLException
```

Arguments

int column

Column number beginning at 1

Return value

`int` type:

Maximum number of characters

Functional detail

This method returns the maximum width of a specified column in characters. The following table shows the maximum width (in number of characters) returned by `getColumnDisplaySize`.

Table 18-48: Maximum width (in number of characters) returned by getColumnDisplaySize

Column's HiRDB data type	Maximum width returned (number of characters)

BLOB(n)
BINARY(n)
	n

INTEGER	11
SMALLINT	6
FLOAT DOUBLE PRECISION	23
SMALLFLT REAL	13

Column's HiRDB data type	Maximum width returned (number of characters)

DECIMAL(n,m)
NUMERIC(n,m)
	n + 2

CHAR(n)
MCHAR(n)
NCHAR(n)
VARCHAR(n)
MVARCHAR(n)
NVARCHAR(n)
	n

DATE	10
TIME	8
TIMESTAMP(n)	*n* > 0: 19 + (*n* + 1) *n* = 0: 19
`BOOLEAN` (column found only in a `ResultSet` created from `DatabaseMetaData`)	5

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(e) getColumnLabel(int column)

Function

Returns a recommended print or display title for a specified column.

This method returns the column name, which is the same value returned by `PrdbResultSetMetaData#getColumnName`. For details, see *(f) getColumnName(int column)*.

Format

```
public synchronized String getColumnLabel(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`String` object

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(f) getColumnName(int column)

Function

Returns a specified column's column name.

Format

```
public synchronized String getColumnName(int column) throws
SQLException
```

Arguments

int column

Column number beginning at 1

Return value

`String` object

Functional detail

This method returns the applicable contents from the HiRDB Column Name Descriptor Area (`SQLCNDA`) as the name of the specified column. The method acquires the name from `SQLNAME` in `SQLCNDA` that is sent from the HiRDB server to this driver (and converts the name to Java's internal code) and then returns the result. Note that if a set function, value expression, or `WRITE` specification is used in the `SELECT` statement, only the first 28 bytes are returned and any additional bytes are discarded during communication between server and client. This driver ignores any of the following characters at the beginning of a name that is received from the HiRDB server:

- Δ Δ
- Δ ■

Legend: Δ: Single-byte space, ■: `0xff`

For details about this method's return value, see Appendix *C.1 Organization and contents of the Column Name Descriptor Area*.

The first column of a `ResultSet` class acquired from an `Array` object is created by the JDBC driver and `JDBC_Array_Index` is returned as its column name.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(g) getColumnType(int column)

Function

Returns a specified column's SQL data type. For the correspondence between the column data types and the return values, see *18.8.1 Mapping SQL data types*.

Format

```
public synchronized int getColumnType(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`int` type:

SQL type from `java.sql.Types`

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(h) getColumnTypeName(int column)

Function

Returns a specified column's database-specific format name.

Format

```
public synchronized String getColumnTypeName(int column)
throws SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

String object

Functional detail

This method returns a specified column's database-specific format name. The following table shows the column data types and return values.

Table 18-49: Character strings returned by getColumnTypeName

Column's HiRDB data type	Character string returned
BLOB	"BLOB"
BINARY	"BINARY"
INTEGER	"INTEGER"
SMALLINT	"SMALLINT"
FLOAT	"FLOAT"
REAL	"REAL"
DECIMAL	"DECIMAL"
CHAR	"CHAR"
MCHAR	"MCHAR"
NCHAR	"NCHAR"
VARCHAR	"VARCHAR"
MVARCHAR	"MVARCHAR"
NVARCHAR	"NVARCHAR"
DATE	"DATE"
TIME	"TIME"
TIMESTAMP	"TIMESTAMP"
BOOLEAN (column found only in a ResultSet created from DatabaseMetaData)	"BOOLEAN"

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(i) getPrecision(int column)

Function

Returns a specified column's length in decimal digits.

Format

```
public synchronized int getPrecision(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`int` type:

Column length in decimal digits

Functional detail

If the specified column has a numeric data type (`INTEGER`, `SMALLINT`, `FLOAT`, `DOUBLE PRECISION`, `SMALLFLT`, `REAL`, `DECIMAL`, or `NUMERIC`), this method returns the number of decimal digits. If it does not have a numeric data type, the method returns the column length in bytes. The following table shows the return value of `getPrecision`.

Table 18-50: Return value of getPrecision

Column's HiRDB data type	Return value
BLOB(n), BINARY(n)	n
INTEGER	10
SMALLINT	5
FLOAT, DOUBLE PRECISION	15
SMALLFLT, REAL	7
DECIMAL(n,m), NUMERIC(n,m)	n
CHAR(n), MCHAR(n), VARCHAR(n), MVARCHAR(n)	n
NCHAR(n), NVARCHAR(n)	*n* x 2
DATE	10
TIME	8

Column's HiRDB data type	Return value
TIMESTAMP(n)	*n* > 0: 19 + (*n* + 1) *n* = 0: 19
BOOLEAN (column found only in a ResultSet created from DatabaseMetaData)	1

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(j) getScale(int column)

Function

Returns the number of decimal places in a specified column.

Format

```
public synchronized int getScale(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

int type:

Column's number of decimal places

Functional detail

This method returns the number of decimal places in a specified column. The following table shows the return value depending on the data type.

Table 18-51: Value returned by getScale

Column's HiRDB data type	Return value
DECIMAL NUMERIC	Decimal places
TIMESTAMP	Number of digits below a millisecond
Other	0

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(k) getSchemaName(int column)

Function

Returns a specified column's table schema.

Format

```
public synchronized String getSchemaName(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`String` object

Functional detail

This method always returns `null`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(l) getTableName(int column)

Function

Returns the table name for a specified column.

Format

```
public synchronized String getTableName(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

String object

Functional detail

This method always returns null.

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(m) isAutoIncrement(int column)

Function

Returns a value indicating whether a specified column is both numbered automatically and treated as being read-only.

Format

```
public synchronized boolean isAutoIncrement(int column)
throws SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

boolean type:

true: The specified column is numbered automatically and is treated as being read-only.

false: The specified column is not numbered automatically or is not treated as being read-only.

Functional detail

This method always returns false.

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(n) isCaseSensitive(int column)

Function

Returns a value indicating whether a specified column is case sensitive.

Format

```
public synchronized boolean isCaseSensitive(int column)
throws SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: The column is case sensitive.

false: The column is not case sensitive.

Functional detail

This method always returns `false`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(o) isCurrency(int column)

Function

Returns a value indicating whether a specified column is for currency values.

Format

```
public synchronized boolean isCurrency(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: The column is for currency values.

false: The column is not for currency values.

Functional detail

This method always returns `false`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(p) isDefinitelyWritable(int column)

Function

Returns a value indicating whether write operations on a specified column will be successful.

Format

```
public synchronized boolean isDefinitelyWritable(int column)
throws SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: Write operations on the column will be successful.

false: Write operations on the column might not be successful.

Functional detail

This method always returns `false`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(q) isNullable(int column)

Function

Returns a value indicating whether the `NULL` value can be set in a specified column.

Format

```
public synchronized int isNullable(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`int` type:

ResultSetMetaData.columnNoNulls: The `NULL` value cannot be set.

ResultSetMetaData.columnNullable: The `NULL` value can be set.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(r) isReadOnly(int column)

Function

Returns a value indicating whether a specified column's value is read-only.

Format

```
public synchronized boolean isReadOnly(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: The specified column's value is read-only.

false: The specified column's value might not be read-only.

Functional detail

This method always returns `false`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(s) isSearchable(int column)

Function

Returns a value indicating whether a specified column can be used in a where section.

Format

```
public synchronized boolean isSearchable(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

boolean type:

true: The column can be used in a where section.

false: The column cannot be used in a where section.

Functional detail

If the ResultSet was created from DatabaseMetaData, the method returns false.

If the ResultSet was not created from DatabaseMetaData, the method returns true.

Exceptions

If the specified column value is 0 or less or is greater than the number of columns, the JDBC driver throws an SQLException.

(t) isSigned(int column)

Function

Returns a value indicating whether a specified column is for signed numeric values.

Format

```
public synchronized boolean isSigned(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: The column is for signed numeric values.

false: The column is not for signed numeric values.

Functional detail

This method returns a value indicating whether a specified column is for signed numeric values. The following table shows the return value depending on the column's data type.

Table 18-52: Value returned by isSigned

Column's HiRDB data type	Return value
INTEGER SMALLINT FLOAT DOUBLE PRECISION REAL SMALLFLT DECIMAL NUMERIC	true
Other	false

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(u) isWritable(int column)

Function

Returns a value indicating whether write operations on a specified column can be successful.

Format

```
public synchronized boolean isWritable(int column) throws
SQLException
```

Arguments

int column

Column number (beginning with 1)

Return value

`boolean` type:

true: Write operations on the column can be successful.

false: Write operations on the column cannot be successful.

Functional detail

This method always returns `false`.

Exceptions

If the specified `column` value is 0 or less or is greater than the number of columns, the JDBC driver throws an `SQLException`.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbResultSetMetaData`

(4) Notes

(a) getColumnName and getColumnLabel methods

The `getColumnName` and `getColumnLabel` methods get retrieval item names from `SQLNAME` in the Column Name Descriptor Area (`SQLCNDA`) that the HiRDB driver sends to the JDBC driver. The methods then convert the names to Java internal codes and return them. For a description of the return values of these methods for specified columns, see *C.1 Organization and contents of the Column Name Descriptor Area*.

18.4.9 Blob interface

(1) Overview

The `Blob` interface provides the following principal functions:

- Acquisition of binary data
- Acquisition of the length of binary data
- Acquisition of the pattern-matching position

The JDBC driver uses the `PrdbBlob` class to install the `Blob` interface.

The JDBC driver generates `PrdbBlob` class objects as return values of the `getBlob` method of `ResultSet` and `CallableStatement`.

(2) Methods

The table below lists the methods of the `Blob` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-53: Blob interface methods

Subsectio n	Method	Function
(a)	getBinaryStream()	Returns a BLOB or BINARY value as a stream (PrdbDataStream object).
(b)	getBytes(long pos, int length)	Returns all or part of a BLOB or BINARY value as a byte array.
(c)	length()	Returns the length (in bytes) of the BLOB or BINARY value specified by this PrdbBlob object.
(d)	position(Blob pattern, long start)	Returns the byte location where pattern begins within the BLOB or BINARY value specified by the PrdbBlob object.
(e)	position(byte[] pattern, long start)	Returns the byte location where pattern begins within the BLOB value indicated by this Blob object.

(a) getBinaryStream()

Function

Returns a BLOB or BINARY value as a stream (PrdbDataStream object).

Format

```
public InputStream getBinaryStream() throws SQLException
```

Arguments

None.

Return value

Returns the PrdbDataStream from which InputStream is derived.

Functional detail

This method returns a BLOB or BINARY value as a stream (PrdbDataStream object).

If the locator facility is used, the method acquires a locator from a PrdbResultSet object and passes that locator to a PrdbDataStream class constructor to create a PrdbDataStream object.

When the locator facility is used, the method issues a data acquisition request to the HiRDB server each time data acquisition is requested because the PrdbDataStream object does not contain a BLOB or BINARY value.

If the locator facility is not used, the method acquires a BLOB or BINARY value from the PrdbResultSet object and passes that value to a PrdbDataStream class constructor to create a PrdbDataStream object. In this case, the method does not issue a data acquisition request to the HiRDB server because the PrdbResultSet object contains the BLOB or BINARY value.

Exceptions

The JDBC driver throws an SQLException in the following cases:

- The locator facility is used, but the PrdbConnection, PrdbStatement, or PrdbResultSet object related to this PrdbBlob object is closed.
- The PrdbBlob object has become invalid due to transaction settlement.
- When the locator facility is used, data acquisition failed due to a communication error.

(b) getBytes(long pos,int length)

Function

Returns all or part of a BLOB or BINARY value as a byte array. The method stores in the byte array as many consecutive bytes as is specified in length beginning at the location specified in pos.

Format

```
public byte[] getBytes(long pos,int length) throws
SQLException
```

Arguments

long pos

Location of the first byte to be extracted from the BLOB value (sequence number)

The location of byte 1 is 1.

int length

Number of consecutive bytes to be copied

Return value

Array containing as many consecutive BLOB or BINARY value bytes as specified in length beginning at the location specified in pos

Functional detail

This method returns all or part of a `BLOB` or `BINARY` value as a byte array. In the byte array, the method stores as many consecutive bytes as is specified in `length` beginning at the location specified in `pos`.

The following table shows the data that is returned depending on the values of `pos` and `length`:

pos	**length**	**Real length (X)#1**	**Limit value#2 and length**	**Limit value#2 and real length (X)#1**	**Data that is returned**
1 <= `pos` <= real length (*X*)#1	< 0	--	--	--	SQLException
	> = 0	< = `length`	--	*Y* < limit value	`BLOB` or `BINARY` value with the real length (*Y*)
			--	*Y* > = limit value	`BLOB` or `BINARY` value with the limit value length
		Y > length	length < limit value	--	`BLOB` or `BINARY` value with the length specified for `length`
			length > = limit value	--	`BLOB` or `BINARY` value with the limit value length
> real length (*X*)#1	--	--	--	--	Data with a length of 0
Other	--	--	--	--	`SQLException`

Legend:

--: Not applicable

#1

Real length (*X*) indicates the real length of the `BLOB` or `BINARY` value that can be acquired (*real length of BLOB or BINARY value* - *pos argument value* + 1).

#2

The limit value corresponds to `MaxFieldSize` or `HiRDB_for_JAVA_MAX_BINARY_SIZE`. The following table shows the value that becomes the limit value:

MaxFieldSize	HiRDB_for_Java_MAX_BINARY_SIZE	HiRDB data type	Limit value (maximum length of data that can be acquired)
0 (default)	0 (default)	All data types	Definition length (default)
0 (default)	> 0 (specified)	BLOB or BINARY type	HiRDB_for_Java_MAX_BINARY_SIZE
		Type other than BLOB or BINARY#	Definition length (default)
> 0 (specified)	0 (default)	All data types	MaxFieldSize
> 0 (specified)	> 0 (specified)	All data types	MaxFieldSize

#

Any character type (the HiRDB data types CHAR, VARCHAR, NCHAR, NVARCHAR, MCHAR, and MVARCHAR)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- The pos argument value is less than 1 or the length argument value is less than 0.
- The PrdbBlob object has become invalid due to transaction settlement.
- The locator facility is used, but the PrdbConnection, PrdbStatement, or PrdbResultSet object related to this PrdbBlob object is closed.
- When the locator facility is used, data acquisition failed due to a communication error.

(c) length()

Function

Returns the length (in bytes) of the BLOB or BINARY value specified by this PrdbBlob object.

Format

```
public long length() throws SQLException
```

Arguments

None.

Return value

Length of BLOB or BINARY value (number of bytes)

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This PrdbDataStream class is closed.
- The locator facility is used, but the PrdbConnection, PrdbStatement, or PrdbResultSet object related to the PrdbDataStream object is closed.
- PrdbDataStream has become invalid due to transaction settlement.
- When the locator facility is used, data acquisition failed due to a communication error.

(d) position(Blob pattern, long start)

Function

Returns the byte location where pattern begins within the BLOB or BINARY value specified by the PrdbBlob object. This method starts searching for pattern at the location of start.

Format

```
public long position(Blob pattern, long start) throws
SQLException
```

Arguments

Blob pattern

Blob object used to specify the BLOB or BINARY value to be retrieved

long start

Location in the BLOB or BINARY value at which retrieval is to begin. The start position is 1.

Return value

Location at which the data specified in pattern begins

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This PrdbDataStream class is closed.
- The locator facility is used, but the PrdbConnection, PrdbStatement, or PrdbResultSet object related to the PrdbDataStream object is closed.

- PrdbDataStream has become invalid due to transaction settlement.
- When the locator facility is used, data acquisition failed due to a communication error.

(e) position(byte[] pattern, long start)

Function

Returns the byte location where pattern begins within the BLOB value indicated by this Blob object. This method starts searching for pattern at the location of start.

Format

```
public long position(byte[] pattern, long start) throws
SQLException
```

Arguments

byte[] pattern

byte[] to be retrieved

long start

Location in the BLOB value at which retrieval is to begin. The start position is 1.

Return value

Location at which the data specified in pattern begins

Exceptions

The JDBC driver throws an SQLException in the following cases:

- This PrdbDataStream class is closed.
- The locator facility is used, but the PrdbConnection, PrdbStatement, or PrdbResultSet object related to the PrdbDataStream object is closed.
- PrdbDataStream has become invalid due to transaction settlement.
- When the locator facility is used, data acquisition failed due to a communication error.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbBlob

18.4.10 Array interface

(1) Overview

The Array interface provides the following principal functions for accessing repetition columns:

- Acquisition of SQL Array values
- Acquisition of a result set containing SQL Array values

The JDBC driver uses the PrdbArray class to install the Array interface.

The JDBC driver generates PrdbArray class objects as return values of the getArray method of ResultSet.

(2) Methods

The table below lists the methods of the Array interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 18-54: Array interface methods

Subsection	Method	Function
(a)	getArray()	Acquires all elements of a repetition column as an Object array.
(b)	getArray(long index,int count)	Retrieves some elements of a repetition column.
(c)	getBaseType()	Acquires the JDBC type of the repetition column represented by the PrdbArray object as a java.sql.Types class literal.
(d)	getBaseTypeName()	Acquires the data type name of the repetition column represented by the PrdbArray object.
(e)	getResultSet()	Returns the ResultSet object containing the elements of a repetition column.
(f)	getResultSet(long index,int count)	Returns the ResultSet object containing the elements of a repetition column.

(a) getArray()

Function

Acquires all elements of a repetition column as an `Object` array.

Format

```
public Object getArray() throws SQLException
```

Arguments

None.

Return value

All elements of a repetition column

Functional detail

This method acquires all elements of a repetition column as an `Object` array.

The method returns the elements of the repetition column contained in the `PrdbArray` object in the `Object` array format. The following table shows the array that is returned depending on the data type:

HiRDB data type	Array that is returned
SMALLINT	java.lang.Short[]
INTEGR	java.lang.Integer[]
SMALLFLT,REAL	java.lang.Float[]
FLOAT,DOUBLE PRECISION	java.sql.Double[]
DECIMAL	java.math.BigDecimal[]
CHAR,NCHAR,MCHAR	java.lagn.String[]
VARCHAR,NVARCHAR,MVARCHAR	java.lagn.String[]
DATE	java.sql.Date[]
TIME	java.sql.Time[]
TIMESTAMP	java.sql.Timestamp[]
BINARY,BLOB	java.io.InputStream[][#]

#

In actuality, no array is returned because HiRDB does not support `BLOB` or `BINARY` repetition columns and cannot create the corresponding `PrdbArray` object.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PrdbResultSet` object that created this `PrdbArray` object has been closed.

  This includes the case where this driver has closed the `PrdbResultSet` object because the `PrdbStatement` object that created the `PrdbResultSet` object was closed.
- The `Connection` object used to create the `PrdbStatement` object, which created the `PrdbResultSet` object that created this `PrdbArray` object, is closed.
- `CltResultSet#get`*XXX*`()` resulted in an error.

(b) getArray(long index, int count)

Function

Retrieves some elements of a repetition column. This method acquires from a specified index as many elements as is specified as the maximum, and returns them as an `Object` array.

Format

```
public Object getArray(long index, int count) throws
SQLException
```

Arguments

long index

Index of the first element to be retrieved (the first element is 1)

int count

Number of consecutive array elements to be acquired

Return value

An `Object` array containing as many elements as specified in `count` from the specified index

Functional detail

This method retrieves some elements of a repetition column. The method acquires from a specified index as many elements as is specified as the maximum, and returns them as an `Object` array.

The following table shows the array that is returned depending on the relationship between the `index` and `count` argument values:

index	count	(index+count)	Array that is returned
0 < index ≤ number of elements	0 ≤ count	(index + count) - 1 ≤ number of elements	Array whose length equals the count value
		(index + count) - 1 > number of elements	Array whose length equals the number of elements - (index - 1)
	count < 0	--	SQLException is thrown
index > number of elements	--	--	SQLException is thrown
index < 1	--	--	SQLException is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an SQLException in the following cases:

- The PrdbResultSet object that created this PrdbArray object has been closed.

  This includes the case where this driver has closed the PrdbResultSet object because the PrdbStatement object that created the PrdbResultSet object was closed.
- The Connection object used to create the PrdbStatement object, which created the PrdbResultSet object that created this PrdbArray object, is closed.
- CltResultSet#get*XXX*() resulted in an error.
- The specified index argument value is less than 1 or the count argument value is less than 0.
- The specified index argument value is greater than the number of elements.

(c) getBaseType()

Function

Acquires the JDBC type of the repetition column represented by the PrdbArray object as a java.sql.Types class literal.

Format

```
public int getBaseType() throws SQLException
```

Arguments

None.

Return value

`java.sql.Types` class literal indicating the JDBC type of the repetition column represented by the `PrdbArray` object

Exceptions

None.

(d) getBaseTypeName()

Function

Acquires the data type name of the repetition column represented by the `PrdbArray` object.

Format

```
public String getBaseTypeName() throws SQLException
```

Arguments

None.

Return value

HiRDB data type name

Exceptions

None.

(e) getResultSet()

Function

Returns the `ResultSet` object containing the elements of a repetition column.

Format

```
public ResultSet getResultSet()
```

Arguments

None.

Return value

`ResultSet` object

Exceptions

The JDBC driver throws an SQLException in the following cases:

- The PrdbResultSet object that created this PrdbArray object has been closed.

  This includes the case where this driver has closed the PrdbResultSet object because the PrdbStatement object that created the PrdbResultSet object was closed.
- The Connection object used to create the PrdbStatement object, which created the PrdbResultSet object that created this PrdbArray object, is closed.
- An error occurred in the JDBC driver.

(f) getResultSet(long index,int count)

Function

Returns the ResultSet object containing the elements of a repetition column.

Format

```
public ResultSet getResultSet(long index,int count) throws
SQLException
```

Arguments

long index

Index of the first element to be retrieved (the first element is 1)

int count

Number of consecutive array elements to be acquired

Return value

ResultSet object

Functional detail

This method returns the ResultSet object containing the elements of a repetition column.

The ResultSet object contains at most as many consecutive elements of the repetition column as specified in count, beginning with the element specified in index.

The following table shows the ResultSet object that is returned depending on the relationship between the index and count argument values:

index	count	(index+count)	ResultSet that is returned
0 < `index` ≤ number of elements	0 ≤ `count`	(`index` + `count`) - 1 ≤ number of elements	Result set containing as many rows as specified in `count`
		(`index` -1 + `count`) - 1 > number of elements	Result set containing as many rows as the number of elements - (`index` - 1)
	`count` < 0	--	`SQLException` is thrown
`index` > number of elements	--	--	`SQLException` is thrown
`index` < 1	--	--	`SQLException` is thrown

Legend:

--: Not applicable

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PrdbResultSet` object that created this `PrdbArray` object has been closed.

  This includes the case where this driver has closed the `PrdbResultSet` object because the `PrdbStatement` object that created the `PrdbResultSet` object was closed.

- The `Connection` object used to create the `PrdbStatement` object, which created the `PrdbResultSet` object that created this `PrdbArray` object, is closed.
- The specified `index` argument value is less than `1` or the `count` argument value is less than `0`.
- The specified `index` argument value is greater than the number of elements.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbArray`

18.4.11 SQLException interface

The `SQLException` interface uses the `SQLException` class of the `java.sql` package directly. For details and usage information about each method provided by the SQLException interface, see the JDBC documentation provided by JavaSoft.

18.4.12 SQLWarning interface

(1) Overview

The SQLWarning interface provides the following principal function:

- Provision of information related to database access warnings

If a method object triggers a warning report, an SQLWarning object is accumulated without an exception notice to that method object.

(2) Notes

(a) Releasing accumulated SQLWarning objects

SQLWarning objects are accumulated as a chain linked to the method object (Connection, Statement, PreparedStatement, CallableStatement, or ResultSet) that triggers the warning reports.

To release accumulated SQLWarning objects explicitly, execute the clearWarnings method for the method object that triggered the warnings.

(b) Issuing conditions for SQLWarning objects

If the specified warning retention level indicates that warnings that occur during SQL execution are to be retained in the JDBC driver, the JDBC driver generates SQLWarning objects and retains warning information. In addition, a property can be used to specify warning retention for Connection objects.

The following table describes the conditions under which SQLWarning objects are generated.

Table 18-55: Conditions for generation of SQLWarning objects

SQL execution result		**Warning retention specification for Connection object**					
		Warning is retained			**Warning is not retained**		
		Warning retention level			**Warning retention level**		
		IGNORE	**SQLWARN**	**ALLWARN**	**IGNORE**	**SQLWARN**	**ALLWARN**
SQLCODE is a value greater than 0 other than 100, 110, or 120	Generated by an object other than a Connection object	No	No	Yes	No	No	Yes
	Generated by a Connection object	No	No	Yes	No	No	No

SQL execution result		Warning retention specification for Connection object					
		Warning is retained			Warning is not retained		
		Warning retention level			Warning retention level		
		IGNORE	SQLWARN	ALLWARN	IGNORE	SQLWARN	ALLWARN
`SQLWARN0` of the SQL Communications Area is `W` (except when `SQLWARN6` is `W`)	Generated by an object other than a `Connection` object	No	Yes	Yes	No	Yes	Yes
	Generated by a `Connection` object	No	Yes	Yes	No	No	No
Warning occurs in the JDBC driver	Generated by an object other than a `Connection` object	No	Yes	Yes	No	Yes	Yes
	Generated by a `Connection` object	No	Yes	Yes	No	No	No

Legend:

Yes: An `SQLWarning` object is generated.

No: An `SQLWarning` object is not generated.

Note

You can use `SQLWARNING_IGNORE` for URL, the `SQLWARNING_IGNORE` user property, or the `setSQLWarningIgnore` method to specify the warning retention specification for `Connection` objects. The default is `false`.

You can use `SQLWARNING_LEVEL` for URL, the `HiRDB_for_Java_SQLWARNING_LEVEL` property, or the `setSQLWarningLevel` method to specify the warning retention level. The default is `SQLWARN`.

18.4.13 Unsupported interfaces

The JDBC1.2 core API does not support the following interfaces:

- `Clob`
- `ParameterMetaData`
- `Savepoint`

- `SQLData`
- `SQLInput`
- `SQLOutput`

18.5 JDBC2.1 Core API

18.5.1 Expansion of the result set

Scrolling and parallel processing have been added to the JDBC2.1 Core API as expansion facilities for result sets (`ResultSet` class).

(1) Scrolling types

There are three types of scrolling for result sets:

- Forward-only scrolling
- Scroll-insensitive scrolling
- Scroll-sensitive scrolling

The JDBC2.1 Core API supports only forward-only scrolling and scroll-insensitive scrolling.

(2) Parallel processing types

There are two types of parallel processing for result sets:

- Read-only parallel processing
- Updatable parallel processing

The JDBC2.1 Core API supports only read-only parallel processing.

(3) Notes

(a) Notes about specifying an unsupported result set or type of parallel processing

No error results when an unsupported result set or an unsupported type of parallel processing is specified. The JDBC2.1 Core API assumes the result set that is closest to the specified type of result set or type of parallel processing, and generates an instance of the `Statement` class or that subclass. At this time, the API generates a warning (`SQLWarning` object) and associates it with an instance of the `Connection` class.

(b) Notes on using a scrolling-type result set

For a scrolling-type result set, all retrieved data is cached in the JDBC driver. This means that a large data size increases the possibility of a memory shortage or a drop in performance. When you use a scrolling-type result set, you should take steps in advance to minimize the amount of retrieved data. For example, you can add appropriate conditions to the SQL statements.

18.5.2 Batch update

In the JDBC 2.1 Core API, a batch update facility has been added to the `Statement` and PreparedStatement classes. This facility enables you to register multiple SQL statements or parameter values and execute them all at once.

When you execute a batch update, you can use facilities that use HiRDB arrays.

Facilities that use arrays are effective when you need to update quickly a large volume of data for HiRDB. For details about facilities that use arrays, see *4.8 Facilities using arrays*.

(1) Batch update with the Statement class

The following notes apply to batch update with the Statement class.

- Use the addBatch method to register multiple update SQL statements.
- Use the executeBatch method to execute registered update SQL statements collectively.
- An array of the numbers of rows updated by the individual update SQL statements is returned as the batch execution results.
- If an error occurs during batch update, the batch update facility throws a BatchUpdateException.
- If the registered SQL statements include a retrieval SQL statement, the batch update facility throws a BatchUpdateException when the executeBatch method is called.

Because the JDBC driver cannot execute multiple SQL statements simultaneously, it executes the registered SQL statements consecutively.

(2) Batch update with the PreparedStatement class

The following notes apply to batch update with the PreparedStatement class.

- Use the normal procedure (set*XXX* method) to specify the ? parameters for the update SQL statements specified during PreparedStatement instance generation.
- Use the addBatch method to register the ? parameter sets.
- Use the executeBatch method to execute the registered ? parameter sets collectively.
- An array of the number of rows updated by the individual ? parameter sets is returned as the batch execution results.
- If an error occurs during batch execution, the batch update facility throws a BatchUpdateException.
- If an SQL statement specified during PreparedStatement instance generation is a retrieval SQL statement, the batch update facility throws a BatchUpdateException when the executeBatch method is called.

Note that the JDBC driver executes SQL statements consecutively in the following cases:

- During batch updating when `addBatch` specifications for parameters and SQL statements are combined
- During batch updating with SQL statements that contain `?` parameters for HiRDB's `BINARY` type (the length of the data to be set for a `BINARY`-type `?` parameter exceeds 32,000 bytes)
- During batch updating for SQL statements that contain `?` parameters for HiRDB's `BLOB` type

Notes

You must pay close attention to subsequent executions of `addBatch`, because the values that were set for the previous execution are inherited when the number of parameters specified by the `set`*XXX* method is insufficient.

The following example has two `INTEGER`-type arrays (array 1 and array 2):

Specification example

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
prepstmt.executeBatch();
```

Explanation

- The values that are set by the first `addBatch` are array 1=`100` and array 2=`100`.

  If the number of parameters specified by `addBatch` is insufficient, an error occurs.
- The values that are set by the second `addBatch` are array 1=`200` and array 2=`100`.

  The second `addBatch` does not update the information for array 2, so the array 2 information is inherited from the first `addBatch`.

(3) Batch update with the CallableStatement class

The following notes apply to batch update with the `CallableStatement` class.

- Use the normal procedure (`set`*XXX* method) to specify the input parameters for the Java stored routines specified during `CallableStatement` class instance generation.

- Use the addBatch method to register the input parameter sets.
- Use the executeBatch method to execute the registered input parameter sets collectively.
- An array of the return values (number of updated rows) of the Java stored routines executed by the input parameter sets is returned as the batch execution results.
- If an error occurs during batch execution, the batch update facility throws a BatchUpdateException.
- If a Java stored routine specified during CallableStatement instance generation is not a routine that returns the number of updated rows, the batch update facility throws a BatchUpdateException when the executeBatch method is called.
- If a Java stored routine specified during CallableStatement instance generation has output and input-output parameters, the batch update facility throws a BatchUpdateException when the executeBatch method is called.

Because the JDBC driver cannot execute multiple rows of ? parameters for a stored procedure in the batch mode, it executes them consecutively.

Note

Whether a stored procedure that returns a result set (ResultSet) will actually return a result set will not be known until the stored procedure has executed during batch updating. Therefore, if data is updated within the stored procedure, updated information might be applied. For example, if you perform batch updating on a stored procedure that searches and acquires the results of update processing, a BatchUpdateException occurs, but updated information might still be applied.

(4) Notes

(a) Implicit commit by the HiRDB server

If the SQL statements registered with addBatch contain one of the following SQL statements, you must use the batch update facility for SQL statements carefully, because the HiRDB server commits that SQL statement implicitly when the statement is executed:

- PURGE TABLE statement
- Any definition SQL statement in which YES is specified for the PDCMMTBFDDL client environment variable

(b) Processing by the batch update facility when addBatch specifications for parameters and SQL statements are combined

When addBatch specifications for parameters and addBatch specifications for SQL statements are combined, the batch update facility executes the addBatch specifications sequentially instead of by batch update. An example is shown below:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.addBatch();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.addBatch();
pstmt.addBatch("INSERT INTO T2 VALUES(1,2,3)");
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.executeBatch();
```

When this UAP is executed, each `addBatch` unit becomes an SQL execution, because there are both `addBatch` specifications for parameters and `addBatch` specifications for SQL statements. Therefore, executing this UAP produces the same results as executing the following UAP:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.executeUpdate();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.executeUpdate("INSERT INTO T2 VALUES(1,2,3)");
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
```

When you use the batch update facility on a combination of `addBatch` for parameters and `addBatch` for SQL statements, it is recommended that you disable the auto-commit mode for the `Connection` class.

(c) Batch update with SQL statements that contain a ? parameter for HiRDB's BINARY type

When batch update is executed with SQL statements that contain a `?` parameter for HiRDB's `BINARY` type, sequential execution is executed instead of batch update when the following condition applies:

- The length of the data to be set with the `set`*XXX* method for the `?` parameter exceeds 32,000 bytes (if character data is specified with the `setString` method, the data length after the data has been encoded into data to be passed to HiRDB exceeds 32,000 bytes).

(d) Batch update for SQL statements that contain a ? parameter for HiRDB's BLOB type

When batch update is executed with SQL statements that include a `?` parameter for HiRDB's `BLOB` type, the statements are executed sequentially instead of by batch update.

(e) Registering multiple parameters with the addBatch method

The JDBC driver accumulates in the driver all parameters registered with the `addBatch` method until the `executeBatch` method is executed. You should make note of the amount of memory being used when you are registering multiple parameters.

When batch update is executed with a facility that uses HiRDB arrays, the maximum number of executions that the JDBC driver can request to the HiRDB server is 30,000. To register more than 30,000 parameters, you must divide them into groups of no more than 30,000 and request SQL execution to the HiRDB server for each group. Note also that because of the amount of memory in the JDBC driver that is used in this case, the performance enhancement expected for batch updating may not be realized. When more than 30,000 SQL executions are necessary, it is recommended that you execute the `executeBatch` method in units of 30,000 or fewer SQL executions.

(f) Update count reported by BatchUpdateException

The following table shows the update count (`int`-type array) that is reported in the return value of the `getUpdateCounts` method of `BatchUpdateException` that occurs during batch update processing.

Table 18-56: Update count reported in the return value of the getUpdateCounts method

Batch update execution mode	BATCHEXCEPTION_BEHAVIOR#1 setting	
	TRUE	FALSE
Batch execution using the array facility	Array containing no elements	Array containing *n* elements *n*: The `addBatch` that registered the parameter resulting in the error. When an error occurs, all elements are set to `Statement.EXECUTE_FAILED` because processing is rolled back.

Batch update execution mode	BATCHEXCEPTION_BEHAVIOR[#1] setting	
	TRUE	FALSE
Consecutive execution using a JDBC driver	Array containing as many elements as there are SQL statements executed The number of updated rows[#2] is set in each array element.	Array containing *n* elements *n*: The `addBatch` that registered the parameter or SQL statement resulting in the error. Array elements are set to the following values: Elements 0 to *n* - 2: Number of updated rows[#2] Element *n* - 1: `Statement.EXECUTE_FAILED`

#1

Specified by one of the following:

- User property used when HiRDB is connected: HiRDB_for_Java_BATCHEXCEPTION_BEHAVIOR
- `BATCHEXCEPTION_BEHAVIOR` for URL
- `setBatchExceptionBehavior` method of a `DataSource` interface

When the connected HiRDB's version is 08-01 or earlier, it is assumed that `TRUE` is specified.

#2

`Statement.SUCCESS_NO_INFO` is set if a `CALL` statement is executed using the `executeBatch` method of the `CallableStatement` class.

The following are examples of update count:

Example program of batch execution using the array facility

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES(?,?)");
pstmt.setInt(1, 1);
pstmt.setString(2,"aaaa");
pstmt.addBatch();
pstmt.setInt(1, 2);
pstmt.setString(2,"bbbbbbbb");
pstmt.addBatch();.....................................[A]
pstmt.setInt(1, 3);
pstmt.setString(2,"cccc");
pstmt.addBatch();
pstmt.executeBatch();
```

Example program of consecutive execution using a JDBC driver

```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO T1 VALUES(1,'aaaa')");
stmt.addBatch("INSERT INTO T1 VALUES(2,'bbbbbbbb')");...[A]
stmt.addBatch("INSERT INTO T1 VALUES(3,'cccc')");
stmt.executeBatch();
```

The following table shows the update count that is returned by the `getUpdateCounts` method when these example programs are executed and the parameter or SQL statement registered in [A] results in an error:

Batch update execution mode	BATCHEXCEPTION_BEHAVIOR setting	
	TRUE	FALSE
Batch execution using the array facility	`int`-type array containing no elements	`int`-type array containing 2 elements Value of element 0: Statement.EXECUTE_FAILED Value of element 1: Statement.EXECUTE_FAILED
Consecutive execution using a JDBC driver	`int`-type array containing 1 element Value of element 0: Number of updated rows	`int`-type array containing 2 elements Value of element 0: Number of updated rows Value of element 1: Statement.EXECUTE_FAILED

18.5.3 Added data types

Several new JDBC SQL types have been added to the JDBC2.1 Core API. Although the following JDBC SQL types have been added, the JDBC driver cannot use them:

- `BLOB`
- `CLOB`
- `ARRAY`
- `REF`
- `DISTINCT`
- `STRUCT`
- `JAVA OBJECT`

18.5.4 Unsupported interfaces

The JDBC2.1 Core API does not support the following interfaces:

- `Array`
- `Clob`
- `Ref`

- `SQLData`
- `SQLInput`
- `SQLOutput`
- `Struct`

18.6 JDBC2.0 Optional Package

The following functions were added to the JDBC2.0 Optional Package:

- JNDI support
- Connection pool
- Distributed transactions
- RowSets

Note, however, that the JDBC driver cannot use RowSets.

The following table lists the functions and interfaces of the JDBC2.0 Optional Package.

Table 18-57: Functions and interfaces of the JDBC2.0 Optional Package

Function	Interface
JNDI support	`DataSource`
Connection pool	`ConnectionPoolDataSource` `PooledConnection`

Distributed transaction	`XAConnection`
	`XADataSource`
	`XAResource`
	`XAException`

18.6.1 DataSource interface

For details and usage information about the methods provided by the `DataSource` interface, see the JDBC documentation. This section shows the `DataSource` interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the `DataSource` interface.

Table 18-58: DataSource interface methods

Subsectio n	Method	Function
(a)	getConnection()	Attempts to connect to the database using the connection information set in the data source.

Subsectio n	Method	Function
(b)	getConnection(String username,String password)	Attempts to connect to the database using the connection information set in the data source.
(c)	getLoginTimeout()	Returns the value specified by the `setLoginTimeout` method.
(d)	getLogWriter()	Acquires the `DataSource` object's log writer.
(e)	setLoginTimeout(int seconds)	Specifies the maximum amount of time (in seconds) to wait for the connection to the database to be completed.
(f)	setLogWriter(PrintWriter out)	Sets a log writer for the `DataSource` object.

(a) getConnection()

Function

Attempts to connect to the database using the connection information set in the data source.

Format

```
public synchronized Connection getConnection() throws
SQLException
```

Arguments

None.

Return value

`Connection` object

Functional detail

This method uses the connection information that has been set in the `DataSource` object to connect to the HiRDB server, and returns a connected `Connection` object. For details about the specification method priorities for the user name and password, see *18.11 Connection information priorities*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurred.

- Specified connection information is invalid.

  For details about the conditions under which the connection information items become invalid, see *18.2.2(1) URL syntax* and *18.2.2(2) User properties*.

(b) getConnection(String username, String password)

Function

Attempts to connect to the database using the connection information set in the data source.

Format

```
public synchronized Connection getConnection(String
username, String password) throws SQLException
```

Arguments

String username

User name used to establish connection

String password

Password used to establish connection

Return value

`Connection` object

Functional detail

This method uses the information specified in the arguments and the connection information that has been set in the `DataSource` object to connect to the HiRDB server, and returns a connected `Connection` object.

When the `null` value is set in the `username` or `password` argument it means that the argument with the `null` value is not being used to set the user name or password. When the `password` argument contains a character string with a length of 0 it means that no password is specified. When a user ID is specified in both the `username` argument and `ConnectionProperty`, the `username` argument takes precedence. Similarly, the `password` argument takes precedence for a password specification. For details about the specification priorities when the `username` and `password` arguments are not specified, see *18.11 Connection information priorities*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurred.
- The specified connection information is invalid.

  For details about the conditions under which the connection information items become invalid, see *18.2.2(1) URL syntax* and *18.2.2(2) User properties*.
- The user name specified in the `username` argument is a character string with a length of 0.

(c) getLoginTimeout()

Function

Returns the value specified by the `setLoginTimeout` method.

Format

```
public synchronized int getLoginTimeout()
```

Arguments

None.

Return value

`int` type:

Value specified by the `setLoginTimeout` method. If no value has been specified by `setLoginTimeout`, this method returns `0`.

Exceptions

None.

(d) getLogWriter()

Function

Acquires the `DataSource` object's log writer.

Format

```
public synchronized PrintWriter getLogWriter() throws
SQLException
```

Arguments

None.

Return value

This method returns the log writer for the PrdbDataSource object. If no log writer has been set, the method returns the NULL value.

Exceptions

None.

(e) setLoginTimeout(int seconds)

Function

Specifies the maximum amount of time (in seconds) to wait for the connection to the database to be completed.

Format

```
public synchronized void setLoginTimeout(int seconds) throws
SQLException
```

Arguments

int seconds

Maximum connection wait time (seconds)

Return value

None.

Functional detail

This method is used when a physical connection is established with the HiRDB server in order for the getConnection method to acquire a Connection object. If 0 is specified or setLoginTimeout is not executed, the value specified in PDCONNECTWAITTIME is assumed as the maximum amount of time to wait for the HiRDB server to be physically connected.

Exceptions

If the seconds argument value is less than 0 or is greater than 300, the method throws an SQLException.

(f) setLogWriter(PrintWriter out)

Function

Sets a log writer for the DataSource object.

Format

```
public synchronized void setLogWriter(PrintWriter out)
throws SQLException
```

Arguments

PrintWriter out

Log writer

Return value

None.

Exceptions

None.

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbDataSource

18.6.2 ConnectionPoolDataSource interface

For details and usage information about the methods provided by the ConnectionPoolDataSource interface, see the JDBC documentation. This section shows the ConnectionPoolDataSource interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the ConnectionPoolDataSource interface.

Table 18-59: ConnectionPoolDataSource interface methods

Subsection	Method	Function
(a)	getLoginTimeout()	Returns the value specified by the setLoginTimeout method.
(b)	getLogWriter()	Acquires the ConnectionPoolDataSource object's log writer.
(c)	getPooledConnection()	Acquires a PooledConnection object that can be used as a pooled connection, based on connection information set in the data source.
(d)	getPooledConnection(String user, String password)	Acquires a PooledConnection object that can be used as a pooled connection, based on connection information set in the data source.

Subsectio n	Method	Function
(e)	setLoginTimeout(int seconds)	Specifies the maximum amount of time (in seconds) to wait for the connection to the database to be completed.
(f)	setLogWriter(PrintWriter out)	Sets a log writer for the ConnectionPoolDataSource object.

(a) getLoginTimeout()

Function

Returns the value specified by the setLoginTimeout method.

Format

```
public synchronized int getLoginTimeout() throws
SQLException
```

Arguments

None.

Return value

int type:

Value specified by the setLoginTimeout method. If the setLoginTimeout method has not been specified, this method returns 0.

Exceptions

None.

(b) getLogWriter()

Function

Acquires the ConnectionPoolDataSource object's log writer.

Format

```
public synchronized PrintWriter getLogWriter() throws
SQLException
```

Arguments

None.

Return value

This method returns the log writer for the PrdbConnectionPoolDataSource object. If no log writer has been set, the method returns the NULL value.

Exceptions

None.

(c) getPooledConnection()

Function

Acquires a PooledConnection object that can be used as a pooled connection, based on connection information set in the data source.

Format

```
public synchronized PooledConnection getPooledConnection()
throws SQLException
```

Arguments

None.

Return value

PooledConnection object

Functional detail

This method returns a PooledConnection object that can be used as pooled connection, based on connection information that has been set in a DataSource object. For details about the specification method priorities for the user name and password, see *18.11 Connection information priorities*.

Exceptions

JDBC driver throws an SQLException if a database access error has occurred.

(d) getPooledConnection(String user, String password)

Function

Acquires a PooledConnection object that can be used as a pooled connection, based on connection information set in the data source.

Format

```
public synchronized PooledConnection
getPooledConnection(String user, String password) throws
SQLException
```

Arguments

String user: User name used to establish connection

String password: Password used to establish connection

Return value

`PooledConnection` object

Functional detail

This method returns a `PooledConnection` object that can be used as a pooled connection, based on the information specified in the arguments and the connection information that has been set in a `DataSource` object.

When the `null` value is set in the `user` or `password` argument it means that the argument with the `null` value is not being used to set the user name or password. When the `password` argument contains a character string with a length of 0, it means that no password has been specified. When a user ID is specified in both the `user` argument and `ConnectionProperty`, the `user` argument takes precedence. Similarly, the `password` argument takes precedence for a password specification. For details about the specification priorities when the `user` and `password` arguments are not specified, see *18.11 Connection information priorities*.

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurred.
- The user name specified in the argument is a character string with a length of 0.

(e) setLoginTimeout(int seconds)

Function

Specifies the maximum amount of time (in seconds) to wait for the connection to the database to be completed.

Format

```
public synchronized void setLoginTimeout(int seconds) throws
SQLException
```

Arguments

int seconds: Maximum connection wait time (seconds)

Return value

None.

Functional detail

This method is used when physical connection is established with the HiRDB server in order for the `getConnection` method to acquire a `Connection` object. If `0` is specified or `setLoginTimeout` is not executed, the value specified in `PDCONNECTWAITTIME` is assumed as the maximum amount of time to wait for the HiRDB server to be physically connected.

Exceptions

If the `seconds` argument value is less than `0` or is greater than `300`, the method throws an `SQLException`.

(f) setLogWriter(PrintWriter out)

Function

Sets a log writer for the `ConnectionPoolDataSource` object.

Format

```
public synchronized void setLogWriter(PrintWriter out)
```

Arguments

`PrintWriter out`: Log writer

Return value

None.

Exceptions

None.

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbConnectionPoolDataSource`

18.6.3 PooledConnection interface

For details and usage information about the methods provided by the `PooledConnection` interface, see the JDBC documentation. This section shows the `PooledConnection` interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the `PooledConnection` interface.

Table 18-60: PooledConnection interface methods

Subsection	Method	Function
(a)	getConnection()	Establishes a physical connection when one is needed and returns a `Connection` object. There is a one-to-one correspondence between a `Connection` object and a physical connection with the HiRDB server.
(b)	addConnectionEventListener(ConnectionEventListener listener)	Registers a specified event listener so that events that occur in this `PooledConnection` object will be reported.
(c)	close()	Closes the physical connection.
(d)	removeConnectionEventListener(ConnectionEventListener listener)	Deletes from the component list a specified event listener that reports on events that occur in this `PooledConnection` object.

(a) getConnection()

Function

Establishes a physical connection when one is needed and returns a `Connection` object. There is a one-to-one correspondence between a `Connection` object and a physical connection with the HiRDB server.

Format

```
public synchronized Connection getConnection() throws
SQLException
```

Arguments

None.

Return value

`Connection` object

Functional detail

This method establishes a physical connection when one is needed and returns a `Connection` object. There is a one-to-one correspondence between a `Connection` object and a physical connection with the HiRDB server. Once established, a physical connection is not closed until this class object is closed. This class object maintains the physical connection even if the `close` method is executed on the `Connection` object. This physical connection can then be used again the next time a connection request is issued by calling this method (the wait time specified in `setLoginTimeout` or the `PDCONNECTWAITTIME` client environment definition does not apply).

Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurred.
- Specified connection information is invalid.

(b) addConnectionEventListener(ConnectionEventListener listener)

Function

Registers a specified event listener so that events that occur in this `PooledConnection` object will be reported.

Format

```
public synchronized void
addConnectionEventListener(ConnectionEventListener
listener)
```

Arguments

ConnectionEventListener listener

Component that implements the `ConnectionEventListener` interface, so that if the connection is closed or an error occurs, that event will be reported. Normally, this is a connection pool management program.

Return value

None.

Functional detail

This method registers a specified event listener so that events that occur in this `PooledConnection` object will be reported. If the listener to be added is `null`, this method does not register a listener.

This driver's method cannot be called from the event listener registered by the `addConnectionEventListener` method. If an attempt is made to call the driver's method, the driver might stop responding (due to a deadlock).

Exceptions

None.

(c) close()

Function

Closes the physical connection.

Format

```
public synchronized void close()
```

Arguments

None.

Return value

None.

Functional detail

This method closes the physical connection for all pooled connections. The method attempts to close the physical connection by executing `PooledConnection.close()` even if a `Connection` object has been acquired and the database is being accessed.

Exceptions

None.

(d) removeConnectionEventListener(ConnectionEventListener listener)

Function

Deletes from the component list a specified event listener that reports on events that occur in this `PooledConnection` object.

Format

```
public synchronized void
removeConnectionEventListener(ConnectionEventListener
listener)
```

Arguments

ConnectionEventListenerlistener

Component registered as a listener by implementing the `ConnectionEventListener` interface. Normally, this is a connection pool management program.

Return value

None.

Exceptions

None.

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbPooledConnection`

18.6.4 XAConnection interface

For details and usage information about the methods provided by the `XAConnection` interface, see the JDBC documentation. This section shows the `XAConnection` interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the `XAConnection` interface.

Table 18-61: XAConnection interface methods

Method	Remarks
`getXAResource()`	--

Legend:

--: None.

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbXAConnection`

18.6.5 XADataSource interface

For details and usage information about the methods provided by the `XADataSource` interface, see the JDBC documentation. This section shows the `XADataSource` interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the `XADataSource` interface.

Table 18-62: XADataSource interface methods

Method	Remarks
`getLoginTimeout()`	Returns the value specified by the `setLoginTimeout` method. If no value was set by the `setLoginTimeout` method, this method returns `0`.

Method	Remarks
getLogWriter()	--
getXAConnection()	For details about the priorities among the setting methods for authorization identifiers and passwords, see *18.11 Connection information priorities*.
getXAConnection(String username,String password)	If the user or password argument is the null value, this method indicates that no authorization identifier or password was specified by this argument. If the password argument is a character string whose length is 0, this method indicates that no password was specified. For details about the setting value used when a password is not specified, see *18.11 Connection information priorities*. If the user argument is a character string whose length is 0, this method throws an SQLException.
setLoginTimeout(int seconds)	This specification is used only for the physical connection time with the HiRDB server. When 0 is specified or when the setLoginTimeout method is not executed, the time that was specified in PDCONNECTWAITTIME in the client environment definition becomes the maximum wait time for the HiRDB server. If a value outside the range 0-300 is specified, this method throws an SQLException.
setLogWriter(PrintWriter out)	--

Legend:

--: None

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbXADataSource

18.6.6 XAResource interface

For details and usage information about the methods provided by the XAResource interface, see the JDBC documentation. This section shows the XAResource interface methods that are supported by the JDBC driver.

(1) Methods

The following table lists the methods of the XAResource interface.

Table 18-63: XAResource interface methods

Method	Remarks
`commit(Xid xid, boolean onePhase)`	--
`end(Xid xid, int flags)`	--
`getTransactionTimeout()`	This method returns `0` unconditionally.
`prepare(Xid xid)`	--
`recover(int flag)`	--
`rollback(Xid xid)`	--
`setTransactionTimeout(int seconds)`	This method does not set the transaction timeout value. Instead, it returns `false` to indicate that the transaction timeout time was not set properly.
`start(Xid xid, int flags)`	--

Legend:

--: None

(2) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbXAResource`

18.6.7 XAException interface

The `XAException` interface directly uses the `XAException` class of the `javax.transaction.xa` package. For details and usage instructions about the methods provided by the `XAException` interface, see the related JDBC documentation.

18.6.8 Unsupported interfaces

The JDBC2.0 Optional Package does not support the following interfaces:

- `RowSet`
- `RowSetInternal`
- `RowSetListner`
- `RowSetMetaData`
- `RowSetReader`

18.7 Connection information setup and acquisition interface

The DataSource, ConnectionPoolDataSource, and XADataSource classes provided by the JDBC2.0 Optional Package provide methods for setting and getting connection information necessary for connection to the database, in addition to the methods prescribed by the JDBC2.0 Optional Package specifications.

The following table lists the methods for setting and getting connection information.

Table 18-64: Methods for setting and getting connection information

Method	Function
setDescription	Sets the additional connection information needed for connection to the database.
getDescription	Gets the additional connection information needed for connection to the database.
setDBHostName	Sets the host name of the HiRDB to be connected.
getDBHostName	Gets the host name of the HiRDB to be connected.
setJDBC_IF_TRC	Sets whether or not a JDBC interface method trace is to be acquired.
getJDBC_IF_TRC	Gets the setting information about whether or not a JDBC interface method trace is to be acquired.
setTRC_NO	Sets the number of entries in the JDBC interface method trace.
getTRC_NO	Gets the number of entries in the JDBC interface method trace.
setUapName	Sets a UAP name.
getUapName	Gets the UAP name.
setUser	Sets an authorization identifier for database connection.
getUser	Gets the authorization identifier for database connection.
setPassword	Sets a password for database connection.
getPassword	Gets the password for database connection.
setXAOpenString	Sets an XA open character string.
getXAOpenString	Gets the XA open character string.
setXACloseString	Sets an XA close character string.
getXACloseString	Gets the XA close character string.

Method	Function
setLONGVARBINARY_Access	Sets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).
getLONGVARBINARY_Access	Gets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).
setSQLInNum	Sets the maximum number of input ? parameters in the SQL statements to be executed.
getSQLInNum	Gets the maximum number of input ? parameters in the SQL statements to be executed.
setSQLOutNum	Sets the maximum number of retrieval items for the SQL statements to be executed.
getSQLOutNum	Gets the maximum number of retrieval items for the SQL statements to be executed.
setSQLWarningLevel	Sets the warning retention level for warnings that occur during SQL execution.
getSQLWarningLevel	Gets the warning retention level for warnings that occur during SQL execution.
setXALocalCommitMode	Sets whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.
getXALocalCommitMode	Gets the setting information about whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.
setSQLWarningIgnore	Sets whether or not warnings returned from the database are to be discarded by the Connection class.
getSQLWarningIgnore	Gets the setting information about whether or not warnings returned from the database are to be discarded by the Connection class.
setHiRDBCursorMode	Sets whether or not objects of the ResultSet class are to be validated when HiRDB executes commit processing.
getHiRDBCursorMode	Gets the setting information about whether or not objects of the ResultSet class are to be validated when HiRDB executes commit processing.
setNotErrorOccurred	Sets whether or not the calling of ConnectionEventListener.connectionErrorOccurred is to be suppressed.
getNotErrorOccurred	Gets the setting information about whether or not the calling of ConnectionEventListener.connectionErrorOccurred has been suppressed.
setEnvironmentVariables	Sets client environment definitions for HiRDB.
getEnvironmentVariables	Gets the client environment definitions for HiRDB that were set.

Method	Function
setEncodeLang	Sets the name of the conversion character set for data conversion.
getEncodeLang	Gets the name of the conversion character set for data conversion that was set.
setMaxBinarySize	Sets the maximum data size for retrieval of data of the LONGVARBINARY type (a JDBC SQL type).
getMaxBinarySize	Gets the maximum data size for retrieval of data of the LONGVARBINARY type (a JDBC SQL type).
setStatementCommitBehavior	Sets whether or not statement objects are to remain valid after a transaction is committed.
getStatementCommitBehavior	Gets the setting information about whether or not statement objects are to remain valid after a transaction is committed.
setLONGVARBINARY_AccessSize	Sets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.
getLONGVARBINARY_AccessSize	Gets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.
setLONGVARBINARY_TruncError	Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the LONGVARBINARY type (a JDBC SQL type).
getLONGVARBINARY_TruncError	Gets the setting information about whether or not an exception is to be thrown if truncation occurs during acquisition of data of the LONGVARBINARY type (a JDBC SQL type).
setStatementCloseBehavior	Sets whether preprocessing results are to be ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes).
getStatementCloseBehavior	Gets the setting information about whether preprocessing results are ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes).
setHiRDBINI	Sets a directory for the HiRDB.INI file.
getHiRDBINI	Gets the directory for the HiRDB.INI file.
setBatchExceptionBehavior	Sets whether a JDBC standard-compliant update count is to be set as the return value of the getUpdateCounts method of java.sql.BatchUpdateException.
getBatchExceptionBehavior	Gets the setting information about whether a JDBC standard-compliant update count is set as the return value of the getUpdateCounts method of java.sql.BatchUpdateException.

18.7.1 setDescription

(a) Function

Sets the additional connection information needed for connection to the database.

(b) Format

```
public void setDescription ( String description ) throws
SQLException
```

(c) Arguments

`String description`

Specifies additional connection information. If the null value is specified, the current additional connection information that had been set by this method is invalidated and the settings are returned to their initial status.

(d) Return value

None.

(e) Functional detail

The following table shows the additional connection information that can be set with this method.

Setting	Setting details	Setting required?
HiRDB port number	Sets the HiRDB port number, as a character string. For details about the priorities among the setting methods for the HiRDB port number, see *18.11 Connection information priorities*.	Optional

Setting	Setting details	Setting required?
Environment variable group name of HiRDB client	Sets the environment variable group name of the HiRDB client. The name is expressed as an absolute path name that follows @HIRDBENVGRP=. Note the following points: • If no value is set following the equal sign, as in @HIRDBENVGRP=,, the JDBC driver assumes that there is no specification for this item. • The environment variable group name is case sensitive. Also, the environment variable group name depends on the OS. • If the environment variable group name contains a single-byte space or a single-byte at mark (@), you must enclose the name in single-byte double quotation marks ("). When an environment variable group name is enclosed in single-byte double quotation marks, any characters following the concluding single-byte double quotation mark through the end of the character string are ignored. An environment variable group name containing a single-byte double quotation mark or a single-byte comma cannot be specified. • For Windows, an environment variable group name that was specified with the HiRDB client environment variable registration tool cannot be specified.	Optional
HiRDB environment variable group identifier	Sets the HiRDB environment variable group identifier, as four alphanumeric characters.	Required during XA connection

Note 1:

Specification examples are shown below. In these examples ds represents the name of a variable that has reference to the PrdbDataSource class's instance. Δ represents a single-byte space character.

Example 1: When specifying the HiRDB port number

```
ds.setDescription ("22200");
```

Example 2: When the path of the environment variable group name is C:\HiRDB_P\Client\HiRDB.ini

```
ds.setDescription
("@HIRDBENVGRP=C:\\HiRDB_P\\Client\\HiRDB.ini");
```

Example 3: When the path of the environment variable group name is C:\ProgramΔFiles\HITACHI\HiRDB\HiRDB.ini

```
ds.setDescription
```

```
("@HIRDBENVGRP=\"C:\\ProgramΔFiles\\HITACHI\\HiRDB\HiR
DB.ini\"");
```

Example 4: When the path of the environment variable group name is `/HiRDB_P/Client/HiRDB.ini`

```
ds.setDescription ("@HIRDBENVGRP=/HiRDB_P/Client/
HiRDB.ini");
```

Example 5: When a HiRDB environment variable group identifier is specified during an XA connection

```
ds.setDescription ("HDB1");ds.setXAOpenString
("HDB1+C:\\ProgramΔFiles\\HITACHI\\HiRDB\\HiRDB.ini");
```

Note 2:

Do not include single-byte spaces in an environment variable group name. Examples of specification errors are shown below:

```
@ΔHIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRPΔ=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=Δ/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.iniΔ
```

Note: Δ represents a single-byte space character.

(f) Exceptions

When an environment variable group name begins with a single-byte at mark (@) and the information specified following the at mark includes a single-byte space, this method throws an `SQLException`.

18.7.2 getDescription

(a) Function

Gets the additional connection information needed for connection to the database.

(b) Format

```
public String getDescription() throws SQLException
```

(c) Arguments

None.

(d) Return value

```
String
```

Additional connection information (if this information has not been specified, the method returns the null value)

(e) Functional detail

Returns the additional connection information needed for connection to the database, as was set by the `setDescription` method.

(f) Exceptions

None.

18.7.3 setDBHostName

(a) Function

Sets the name of the HiRDB host to be connected.

(b) Format

```
public void setDBHostName ( String db_host_name ) throws
SQLException
```

(c) Arguments

```
String db_host_name
```

Sets a HiRDB host name.

If the null value is specified, the current host name that had been set with this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the host name of the HiRDB to be connected.

For details about the priorities among the settings methods for the HiRDB host name, see *18.11 Connection information priorities*.

(f) Exceptions

None.

18.7.4 getDBHostName

(a) Function

Gets the name of the HiRDB host to be connected.

(b) Format

```
public String getDBHostName() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

HiRDB host name (if this information has not been specified, the method returns the null value)

(e) Functional detail

Returns the host name of the HiRDB to be connected, as was set with the `setDBHostName` method.

(f) Exceptions

None.

18.7.5 setJDBC_IF_TRC

(a) Function

Sets whether or not a JDBC interface method trace is to be acquired.

(b) Format

```
public void setJDBC_IF_TRC ( boolean flag ) throws SQLException
```

(c) Arguments

`boolean flag`

Specifies whether or not a trace is to be acquired:

`true`: Acquire a trace.

`false`: Do not acquire a trace.

(d) Return value

None.

(e) Functional detail

Sets whether or not a JDBC interface method trace is to be acquired.

The default value when this method is not called is `false` (trace is not acquired). You can use the `setLogWriter` method in a separate operation to set the effective output destination. For details about the JDBC interface method trace, see *18.14 JDBC interface method trace*.

(f) Exceptions

None.

(g) Note

Whether or not a JDBC interface method trace is to be acquired cannot be set separately for each instance. The setting that is set by this method affects all `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` instances in existence, both when the setting is set and after the setting has been set.

18.7.6 getJDBC_IF_TRC

(a) Function

Gets setting information about whether or not a JDBC interface method trace is to be acquired.

(b) Format

```
public boolean getJDBC_IF_TRC() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not a trace is to be acquired:

`true`: A trace is acquired.

`false`: A trace is not acquired.

(e) Functional detail

Returns the setting information about whether or not a trace is to be acquired, as was set by the `setJDBC_IF_TRC` method.

For details about the JDBC interface method trace, see *18.14 JDBC interface method trace*.

(f) Exceptions

None.

18.7.7 setTRC_NO

(a) Function

Sets the number of entries in the JDBC interface method trace.

(b) Format

```
public void setTRC_NO ( int trc_no ) throws SQLException
```

(c) Arguments

`int trc_no`

Specifies the number of entries in the JDBC interface method trace.

(d) Return value

None.

(e) Functional detail

Sets the number of entries in the JDBC interface method trace, as a value in the range from 10 to 1,000.

When this method is not called, the default number of entries in the JDBC interface method trace is `500`.

For details about the JDBC interface method trace, see *18.14 JDBC interface method trace*.

(f) Exceptions

If a value outside the range from 10 to 1,000 is set, this method throws an `SQLException`.

18.7.8 getTRC_NO

(a) Function

Gets the number of entries in the JDBC interface method trace.

(b) Format

```
public int getTRC_NO() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

Number of JDBC interface method trace entries (if this information has not been specified, the method returns the default value (500))

(e) Functional detail

Returns the number of entries in the JDBC interface method trace, as was set by the `setTRC_NO` method.

For details about the JDBC interface method trace, see *18.14 JDBC interface method trace*.

(f) Exceptions

None.

18.7.9 setUapName

(a) Function

Sets a UAP name.

(b) Format

```
public void setUapName ( String uap_name ) throws SQLException
```

(c) Argument

`String uap_name`

Specifies a UAP name.

If the null value is specified, the current UAP name that had been set with this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets a UAP name.

The specified UAP name is used for the following purposes:

- In the output information to each type of trace information
- In the UAP identification information that is output when the `-d prc` option is specified in the `pdls` command

In the following cases, the JDBC driver assumes that no UAP name has been set by this method (for details about how the JDBC driver handles the situation when there is no setting, see *18.11 Connection information priorities*):

- When the null value is specified in the `uap_name` argument
- When a character string whose length is 0 or a character string consisting of only single-byte spaces is specified in the `uap_name` argument

(f) Exceptions

None.

(g) Notes

The UAP specified by this method is encoded using the conversion character set specified by the `setEncodeLang` method, and the first 30 bytes of the encoded UAP name are transferred to the HiRDB server (the name is truncated after 30 bytes even if the 30th byte is only part of a character). The UAP name that can be obtained by the HiRDB server is only the first 30 bytes after encoding.

18.7.10 getUapName

(a) Function

Gets the UAP name.

(b) Format

```
public String getUapName() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

UAP name

(e) Functional detail

Returns the UAP name that was set with the `setUapName` method. If a UAP name has not been set, `HiRDB_Type4_JDBC_Driver` is returned.

(f) Exceptions

None.

18.7.11 setUser

(a) Function

Sets an authorization identifier for database connection.

(b) Format

```
public void setUser ( String user ) throws SQLException
```

(c) Arguments

`String user`

Specifies an authorization identifier.

If the null value is specified, the current authorization identifier that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an authorization identifier.

When one of the following methods is executed, the authorization identifier and password that were specified with the `setUser` and `setPassword` methods are used in establishing a physical connection to the database:

- `getConnection` method (no arguments) of the `DataSource` interface
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface
- `getXAConnection` method of the `XADataSource` interface

If the `user` argument is the null value, the JDBC driver assumes that no authorization identifier has been set by this method.

For details about how the JDBC driver handles the situation when there is no setting, see *18.11 Connection information priorities*.

(f) Exceptions

If the length of the character string specified by the `user` argument is `0`, this method throws an `SQLException`.

18.7.12 getUser

(a) Function

Gets the authorization identifier for database connection.

(b) Format

```
String void getUser() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

Authorization identifier

(e) Functional detail

Returns the authorization identifier that was set by the `setUser` method. If an authorization identifier has not been set, the null value is returned.

(f) Exceptions

None.

18.7.13 setPassword

(a) Function

Sets a password for database connection.

(b) Format

```
public void setPassword ( String password ) throws SQLException
```

(c) Arguments

`String password`

Specifies a password.

If the null value is specified, the current password that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets a password.

When one of the following methods is executed, the authorization identifier and password that were specified with the `setUser` and `setPassword` methods are used in establishing a physical connection to the database:

- `getConnection` method (no arguments) of the `DataSource` interface
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface

- getXAConnection method of the XADataSource interface

If the password argument is the null value or a character string whose length is 0, the JDBC driver assumes that no password has been set by this method.

For details about how the JDBC driver handles the situation when there is no setting, see *18.11 Connection information priorities*.

(f) Exceptions

None.

18.7.14 getPassword

(a) Function

Gets the password for database connection.

(b) Format

```
public String getPassword() throws SQLException
```

(c) Arguments

None.

(d) Return value

String

Password

(e) Functional detail

Returns the password that was set by the setPassword method.

(f) Exceptions

None.

18.7.15 setXAOpenString

(a) Function

Sets an XA open character string.

(b) Format

```
public void setXAOpenString ( String xa_string ) throws
SQLException
```

(c) Arguments

`String xa_string`

Specifies an XA open character string.

If the null value is specified, the current XA open character string that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an XA open character string. This method is provided by the `XADataSource` interface only. Specify the XA open character string in the following format:

Format

HiRDB-environment-variable-group-identifier + *environment-variable-group-name-of-HiRDB-client*

Specify the HiRDB environment variable group identifier that was set by the `setDescription` method. Unlike when the environment variable group name of the HiRDB client is specified by the `setDescription` method, in this case the environment variable group name of the HiRDB client doe not need to be enclosed in quotation marks even if the name includes a single-byte at mark (`@`) or a single-byte space.

Setting example 1

When the path of the environment variable group name of the HiRDB client is `/HiRDB/HiRDB.ini`

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+/HiRDB/HiRDB.ini");
```

Setting example 2

When the path of the environment variable group name of the HiRDB client is `C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini` (∆ is a single-byte space)

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+C:\\Program∆Files\\HITACHI\\HiRDB
\\HiRDB.ini");
```

(f) Exceptions

None.

18.7.16 getXAOpenString

(a) Function

Gets the XA open character string.

(b) Format

```
public String getXAOpenString() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

XA open character string (if this information has not been specified, the method returns the null value)

(e) Functional detail

Returns the XA open character string that was set by the `setXAOpenString` method. This method is provided by the `XADataSource` interface only.

(f) Exceptions

None.

18.7.17 setXACloseString

(a) Function

Sets an XA close character string.

(b) Format

```
public void setXACloseString ( String xa_string ) throws
SQLException
```

(c) Arguments

`String xa_string`

Specifies an XA close character string.

If the null value is specified, the current XA close character string that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an XA close character string. This method is provided by the XADataSource interface only.

(f) Exceptions

None.

18.7.18 getXACloseString

(a) Function

Gets the XA close character string.

(b) Format

```
public String getXACloseString() throws SQLException
```

(c) Arguments

None.

(d) Return value

String

XA close character string (if this information has not been specified, the method returns the null value)

(e) Functional detail

Returns the XA close character string that was set by the setXACloseString method. This method is provided by the XADataSource interface only.

(f) Exceptions

None.

18.7.19 setLONGVARBINARY_Access

(a) Function

Sets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).

(b) Format

```
public void setLONGVARBINARY_Access ( String mode ) throws
SQLException
```

(c) Arguments

`String mode`

Specifies the method of accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types).

For this method, the value specified in the argument is not case sensitive.

`REAL`

Access the data with real data.

`LOCATOR`

Access the data using HiRDB's locator facility.

If the null value is specified, the current data access method that had been set is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the method for accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types). The default value when this method is not called is `REAL`.

Setting a value with this method is equivalent to setting the `LONGVARBINARY_ACCESS` property, which is shown in *18.2.2(2) User properties*.

(f) Exceptions

If a value other than `REAL` or `LOCATOR` is specified in the `mode` argument, this method throws a `java.sql.SQLException`.

(g) Notes

See *18.2.2(2)(i) LONGVARBINARY_ACCESS*.

18.7.20 getLONGVARBINARY_Access

(a) Function

Gets the method of accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types).

(b) Format

```
public String getLONGVARBINARY_Access()
```

(c) Arguments

None.

(d) Return value

`String`

Information about the method used to access data of the `LONGVARBINARY` JDBC SQL type (HiRDB's `BLOB` and `BINARY` data types)

`REAL`

The data is accessed using real data.

`LOCATOR`

The data is accessed using HiRDB's locator facility.

(e) Functional detail

Returns the information that was set by the `setLONGVARBINARY_Access` method.

(f) Exceptions

None.

18.7.21 setSQLInNum

(a) Function

Sets the maximum number of input `?` parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLInNum ( int inNum ) throws SQLException
```

(c) Arguments

`int inNum`

Specifies the maximum number of input `?` parameters in the SQL statements to be executed. The specification value range is from 1 to 30,000.

(d) Return value

None.

(e) Functional detail

Sets the number of input `?` parameter information items to be retrieved during SQL preprocessing.

If the actual number of `?` parameters is greater than the specification value of this method, the input `?` parameter information is retrieved after SQL preprocessing. The default value when this method is not called is 300.

At the time of database connection, the value specified by this method becomes the value of the HiRDB_for_Java_SQL_IN_NUM property, which is shown in *18.2.2(2) User properties*.

(f) Exceptions

If a value outside the range from 1 to 30,000 is specified in the argument, this method throws an SQLException.

(g) Notes

If the application does not execute SQL statements that have input ? parameters, you should specify 1 as the argument value.

18.7.22 getSQLInNum

(a) Function

Gets the maximum number of input ? parameters in the SQL statements to be executed.

(b) Format

```
public int getSQLInNum() throws SQLException
```

(c) Arguments

None.

(d) Return value

int

Maximum number of input ? parameters in the SQL statements to be executed, as set by setSQLInNum (if this information has not been specified, the method returns the default value (300))

(e) Functional detail

Gets the maximum number of input ? parameters in the SQL statements to be executed, as set by the setSQLInNum method.

(f) Exceptions

None.

18.7.23 setSQLOutNum

(a) Function

Sets the maximum number of retrieval items for the SQL statements to be executed.

(b) Format

```
public void setSQLOutNum ( int outNum ) throws SQLException
```

(c) Arguments

`int outNum`

Specifies the maximum number of retrieval items for the SQL statements to be executed. The specification value range is from 1 to 30,000.

(d) Return value

None.

(e) Functional detail

Sets the maximum number of retrieval items for the SQL statements to be executed.

This specification becomes the number of output items to be acquired during SQL preprocessing. The default value when this method is not called is `300`.

If the actual number of output items is greater than the specification value of this method, the output item information is acquired after SQL preprocessing.

The value specified by this method becomes the value of the `HiRDB_for_Java_SQL_OUT_NUM` property, which is shown in *18.2.2(2) User properties*.

(f) Exceptions

If the specified value is outside the range from 1 to 30,000, this method throws an `SQLException`.

(g) Notes

When there are no retrieval items, you should specify `1` as the argument value.

18.7.24 getSQLOutNum

(a) Function

Gets the maximum number of retrieval items for the SQL statements to be executed.

(b) Format

```
public int getSQLOutNum() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

Maximum number of retrieval items for the SQL statements to be executed, as set by `setSQLOutNum` (if this information has not been specified, the method returns the default value (300))

(e) Functional detail

Gets the maximum number of retrieval items for the SQL statements to be executed, as set by the `setSQLOutNum` method.

(f) Exceptions

None.

18.7.25 setSQLWarningLevel

(a) Function

Sets the warning retention level for warnings that occur during SQL execution.

(b) Format

```
public void setSQLWarningLevel ( String warningLevel ) throws
SQLException
```

(c) Arguments

`String warningLevel`

Specifies the retention level for the warning information that occurs during SQL execution.

The following values can be specified (for details about the relationships between the specification values and the warnings to be retained, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*):

- `IGNORE`
- `SQLWARN`
- `ALLWARN`

For this method, the value specified in the argument is not case sensitive. If the null value is specified, the current warning retention level that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the retention level for the warning information that occurs during SQL execution. The default value when this method is not called is SQLWARN.

The value specified by this method becomes the value of the HiRDB_for_Java_SQLWARNING_LEVEL property, which is shown in *18.2.2(2) User properties*.

(f) Exceptions

If the argument is a value other than the specification values shown above, this method throws an SQLException.

18.7.26 getSQLWarningLevel

(a) Function

Gets the warning retention level that was set by the setSQLWarningLevel method.

(b) Format

```
public String getSQLWarningLevel() throws SQLException
```

(c) Arguments

None.

(d) Return value

String

This is the warning retention level that was set by the setSQLWarningLevel method. For details about the return value and the warnings that are retained, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*.

(e) Functional detail

Returns the information that was set by the setSQLWarningLevel method. If no information has been set, the default value SQLWARN is returned.

(f) Exceptions

None.

18.7.27 setXALocalCommitMode

(a) Function

Sets whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.

(b) Format

```
public void setXALocalCommitMode ( boolean autoCommitMode )
throws SQLException
```

(c) Arguments

`boolean autoCommitMode`

Specifies the auto-commit facility:

`true`: Enable the auto-commit facility.

`false`: Disable the auto-commit facility.

(d) Return value

None.

(e) Functional detail

Sets the auto-commit facility during an XA connection. The default value is `false` (the auto-commit facility is disabled). The table below shows the relationships between this method's specification values and the JDBC driver operations.

Specification value	Condition	JDBC driver operation
`true`	Auto-commit default during `Connection` object generation	Enables auto-commit.
	Transaction termination by the `con.commit` or `con.rollback` method	Accepts normally.
	`setAutoCommit(true)` execution	Enables auto-commit.
	`setAutoCommit(false)` execution	Disables auto-commit.
`false` (default)	Auto-commit default during `Connection` object generation	Disables auto-commit.
	Transaction termination by the `con.commit` or `con.rollback` method	`SQLException`
	`setAutoCommit(true)` execution	`SQLException`
	`setAutoCommit(false)` execution	Normal termination (the driver does nothing because auto-commit cannot be enabled)

(f) Exceptions

None.

18.7.28 getXALocalCommitMode

(a) Function

Gets the setting information about whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.

(b) Format

```
public boolean getXALocalCommitMode() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting for the auto-commit facility:

`true`: The auto-commit facility is enabled.

`false`: The auto-commit facility is disabled.

(e) Functional detail

Gets the setting for the auto-commit facility.

(f) Exceptions

None.

18.7.29 setSQLWarningIgnore

(a) Function

Sets whether or not warnings returned from the database are to be discarded by the `Connection` class.

(b) Format

```
public void setSQLWarningIgnore ( boolean mode )
```

(c) Arguments

`boolean mode`

Specifies whether warnings are to be discarded.

`true`: Discard warnings.

`false`: Retain warnings.

(d) Return value

None.

(e) Functional detail

This method specifies whether warnings that occur in the `Connection` class are to be discarded. The default value is `false` (the warnings are to be retained).

Using this method is equivalent to specifying the `SQLWARNING_IGNORE` item described in *18.2.2(1) URL syntax* and the `SQLWARNING_IGNORE` property described in *18.2.2(2) User properties*.

For details about the relationships between the specification values and the warnings to be retained, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*.

(f) Exceptions

None.

18.7.30 getSQLWarningIgnore

(a) Function

Gets the setting information about whether or not warnings returned from the database are to be discarded by the `Connection` class.

(b) Format

```
public boolean getSQLWarningIgnore()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not warnings are to be discarded:

`true`: Discards warnings.

`false`: Retains warnings.

(e) Functional detail

This method acquires whether warnings that occur in the `Connection` class are to be discarded.

The method returns the information specified by the `setSQLWarningIgnore` method. If this information has not been specified, the method returns the default value `false`.

For details about the relationships between the return value and the warnings to be retained, see *18.4.12(2)(b) Issuing conditions for SQLWarning objects*.

(f) Exceptions

None.

18.7.31 setHiRDBCursorMode

(a) Function

Set whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(b) Format

```
public void setHiRDBCursorMode ( boolean mode )
```

(c) Arguments

`boolean mode`

Specifies one of the following values:

`true`: Validate objects of the `ResultSet` class after commit processing. When true is specified, objects of the following classes also become valid after commit processing:

- `Statement` class
- `PreparedStatement` class
- `CallableStatement` class

`false`: Invalidate objects of the `ResultSet` class after commit processing.

(d) Return value

None.

(e) Functional detail

Sets whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing. The default value if this method cannot be called is `false`.

If an invalidated `ResultSet` object executes an operation other than `close` method calling, this method throws an `SQLException`.

Executing this method is the same as setting the `HIRDB_CURSOR` item, which is shown in *18.2.2(1) URL syntax*.

(f) Exceptions

None.

(g) Notes

See *18.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

18.7.32 getHiRDBCursorMode

(a) Function

Gets the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(b) Format

```
public boolean getHiRDBCursorMode()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing:

`true`: Objects of the `ResultSet` class are valid after commit processing.

`false`: Objects of the `ResultSet` class become invalid after commit processing.

(e) Functional detail

Gets the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(f) Exceptions

None.

(g) Notes

None.

18.7.33 setNotErrorOccurred

(a) Function

Sets whether or not the calling of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.

(b) Format

```
public void setNotErrorOccurred ( boolean mode )
```

(c) Arguments

boolean mode

Specifies whether or not occurrences of connectionErrorOccurred are to be suppressed:

true: Suppress the calling of connectionErrorOccurred.

false: Do not suppress the calling of connectionErrorOccurred (default).

(d) Return value

None.

(e) Functional detail

Specifies the setting for suppressing the calling of ConnectionEventListener.connectionErrorOccurred, which is called when an error occurs while ConnectionPoolDataSource or XADataSource is being used.

If this method is not set, connectionErrorOccurred is called. Normally, do not set this method or set false.

(f) Exceptions

None.

18.7.34 getNotErrorOccurred

(a) Function

Gets the setting information about whether or not the calling of ConnectionEventListener.connectionErrorOccurred is to be suppressed.

(b) Format

```
public boolean getNotErrorOccurred()
```

(c) Arguments

None.

(d) Return value

boolean

This is the setting information about whether or not `ConnectionEventListener.connectionErrorOccurred` is called:

`true`: `connectionErrorOccurred` is not called.

`false`: `connectionErrorOccurred` is called (default).

(e) Functional detail

Gets the setting information about whether or not `ConnectionEventListener.connectionErrorOccurred` is to be called when a fatal connection error occurs while `ConnectionPoolDataSource` or `XADataSource` is being used. If no setting information has been set, this method returns `false`.

(f) Exceptions

None.

18.7.35 setEnvironmentVariables

(a) Function

Sets client environment definitions for HiRDB.

(b) Format

```
public void setEnvironmentVariables ( String variables ) throws
SQLException
```

(c) Arguments

`String variables`

Specifies HiRDB client environment definitions in the format shown below:

Format

"*variable-name*=*value*;*variable-name*=*value*;...;*variable-name*=*value*"

A specification example is shown below:

Specification example

```
setEnvironmentVariables
("PDFESHOST=FES1;PDCWAITTIME=0");
```

If the null value is specified, the current client environment definitions that had been set by this method are invalidated, and the settings are returned to their initial status.

(d) Return value

None.

(e) Functional detail

This method specifies HiRDB client environment definitions.

For details about the client environment definitions that can be specified by the JDBC driver, see *18.10 Supported client environment definitions*. If a client environment definition that cannot be specified by the JDBC driver is specified for a variable, the specification is ignored. Note that the variable names are case sensitive.

For details about the priorities among connection information items that have multiple setting methods, see *18.11 Connection information priorities*.

This method does not check each specification of the client environment definitions. The specification values are checked during connection to the database, and an `SQLException` is thrown if an error is detected.

(f) Exceptions

None.

18.7.36 getEnvironmentVariables

(a) Function

Gets the client environment definitions for HiRDB.

(b) Format

```
String void getEnvironmentVariables()
```

(c) Arguments

None.

(d) Return value

`String`

HiRDB client environment definitions (if this information has not been specified, the method returns the null value)

(e) Functional detail

Gets the client environment definitions of HiRDB.

(f) Exceptions

None.

18.7.37 setEncodeLang

(a) Function

Sets the name of the conversion character set for data conversion.

(b) Format

```
public void setEncodeLang ( String encode_lang ) throws
SQLException
```

(c) Arguments

`String encode_lang`

Specifies the name of the conversion character set. You must select a name from the list of encodings shown under *Internationalization* in the *Java*^{TM} *2 SDK, Standard Edition* documentation.

The table below shows the HiRDB character encodings and the corresponding conversion character sets.

HiRDB character encoding (character encoding set with pdntenv or pdsetup command)	Conversion character set to be specified
lang-c	ISO8859_1
sjis	SJIS or MS932#
ujis	EUC_JP
utf-8	UTF-8
chinese	EUC_CN
chinese-gb18030	GB18030

Note:

If the specified conversion character set name is not in compliance with the applicable name shown in this table, the operation of the JDBC driver is not guaranteed.

#

The specification of SJIS or MS932 depends on the handling of Windows special characters in the application.

When OFF is specified, the JDBC driver assumes that the applicable conversion character set name shown in this table was specified. If the HiRDB character encoding is `sjis`, the conversion character set determined by the OS running the JDBC driver is as follows.

In UNIX: `SJIS`

In Windows: `MS932`

If the null value is specified, the current conversion character set name that had been set by this method is invalidated, and the setting is returned to its initial status.

Note that the specification values are case sensitive (except for `OFF`).

(d) Return value

None.

(e) Functional detail

The conversion character set that was specified by this method is used for carrying out the following data conversions:

- Conversion to character data (Unicode) when the application uses `String` to get data that was retrieved from HiRDB
- Conversion to binary data when the application uses `String` to set a value in HiRDB

If this method is not specified, the JDBC driver converts characters using the applicable conversion character set shown in the table above. However, the JDBC driver uses the default character conversion set of the Java Virtual Machine to convert the following items:

- Specification value of `setUapName`
- Authorization identifier and password (values specified by `setUser`, `setPassword`, and `getConnection`)
- Specification values of client environment definitions specified by `setEnvironmentVariables`
- Specification values of environment variables specified by the environment variable group name of the HiRDB client

(f) Exceptions

If the specified conversion character set is not supported by the Java Virtual Machine, this method throws an `SQLException`.

18.7.38 getEncodeLang

(a) Function

Gets the conversion character set for data conversion name that was set.

(b) Format

```
public String getEncodeLang()
```

(c) Arguments

None.

(d) Return value

`String`

Conversion character set name (if a conversion character set name has not been specified by the `setEncodeLang` method, the method returns the null value)

(e) Functional detail

Returns the conversion character set name that was set by the `setEncodeLang` method.

(f) Exceptions

None.

18.7.39 setMaxBinarySize

(a) Function

Sets the maximum data size for retrieval of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public void setMaxBinarySize ( int size ) throws SQLException
```

(c) Arguments

`int size`

Specifies the maximum data size, in the range from 0 to 2,147,483,647.

If `0` is specified, the defined length of the data to be retrieved is set as the maximum size.

(d) Return value

None.

(e) Functional detail

Sets the maximum data size (bytes) when data of the `LONGVARBINARY` JDBC SQL type is retrieved.

When the JDBC driver retrieves LONGVARBINARY data, it allocates memory of the defined length because it cannot recognize the actual data length until it retrieves the data. Consequently, if the JDBC driver retrieves values from a column that is very large, such as 2,147,483,647 bytes (the maximum size for HiRDB's BINARY and BLOB data types), it attempts to allocate memory space of the defined length (2,147,483,647 bytes) as the defined length. Depending on the execution environment, this may cause a memory shortage.

You should use this method to specify the maximum length of the data that is actually stored. If the defined length of the BINARY and BLOB data to be retrieved is larger than the size specified by this method, the JDBC driver truncates the retrieved data to the specified size. When data has been truncated, the JDBC driver receives a warning from the HiRDB server when the next method of ResultSet is executed. The JDBC driver responds to the received warning by throwing an SQLException or issuing (or ignoring) an SQLWarning, as specified by the setLONGVARBINARY_TruncError value.

If a maximum data size has not been set by this method, the defined length of the data to be retrieved is used as the maximum data size.

(f) Exceptions

If a negative value is specified, this method throws an SQLException.

(g) Notes

Any value specified for this method is not effective when LOCATOR is specified in the mode argument of the setLONGVARBINARY_Access method. In such a case, the JDBC driver allocates an area based on the actual data length and retrieves all of the data.

18.7.40 getMaxBinarySize

(a) Function

Gets the maximum data size for retrieval of data of the LONGVARBINARY type (a JDBC SQL type).

(b) Format

```
public int getMaxBinarySize()
```

(c) Arguments

None.

(d) Return value

int

Value specified as the maximum data size

(e) Functional detail

Returns the maximum data size for retrieving data of the `LONGVARBINARY` type (a JDBC SQL type), as set by the `setMaxBinarySize` method.

If a maximum data size has not been set by the `setMaxBinarySize` method, `0` is returned.

(f) Exceptions

None.

18.7.41 setStatementCommitBehavior

(a) Function

Sets whether or not statement objects are to remain valid after a transaction is committed. Here, *statement objects* refer to the following classes:

- `Statement` class
- `PreparedStatement` class
- `CallableStatement` class

(b) Format

```
public void setStatementCommitBehavior ( boolean mode ) throws
SQLException
```

(c) Arguments

`boolean mode`

Specifies whether statement objects are to be valid both before and after a transaction is terminated by commit processing:

`true`: Validate statement objects after a transaction is completed.

`false`: Invalidate statement objects after a transaction is completed.

(d) Functional detail

Sets whether or not statement objects are to remain valid after a transaction is committed. The default when this method is not called is `true`.

Executing this method is the same as setting the `STATEMENT_COMMIT_BEHAVIOR` item, which is shown in *18.2.2(1) URL syntax*.

(e) Exceptions

None.

(f) Notes

See *18.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

18.7.42 getStatementCommitBehavior

(a) Function

Gets setting information about whether or not statement objects are to remain valid even after a transaction is committed. Here, *statement objects* refer to the following classes:

- `Statement` class
- `PreparedStatement` class
- `CallableStatement` class

(b) Format

```
public boolean getStatementCommitBehavior() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

Indicates whether statements objects are to remain valid after a transaction is terminated by commit processing:

`true`: The statement objects are to remain valid.

`false`: The statement objects are not to remain valid.

(e) Functional detail

Gets the setting information about whether objects of the following classes are to remain valid after commit execution:

- `Statement` class
- `PreparedStatement` class
- `CallableStatement` class

This method returns the setting value of the `setStatementCommitBehavior` method. If no value has been set, `true` is returned.

(f) Exceptions

None.

(g) Notes

None.

18.7.43 setLONGVARBINARY_AccessSize

(a) Function

Sets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.

(b) Format

```
public void setLONGVARBINARY_AccessSize ( int access_size )
throws SQLException
```

(c) Arguments

int access_size

Specifies the data length (kilobytes) to be requested. The specification value range is from 0 to 2,097,151 (the default is 0). If 0 is specified, the entire data is requested at once.

(d) Return value

None.

(e) Functional detail

Sets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.

For example, if 20 is specified for the access_size argument and the application uses the getBytes method of ResultSet to retrieve 100 kilobytes of LONGVARBINARY data stored in the database, the JDBC driver retrieves the data by dividing the operation into five executions of 20 kilobytes each.

This specification value becomes invalid if a value other than LOCATOR is specified in the mode argument of the setLONGVARBINARY_Access method.

Specifying a value for this method is equivalent to setting the HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE property, which is shown in *18.2.2(2) User properties*.

(f) Exceptions

If a value outside the range from 0 to 2,097,151 is specified in the access_size argument, this method throws a java.sql.SQLException.

(g) Notes

See *18.2.2(2)(i) LONGVARBINARY_ACCESS*.

18.7.44 getLONGVARBINARY_AccessSize

(a) Function

Gets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.

(b) Format

```
public int getLONGVARBINARY_AccessSize() throws SQLException
```

(c) Arguments

None.

(d) Return value

int

Length of the data (in kilobytes) that can be requested at one time (if this information has not been specified, the method returns 0)

(e) Functional detail

Gets the LONGVARBINARY (JDBC SQL type) data length for one access request to the HiRDB server. This method returns the setting value of the setLONGVARBINARY_AccessSize method.

(f) Exceptions

None.

18.7.45 setLONGVARBINARY_TruncError

(a) Function

Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the LONGVARBINARY type (a JDBC SQL type).

(b) Format

```
public void setLONGVARBINARY_TruncError ( boolean mode ) throws
SQLException
```

(c) Arguments

`boolean mode`

Specifies whether or not an exception is to be thrown when truncation occurs:

`true`

Throw an exception.

`false`

Do not throw an exception.

(d) Return value

None.

(e) Functional detail

Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type). If this method is not set, the JDBC driver assumes that `true` was specified.

The specification value of this method becomes invalid if `IGNORE` is specified in the `warningLevel` argument of the `setSQLWarningLevel` method. In such a case, the JDBC driver operates as if `false` were specified.

A truncation that occurs when `LONGVARBINARY` data is retrieved refers to the action that occurs when the flowing conditional expression is satisfied:

actual-length-of-LONGVARBINARY-data-retrieved-by-SQL-execution > *data-length-specified-by-setMaxBinarySize*

(f) Exceptions

None.

18.7.46 getLONGVARBINARY_TruncError

(a) Function

Gets the setting information about whether or not an exception is to be thrown if truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public boolean getLONGVARBINARY_TruncError()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether not an exception is to be thrown when truncation occurs:

`true`

An exception is thrown.

`false`

An exception is not thrown.

(e) Functional detail

Gets the setting information about whether or not an exception is to be thrown when truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type).

(f) Exceptions

None.

18.7.47 setStatementCloseBehavior

(a) Function

Sets whether preprocessing results are to be ignored during execution of the `close` method of `Statement` (`Statement`, `PreparedStatement`, and `CallableStatement` classes).

(b) Format

```
public synchronized void setStatementCloseBehavior(boolean
mode)
```

(c) Arguments

boolean mode

Specifies whether preprocessing results are to be ignored during execution of the `close` method of `Statement`:

true

Ignore preprocessing results.

false

Do not ignore preprocessing results.

(d) Return value

None.

(e) Functional detail

This method sets whether preprocessing results are to be ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes).

If this information is not specified, the method assumes that false is specified.

Specifying a value with this method is equivalent to specifying the HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR property described in *18.2.2(2) User properties*.

(f) Exceptions

None.

18.7.48 getStatementCloseBehavior

(a) Function

Gets the setting information about whether preprocessing results are ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes).

(b) Format

```
public boolean getStatementCloseBehavior()
```

(c) Arguments

None.

(d) Return value

true

Preprocessing results are ignored.

false

Preprocessing results are not ignored.

(e) Functional detail

Gets setting information about whether preprocessing results are ignored during execution of the close method of Statement (Statement, PreparedStatement, and CallableStatement classes). This method returns the value set by the setStatementCloseBehavior method. If this information has not been specified, the method returns false.

(f) Exceptions

None.

18.7.49 setHiRDBINI

(a) Function

Sets a directory for the HiRDB.INI file when client environment variables specified in the HiRDB.INI file are to be used.

(b) Format

```
public synchronized void setHiRDBINI (String dir)
```

(c) Arguments

string dir

Specifies the absolute path of the directory that is to contain the HiRDB.INI file. If null is specified, the method assumes that this argument is not specified.

(d) Return value

None.

(e) Functional detail

When client environment variables specified in a HiRDB.INI file are to be used, this method specifies the absolute path for the directory that is to contain the HiRDB.INI file. For details about this method, see the description of HiRDB_INI in *18.2.2(1)(b) Explanation of URL items*.

(f) Exceptions

None.

18.7.50 getHiRDBINI

(a) Function

Gets the directory for the HiRDB.INI file (set by the setHiRDBINI method).

(b) Format

```
public String getHiRDBINI()
```

(c) Arguments

None.

(d) Return value

String

Absolute path of the directory specified by the `setHiRDBINI` method for the `HiRDB.INI` file. If this information has not been specified, the method returns `null`.

(e) Functional detail

This method returns the absolute path of the directory specified by the `setHiRDBINI` method.

(f) Exceptions

None.

18.7.51 setBatchExceptionBehavior

(a) Function

Sets whether a JDBC standard-compliant update count is to be set as the return value of the `getUpdateCounts` method of `java.sql.BatchUpdateException`.

(b) Format

```
public synchronized void setBatchExceptionBehavior(boolean
mode)
```

(c) Arguments

boolean mode

Specifies whether a JDBC standard-compliant update count is to be set as the return value of the `getUpdateCounts` method of `java.sql.BatchUpdateException`.

true: Set a JDBC standard-compliant update count.

false: Set a HiRDB-specific update count.

(d) Return value

None.

(e) Functional detail

Specifies whether a JDBC standard-compliant update count is to be set as the return value of the `getUpdateCounts` method of `java.sql.BatchUpdateException`.

The default value applied when this method is not called is TRUE.

Specifying a value with this method is equivalent to specifying the HiRDB_for_Java_ BATCHEXCEPTION_BEHAVIOR property described in *18.2.2(2) User properties*.

(f) Exceptions

None.

18.7.52 getBatchExceptionBehavior

(a) Function

Gets the setting information about whether a JDBC standard-compliant update count is being set as the return value of the getUpdateCounts method of java.sql.BatchUpdateException, which was specified in setBatchExceptionBehavior.

(b) Format

```
public synchronized boolean getBatchExceptionBehavior()
```

(c) Arguments

None.

(d) Return value

boolean

Returns a value indicating whether a JDBC standard-compliant update count is being set as the return value of the getUpdateCounts method of java.sql.BatchUpdateException, which was specified in setBatchExceptionBehavior. If this information has not been specified, the method returns the default value TRUE.

true: A JDBC standard-compliant update count is being set.

false: A HiRDB-specific update count is being set.

(e) Functional detail

This method specifies the setting as to whether a JDBC standard-compliant update count is being set as the return value of the getUpdateCounts method of java.sql.BatchUpdateException, which was specified in setBatchExceptionBehavior.

(f) Exceptions

None.

18.8 Data types

18.8.1 Mapping SQL data types

There is not an exact match between HiRDB's SQL data types and JDBC's SQL data types. For this reason, the JDBC driver performs mapping (conversion) between JDBC's SQL data types and the SQL data types of the HiRDB to be connected. If an unmappable SQL data type is used for data access, the JDBC driver throws an `SQLException`. If an SQL statement that uses HiRDB's `ROW` type, which cannot be mapped to any of JDBC's SQL data types, is executed for an HiRDB server that uses little endian, the JDBC driver throws an `SQLException` that includes the `KFPA11104-E` message indicating a syntax error.

The SQL data types are mapped with `get`*XXX* and `set`*XXX* methods of the `ResultSet`, `PreparedStatement`, and `CallableStatement` classes. For details about the mapping rules for the SQL data types and the `get`*XXX* and `set`*XXX* methods, see the documentation for the JDBC1.0 standard and JDBC2.0 basic standard.

The following table shows the correspondences between the HiRDB and the JDBC SQL data types.

Table 18-65: SQL data type correspondences between HiRDB and JDBC (Type4 JDBC driver)

HiRDB's SQL data type	JDBC's SQL data type
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL, NUMERIC	DECIMAL (NUMERIC)#1
FLOAT, DOUBLE PRECISION	FLOAT (DOUBLE)#1
SMALLFLT, REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR (LONGVARCHAR)#1
NCHAR	CHAR
NVARCHAR	VARCHAR (LONGVARCHAR)#1
MCHAR	CHAR
MVARCHAR	VARCHAR (LONGVARCHAR)#1

HiRDB's SQL data type	JDBC's SQL data type
DATE	DATE
TIME	TIME
BLOB	LONGVARBINARY (BINARY, VARBINARY, BLOB)#1
BINARY	LONGVARBINARY (BINARY, VARBINARY, BLOB)#1
TIMESTAMP	TIMESTAMP
BOOLEAN#2	BIT

#1

Data types shown in parentheses are supported only when JDBC's SQL data types are specified in the arguments of the setNull, setObject, or registerOutParameter method. They are not supported during mapping from HiRDB's SQL data types to JDBC's SQL data types.

#2

This refers to a BOOLEAN column in a ResultSet object that is generated by the getTypeInfo method of DatabaseMetaData.

18.8.2 Mapping during retrieval data acquisition

The tables below show the mapping between get*XXX* methods of the ResultSet and CallableStatement classes and JDBC's SQL data types. If a get*XXX* method is called for one of JDBC's unmappable SQL data types, the JDBC driver throws an SQLException.

Table 18-66: Mapping between getXXX methods and JDBC's SQL data types (1/2)

getXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getByte	Y	Y	Y	Y	Y	Y#1
getShort	Rec.	Y	Y	Y	Y	Y#1
getInt	Y	Rec.	Y	Y	Y	Y#1
getLong	Y	Y	Y	Y	Y	Y#1
getFloat	Y	Y	Y	Rec.	Y	Y#1

getXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
`getDouble`	Y	Y	Rec.	Y	Y	Y[#1]
`getBigDecimal`	Y	Y	Y	Y	Rec.	Y[#1]
`getBoolean`	Y	Y	Y	Y	Y	Y
`getString`	Y	Y	Y	Y	Y	Rec.
`getBytes`	--	--	--	--	--	--
`getDate`	--	--	--	--	--	Y[#1]
`getTime`	--	--	--	--	--	Y[#1]
`getTimestamp`	--	--	--	--	--	Y[#1]
`getAsciiStream`	--	--	--	--	--	Y
`getBinaryStream`	--	--	--	--	--	--
`getObject`	Y	Y	Y	Y	Y	Y
`getCharacterStream`	--	--	--	--	--	Y
`getArray`	--	--	--	--	--	--
`getBlob`	--	--	--	--	--	--

Table 18-67: Mapping between getXXX methods and JDBC's SQL data types (2/2)

getXXX method	JDBC's SQL data type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`getByte`	Y[#1]	--	--	--	--	--
`getShort`	Y[#1]	--	--	--	--	--
`getInt`	Y[#1]	--	--	--	--	--
`getLong`	Y[#1]	--	--	--	--	--
`getFloat`	Y[#1]	--	--	--	--	--

getXXX method	JDBC's SQL data type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`getDouble`	Y#1	--	--	--	--	--
`getBigDecimal`	Y#1	--	--	--	--	--
`getBoolean`	Y	--	--	--	--	--
`getString`	Rec.	Y	Y	Y	Y	--
`getBytes`	--	--	--	--	Y	--
`getDate`	Y#1	Rec.#2	--	Y	--	--
`getTime`	Y#1	--	Rec.	Y	--	--
`getTimestamp`	Y#1	Y	--	Rec.	--	--
`getAsciiStream`	Y	--	--	--	Y	--
`getBinaryStream`	--	--	--	--	Rec.	--
`getObject`	Y	Y	Y	Y	Y	--
`getCharacterStream`	Y	--	--	--	Y	--
`getArray`	--	--	--	--	--	Rec.
`getBlob`	--	--	--	--	Y	--

Legend:

Rec.: Mapping is recommended

Y: Can be mapped. Note, however, that data loss or a conversion error may occur depending on the format of the conversion-source data.

--: Cannot be mapped.

#1

If there are any single-byte spaces preceding or following the character string data retrieved from the database during conversion by this method, the JDBC driver removes them. After removing the single-byte spaces, the JDBC driver converts the data to the Java data type returned by the `get`*XXX* method.

Note the following items when data is converted to a Java data type:

- If the character string data contains a fractional part and the `getByte`, `getInt`, getShort, or getLong method is executed, the JDBC driver discards the fractional part and then converts and returns only the integer.

- If the character string data contains double-byte characters, the JDBC driver throws an SQLException without converting the data. Double-byte characters include double-byte spaces used for padding when a character string shorter than the defined column length is stored in a column of HiRDB's NCHAR data type.
- If overflow occurs after character string data is converted to a Java data type, the JDBC driver throws an SQLException.

#2

When the JDBC SQL type is the DATE type and a java.util.Calender object is specified in a setDate method that is then executed, the JDBC driver uses the specified java.util.Calendar object to convert the data, discards the time data, and stores only the date data in the database. Because the time data is discarded, if an application specifies a java.util.Calendar object in the getDate method and executes the method to retrieve the data that was stored with the setDate method, the retrieved date may differ from the one that was specified in the setDate method.

Example

In this example, the UAP uses Japan Standard Time as the default time zone and specifies a java.util.Calendar object that uses Greenwich Mean Time in the setDate and getDate methods.

When the UAP specifies a java.sql.Date object representing 2005-10-03 in the setDate method and executes the method, the JDBC driver supplements 00:00:00 to the time portion, and then subtracts 9 hours because of the time zone difference. The result is 2005-10-02 15:00:00, and the JDBC driver stores the date portion 2005-10-02 of the result in the database. When the UAP uses the getDate method to retrieve this data, the JDBC driver gets the date portion 2005-10-02 from the database, supplements 00:00:00 to the time portion, and adds 9 hours because of the time difference to produce 2005-10-02 09:00:00. Consequently, in the java.sql.Date object used as the return value of the getDate method, the JDBC driver sets 2005-10-02, which differs from 2005-10-03, as was specified in the SetDate method.

18.8.3 Mapping when a ? parameter is set

The table below lists the set*XXX* methods of the PreparedStatement and CallableStatement classes and shows the corresponding JDBC SQL types that are mapped. If a JDBC SQL type cannot be used, the set*XXX* method throws an SQLException.

The setCharacterStream method has been added as a replacement for the setUnicodeStream method, because the JDBC2.0 basic standard does not recommend the latter method.

Table 18-68: JDBC SQL types mapped by the setXXX methods

setXXX method of PreparedStatement class	Mapped JDBC SQL type
setCharacterStream	CHAR or VARCHAR
setRef#	REF
setBlob	LONGVARBINARY
setClob#	CLOB
setArray	ARRAY

#

The JDBC driver cannot use this method.

The following tables show the mapping between the set*XXX* methods of the PreparedStatement and CallableStatement classes and JDBC's SQL types.

Table 18-69: Mapping between the setXXX methods and JDBC's SQL data types (1/2)

setXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL#3	CHAR
setByte	Y	Y	Y	Y	Y	Y
setShort	Rec.	Y	Y	Y	Y	Y
setInt	Y	Rec.	Y	Y	Y	Y
setLong	Y	Y	Y	Y	Y	Y
setFloat	Y	Y	Y	Rec.	Y	Y
setDouble	Y	Y	Rec.	Y	Y	Y
setBigDecimal	Y	Y	Y	Y	Rec.	Y
setBoolean	Y	Y	Y	Y	Y	Y
setString	Y	Y	Y	Y	Y	Rec.
setBytes	--	--	--	--	--	--
setDate	--	--	--	--	--	Y

setXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL[#3]	CHAR
`setTime`	--	--	--	--	--	Y
`setTimestamp`[#1]	--	--	--	--	--	Y
`setAsciiStream`	--	--	--	--	--	Y
`setBinaryStream`	--	--	--	--	--	--
`setObject`[#2]	Y	Y	Y	Y	Y	Y
`setCharacterStream`	--	--	--	--	--	Y[#4]
`setArray`	--	--	--	--	--	--
`setBlob`	--	--	--	--	--	--

Table 18-70: Mapping between the setXXX methods and JDBC's SQL data types (2/2)

setXXX method	JDBC's SQL data type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`setByte`	Y	--	--	--	--	--
`setShort`	Y	--	--	--	--	--
`setInt`	Y	--	--	--	--	--
`setLong`	Y	--	--	--	--	--
`setFloat`	Y	--	--	--	--	--
`setDouble`	Y	--	--	--	--	--
`setBigDecimal`	Y	--	--	--	--	--
`setBoolean`	Y	--	--	--	--	--
`setString`	Rec.	Y	Y	Y	Y	--
`setBytes`	--	--	--	--	Y	--
`setDate`	Y	Rec.[#5]	--	Y	--	--
`setTime`	Y	--	Rec.	Y	--	--

setXXX method	JDBC's SQL data type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVAR BINARY	ARRA Y
`setTimestamp`[#1]	Y	Y	--	Rec.	--	--
`setAsciiStream`	Y	--	--	--	Y	--
`setBinaryStream`	--	--	--	--	Y	--
`setObject`[#2]	Y	Y	Y	Y	Y	--
`setCharacterStream`	Y[#4]	--	--	--	Y[#4]	--
`setArray`	--	--	--	--	--	Y
`setBlob`	--	--	--	--	Y	--

Legend:

Rec.: Mapping is recommended

Y: Can be mapped. Note, however, that data loss or a conversion error may occur depending on the format of the conversion-source data.

--: Cannot be mapped.

#1

If a `set`*XXX* method specifies a value for a `?` parameter of HiRDB's `TIMESTAMP` data type, and the `?` parameter and the value have different precisions for the fractional seconds part, the JDBC driver performs one of the following operations:

- If the value has a larger fractional seconds precision than the `?` parameter: truncates the fractional seconds part of the value.
- If the value has a smaller fractional seconds precision than the `?` parameter: expands the fractional seconds part of the value.

#2

Objects of the `InputStream` class and the `Reader` class (including subclasses) cannot be specified in the `setObject` method.

#3

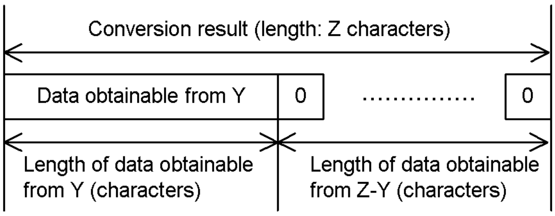
If a `set`*XXX* method specifies a value for a `?` parameter of HiRDB's `DECIMAL` data type, and the `?` parameter and the value have different precisions and decimal scaling positions, the JDBC driver performs one of the following operations, as applicable:

- When the value has a larger precision than the ? parameter: Throws an SQLException.
- When the value has a smaller precision than the ? parameter: Expands the precision.
- When the value has a larger decimal scaling position than the ? parameter: Truncates the value according to the actual scaling position.
- When the value has a smaller decimal scaling position than the ? parameter: Adds zeros to expand the decimal scaling position.

#4

If the length of the data that can be retrieved from a java.io.Reader object is shorter than the length specified in the arguments, the JDBC driver adds zeros as shown below until the length specified in the arguments is reached:

```
int X,Z;
java.io.Reader Y;
setCharacterStream(X,Y,Z)
```


Conversion result (length: Z characters)
Data obtainable from Y
0
....
0
Length of data obtainable from Y (characters)
Length of data obtainable from Z-Y (characters)

#5

When the JDBC SQL type is the DATE type and a java.util.Calender object is specified in a setDate method that is then executed, the JDBC driver uses the specified java.util.Calendar object to convert the data, discards the time data, and stores only the date data in the database. Because the time data is discarded, if an application specifies a java.util.Calendar object in the getDate method and executes the method to retrieve the data that was stored with the setDate method, the retrieved date may differ from the one that was specified in the setDate method.

Example

In this example, the UAP uses Japan Standard Time as the default time zone and specifies a java.util.Calendar object that uses Greenwich Mean Time in the setDate and getDate methods.

When the UAP specifies a java.sql.Date object representing 2005-10-03 in the setDate method and executes the method, the JDBC driver supplements 00:00:00 to the time portion, and then subtracts 9 hours because of the time zone difference. The result is 2005-10-02 15:00:00, and the JDBC driver stores the date portion 2005-10-02 of the result in the database. When the UAP uses the getDate method to retrieve this data, the JDBC driver gets the date portion 2005-10-02 from the database, supplements 00:00:00 to the time portion, and adds 9 hours because of the time difference to produce 2005-10-02 09:00:00. Consequently, in the java.sql.Date object used as the return value of the getDate method, the JDBC driver sets 2005-10-02, which differs from 2005-10-03, which was specified in the SetDate method.

18.8.4 Data conversion of TIME, DATE, and TIMESTAMP columns

(1) setTime, setDate, setTimestamp, and setString methods

This item explains the conversion process when data of HiRDB's TIME, DATE, or TIMESTAMP data type is set in the setTime, setDate, setTimestamp, or setString method.

When the setTime, setDate, setTimestamp, or setString method is used to set data in a column of HiRDB's TIME, DATE, or TIMESTAMP data type, data conversion takes place according to the HiRDB data type.

The following table shows the conversion processing for combinations of the different column data types and methods.

Table 18-71: Conversion processing for combinations of the TIME, DATE, and TIMESTAMP types and the setXXX methods

setXXX method	HiRDB data type		
	TIME type	DATE type	TIMESTAMP type
setTime(Time Obj)[#1]	Stores the UAP setting value in the database without any conversion.	Throws an SQLException.	Stores in the database data that has 1970-01-01 added before the UAP setting value *hh*:*mm*:*ss*[.000000].
setDate(Date Obj)[#2]	Throws an SQLException.	Stores the UAP setting value in the database without any conversion.	Stores data in the database that has 00:00:00[.000000] added after the UAP setting value *yyyy*-*MM*-*DD*.
setTimestamp(Timestamp Obj)[#3]	Throws an SQLException.	Stores in the database the data formed when *yyyy*-*MM*-*DD* is removed from the UAP setting value.	Stores the UAP setting value in the database without any conversion.

setXXX method	HiRDB data type		
	TIME type	DATE type	TIMESTAMP type
`setString`(character string in *hh*:*mm*:*ss* format)	Converts the specified time with `java.sql.Time.valueOf()` and stores the result in the database.#5	Throws an `SQLException`.	Throws an `SQLException`.
`setString`(character string in *yyyy*-*MM*-*DD* format)	Throws an `SQLException`.	Converts the specified date with `java.sql.Date.valueOf()` and stores the result in the database.#5	Throws an `SQLException`.
`setString`(character string in *yyyy*-*MM*-*DD* Δ *hh:mm:ss* [.*ffffff*] format)#4	Throws an `SQLException`.	Throws an `SQLException`.	Converts the specified date/time with `java.sql.Timestamp.valueOf()` and stores the result in the database.#5

Note:

If a non-existent date or time is specified, the specified value is returned by the Java Virtual Machine.

#1

`Time Obj` is an object that has the value of a `java.sql.Time` object with the format *hour*:*minute*:*second*.

#2

`Date Obj` is an object that has the value of the `java.sql.Date` object with the format *year*-*month*-*day*.

#3

`Timestamp Obj` is an object that has the value of the `java.sql.Timestamp` object with the format *year*-*month*-*day hour*:*minute*:*second*:*nanosecond*.

#4

For [.*ffffff*], the number of digits after the decimal point depends on the precision of HiRDB's `TIMESTAMP` type.

Δ represents a single-byte space character.

#5

The result when a non-existent date or time is specified depends on `java.sql.Time.valueOf()`, `java.sql.Date.valueOf()`, or `java.sql.Timestamp.valueOf()`:

Example 1: `25:00:00` becomes `01:00:00`.

Example 2: `2000-01-32` becomes `2000-02-01`.

Example 3: `1582-10-05` becomes `1582-10-15` (switching from the Julian to the Gregorian calendar).

(2) getTime, getDate, and getTimestamp methods

This item explains the conversion process when data of HiRDB's `TIME`, `DATE`, `TIMESTAMP` or character string (`CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, or `NVARCHAR`) data type is set in the `getTime`, `getDate`, or `getTimestamp` method.

When the `getTime`, `getDate`, or `getTimestamp` method is used to set data in a column of HiRDB's `TIME`, `DATE`, `TIMESTAMP`, or character string data type, data conversion takes place according to the HiRDB data type.

The following table shows the conversion processing for combinations of the different column data types and methods.

Table 18-72: Conversion processing for combinations of the TIME, DATE, TIMESTAMP, and character string types and the getXXX methods

getXXX method	**HiRDB data type**			
	TIME type	**DATE type**	**TIMESTAMP type**	**Character string type**
`getTime()` [#2]	Gets the value stored in the database and sets it as the `java.sql.Time` object without any conversion.[#1]	Throws an `SQLException`.	Removes the *hour*:*minute*:*second* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Time` object.[#1]	Gets only an *hh*:*mm*:*ss* character string expression of the `TIME` type as the `java.sql.Time` object. For other expressions, the method throws an exception.
`getDate()` [#2]	Throws an `SQLException`.	Gets the value stored in the database and sets it as the `java.sql.Date` object without any conversion.[#1]	Removes the *year*-*month*-*day* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Date` object.[#1]	Gets only a *yyyy*-*MM*-*DD* character string expression of the `DATE` type as the `java.sql.Date` object. For other expressions, the method throws an exception.

getXXX method	HiRDB data type			
	TIME type	DATE type	TIMESTAMP type	Character string type
getTimestamp()#2	Throws an SQLException.	Appends 00:00:00.000000 to the DATE data retrieved from the database and sets the result as the java.sql.Timestamp object.	Gets the value stored in the database and sets it as the java.sql.Timestamp object without any conversion.	Gets only a *yyyy-MM-DD*Δ*hh:mm:ss*[.*ffffff*] character string expression of the TIMESTAMP type as the java.sql.Timestamp object (Δ is a single-byte space character). For other expressions, the method throws an SQLException.

Legend:

Character string types: CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, and NVARCHAR

#1

The setting value of an unspecified date item (*year-month-day*) is 1970-01-01, and the setting value of an unspecified time item (*hour*:*minute*:*second*.*millisecond*) is 00:00:00.000000.

#2

The date and time stored in the database may be different from the date and time obtained from java.sql.Time, java.sql.Date, and java.sql.Timestamp:

Example 1: 25:00:00 becomes 01:00:00.

Example 2: 2000-01-32 becomes 2000-02-01.

Example 3: Both 1582-10-05 and 1582-10-15 become 1582-10-15 (the calendar switches from the Julian to the Gregorian calendar).

18.8.5 Overflow handling

This section explains when overflow is set when a program uses a set*XXX* method to set a value, or uses a get*XXX* method to get a value.

(1) setXXX methods (except for the setObject method)

The following tables show for each HiRDB data type whether overflow occurs when a `set`*XXX* method is used.

Table 18-73: Possibility of overflow when the setXXX method is used (1/2)

setXXX method	**HiRDB data type**					
	SMALLINT	**INTEGER**	**FLOAT**	**REAL**	**DECIMAL**	**Character string types**
`setByte`	--	--	--	--	Y	--
`setShort`	--	--	--	--	Y	--
`setInt`	Y	--	--	--	Y	--
`setLong`	Y	Y	--	--	Y	--
`setFloat`	Y	Y	--	--	Y	--
`setDouble`	Y	Y	--	--	Y	--
`setBigDecimal`	Y	Y	--	--	Y	--
`setBoolean`	--	--	--	--	Y	--
`setString`	Y	Y	--	--	Y	--
`setBytes`	N/A	N/A	N/A	N/A	N/A	N/A
`setDate`	N/A	N/A	N/A	N/A	N/A	--
`setTime`	N/A	N/A	N/A	N/A	N/A	--
`setTimestamp`	N/A	N/A	N/A	N/A	N/A	--
`setBlob`	N/A	N/A	N/A	N/A	N/A	N/A
`setBinaryStream`	N/A	N/A	N/A	N/A	N/A	N/A
`setAsciiStream`	N/A	N/A	N/A	N/A	N/A	--
`setArray`	Y	Y	Y	Y	Y	--
`setCharacterStream`	N/A	N/A	N/A	N/A	N/A	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 18-74: Possibility of overflow when the setXXX method is used (2/2)

setXXX method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`setByte`	N/A	N/A	N/A	N/A	N/A
`setShort`	N/A	N/A	N/A	N/A	N/A
`setInt`	N/A	N/A	N/A	N/A	N/A
`setLong`	N/A	N/A	N/A	N/A	N/A
`setFloat`	N/A	N/A	N/A	N/A	N/A
`setDouble`	N/A	N/A	N/A	N/A	N/A
`setBigDecimal`	N/A	N/A	N/A	N/A	N/A
`setBoolean`	N/A	N/A	N/A	N/A	N/A
`setString`	Y	--	Y	N/A	N/A
`setBytes`	N/A	N/A	N/A	--	--
`setDate`	Y	N/A	Y	N/A	N/A
`setTime`	N/A	Y	Y	N/A	N/A
`setTimestamp`	Y	N/A	Y	N/A	N/A
`setBlob`	N/A	N/A	N/A	--	--
`setBinaryStream`	N/A	N/A	N/A	--	--
`setAsciiStream`	N/A	N/A	N/A	--	--
`setArray`	Y	Y	Y	N/A	N/A
`setCharacterStream`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

#

Overflow occurs when the value obtained by the `getTime` method of the java.sql.Date, java.sql.Time, or java.sql.Timestamp class is an object larger than 253,402,268,399,999 or smaller than -62,135,802,000,000. The getTime method returns the number of milliseconds since 1970-01-01 00:00:00 (Greenwich Mean Time).

The methods shown below can be used to obtain 253,402,268,399,999 from the maximum value that can be stored in HiRDB's TIMESTAMP type, and -62,135,802,000,000 from the minimum value that can be represented by the java.sql.Timestamp class.

253,402,268,399,999:

```
Timestamp.valueOf("9999-12-31
23:59:59.999999").getTime()
```

-62,135,802,000,000:

```
Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()
```

(2) setObject method

The following tables show whether overflow occurs for each HiRDB data type when a setObject method is used.

Table 18-75: Possibility of overflow when the setObject method is used (1/2)

setObject method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
Byte	--	--	--	--	Y	--
Short	--	--	--	--	Y	--
Integer	Y	--	--	--	Y	--
Long	Y	Y	--	--	Y	--
Decimal	Y	Y	--	--	Y	--
Float	Y	Y	--	--	Y	--
Double	Y	Y	--	Y	Y	--
Boolean	--	--	--	--	Y	--
String	Y	Y	---	--	Y	--
Date	N/A	N/A	N/A	N/A	N/A	--
Time	N/A	N/A	N/A	N/A	N/A	--
Timestamp	N/A	N/A	N/A	N/A	N/A	--

setObject method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`byte[]`	N/A	N/A	N/A	N/A	N/A	--
`Blob`	N/A	N/A	N/A	N/A	N/A	N/A
`Array`	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 18-76: Possibility of overflow when the setObject method is used (2/2)

setObject method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`Byte`	N/A	N/A	N/A	N/A	N/A
`Short`	N/A	N/A	N/A	N/A	N/A
`Integer`	N/A	N/A	N/A	N/A	N/A
`Long`	N/A	N/A	N/A	N/A	N/A
`Decimal`	N/A	N/A	N/A	N/A	N/A
`Float`	N/A	N/A	N/A	N/A	N/A
`Double`	N/A	N/A	N/A	N/A	N/A
`Boolean`	N/A	N/A	N/A	N/A	N/A
`String`	Y	--	Y	N/A	N/A
`Date`	Y	N/A	Y	N/A	N/A
`Time`	N/A	Y	N/A	N/A	N/A
`Timestamp`	Y	N/A	Y	N/A	N/A
`byte[]`	N/A	N/A	N/A	--	--

setObject method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`Blob`	N/A	N/A	N/A	--	--
`Array`	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

#

Overflow occurs if the value obtained by the `getTime` method of the `java.sql.Date`, `java.sql.Time`, or `java.sql.Timestamp` class is an object larger than 253,402,268,399,999 or smaller than -62,135,802,000,000. The `getTime` method returns the number of milliseconds since 1970-01-01 00:00:00 (Greenwich Mean Time).

The methods shown below can be used to obtain 253,402,268,399,999 from the maximum value that can be stored in HiRDB's `TIMESTAMP` type, and -62,135,802,000,000 from the minimum value that can be represented by the `java.sql.Timestamp` class.

253,402,268,399,999:

```
Timestamp.valueOf("9999-12-31
23:59:59.999999").getTime()
```

-62,135,802,000,000:

```
Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()
```

(3) getXXX methods (except the getObject method)

The following tables show whether overflow occurs for each HiRDB data type when a `get`*XXX* method is used.

Table 18-77: Possibility of overflow when the getXXX method is used (1/2)

getXXX method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`getByte`	Y	Y	Y	Y	Y	Y
`getShort`	--	Y	Y	Y	Y	Y

getXXX method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
getInt	--	--	Y	Y	Y	Y
getLong	--	--	Y	Y	Y	Y
getFloat	--	--	--	--	--	--
getDouble	--	--	--	--	--	--
getBigDecimal	--	--	--	--	--	--
getBoolean	--	--	--	--	--	--
getString	--	--	--	--	--	--
getBytes	N/A	N/A	N/A	N/A	N/A	N/A
getDate	N/A	N/A	N/A	N/A	N/A	--
getTime	N/A	N/A	N/A	N/A	N/A	--
getTimestamp	N/A	N/A	N/A	N/A	N/A	--
getAsciiStream	N/A	N/A	N/A	N/A	N/A	--
getBinaryStream	N/A	N/A	N/A	N/A	N/A	N/A
getCharacterStream	N/A	N/A	N/A	N/A	N/A	--
getArray	N/A	N/A	N/A	N/A	N/A	--
getBlob	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: CHAR, MCHAR, NCHAR, VARCHAR, MVARCHAR, and NVARCHAR

Table 18-78: Possibility of overflow when the getXXX method is used (2/2)

getXXX method	HiRDB data type				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
`getByte`	N/A	N/A	N/A	N/A	N/A
`getShort`	N/A	N/A	N/A	N/A	N/A
`getInt`	N/A	N/A	N/A	N/A	N/A
`getLong`	N/A	N/A	N/A	N/A	N/A
`getFloat`	N/A	N/A	N/A	N/A	N/A
`getDouble`	N/A	N/A	N/A	N/A	N/A
`getBigDecimal`	N/A	N/A	N/A	N/A	N/A
`getBoolean`	N/A	N/A	N/A	N/A	N/A
`getString`	--	--	--	--	--
`getBytes`	N/A	N/A	N/A	--	--
`getDate`	--	N/A	--	N/A	N/A
`getTime`	N/A	--	--	N/A	N/A
`getTimestamp`	--	N/A	--	N/A	N/A
`getAsciiStream`	N/A	N/A	N/A	--	--
`getBinaryStream`	N/A	N/A	N/A	--	--
`getCharacterStream`	N/A	N/A	N/A	--	--
`getArray`	N/A	N/A	N/A	--	--
`getBlob`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

N/A: This combination is not allowed.

(4) getObject method

The following tables show whether overflow occurs for each HiRDB data type when a `getObject` method is used.

Table 18-79: Possibility of overflow when the getObject method is used (1/2)

getObject method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string type
`Byte`	Y	Y	Y	Y	Y	Y
`Short`	--	Y	Y	Y	Y	Y
`Int`	--	--	Y	Y	Y	Y
`Long`	--	--	Y	Y	Y	Y
`Float`	--	--	Y	--	Y	Y
`Double`	--	--	--	Y	Y	Y
`BigDecimal`	--	--	--	Y	Y	Y
`Boolean`	--	--	--	--	--	--
`String`	--	--	--	--	--	--
`Bytes`	N/A	N/A	N/A	N/A	N/A	N/A
`Date`	N/A	N/A	N/A	N/A	N/A	--
`Time`	N/A	N/A	N/A	N/A	N/A	--
`Timestamp`	N/A	N/A	N/A	N/A	N/A	--
`AsciiStream`	N/A	N/A	N/A	N/A	N/A	--
`BinaryStream`	N/A	N/A	N/A	N/A	N/A	N/A
`Object`	--	--	--	--	--	--
`CharacterStream`	N/A	N/A	N/A	N/A	N/A	--
`Array`	N/A	N/A	N/A	N/A	N/A	--
`Blob`	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 18-80: Possibility of overflow when the getObject method is used (2/2)

getObject method	HiRDB data type				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
`Byte`	N/A	N/A	N/A	N/A	N/A
`Short`	N/A	N/A	N/A	N/A	N/A
`Int`	N/A	N/A	N/A	N/A	N/A
`Long`	N/A	N/A	N/A	N/A	N/A
`Float`	N/A	N/A	N/A	N/A	N/A
`Double`	N/A	N/A	N/A	N/A	N/A
`BigDecimal`	N/A	N/A	N/A	N/A	N/A
`Boolean`	N/A	N/A	N/A	N/A	N/A
`String`	--	--	--	N/A	N/A
`Bytes`	N/A	N/A	N/A	--	--
`Date`	--	N/A	--	N/A	N/A
`Time`	N/A	--	--	N/A	N/A
`Timestamp`	--	N/A	--	N/A	N/A
`AsciiStream`	N/A	N/A	N/A	--	--
`BinaryStream`	N/A	N/A	N/A	--	--
`Object`	--	--	--	--	--
`CharacterStream`	N/A	N/A	N/A	--	--
`Array`	N/A	N/A	N/A	--	--
`Blob`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

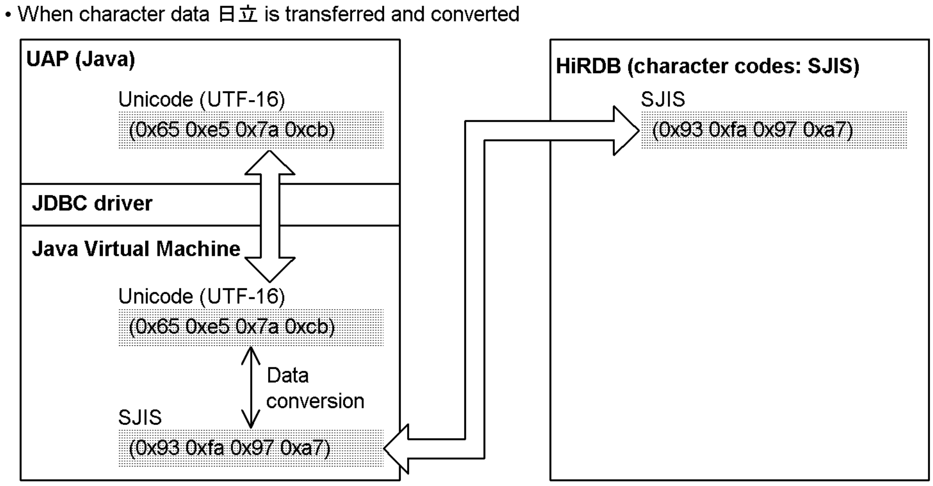
N/A: This combination is not allowed.

18.9 Character conversion facility

Because character codes in Java programs are handled as Unicode, the JDBC driver performs mutual character code conversion between HiRDB character data and Unicode. In this character code conversion process, the JDBC driver uses the encoder provided by the Java Virtual Machine.

The following figure shows the flow of mutual character code conversion between HiRDB character data and Unicode.

Figure 18-1: Flow of mutual character code conversion between HiRDB character data and Unicode



When the JDBC driver exchanges character data with HiRDB, it specifies the character set name to the encoder of the Java Virtual Machine. At this time, the JDBC driver gets the character encoding of the HiRDB server and specifies the character set name that corresponds to that encoding. If a character set name was specified by the ENCODELANG property or by the setEncodeLang method when the connection was established, the specified character set name is specified with priority to the encoder of the Java Virtual Machine. Therefore, if a character set name that does not correspond to the character encoding of the HiRDB server is specified in the ENCODELANG property or by the setEncodeLang method, an error occurs during character code conversion.

18.10 Supported client environment definitions

The table below lists the client environment definitions that can be specified with the JDBC driver. The numbers in the list correspond to the numbers of the individual environment variables in *6.6.4 Environment definition information*.

Table 18-81: Client environment variables that can be specified with the JDBC driver

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
1	`PDHOST`	HiRDB_for_Java_PDHOST	Specifies the host name of the HiRDB server to be connected.	System configuration
2	`PDNAMEPORT`	HiRDB_for_Java_PDNAMEPORT	Specifies the port number of the HiRDB server.	
3	`PDFESHOST`	HiRDB_for_Java_PDFESHOST	Specifies the host name of the front-end server.	
4	`PDSERVICEGRP`	HiRDB_for_Java_PDSERVICEGRP	Specifies the server name of the single server or front-end server.	
5	`PDSRVTYPE`	HiRDB_for_Java_PDSRVTYPE	Specifies the HiRDB server type.	
6	`PDSERVICEPORT`	HiRDB_for_Java_PDSERVICEPORT	Specifies the port number for high-speed connection.	
7	`PDFESGRP`	HiRDB_for_Java_PDFESGRP	Specifies a FES group to be connected when high-speed connection is used.	
8	`PDCLTRCVPORT`	HiRDB_for_Java_PDCLTRCVPORT	Specifies the client receive port number.	
9	`PDCLTRCVADDR`	HiRDB_for_Java_PDCLTRCVADDR	Specifies the IP address or host name of the client.	
20	`PDUSER`	HiRDB_for_Java_PDUSER	Specifies the authorization identifier and password. In UNIX, this environment variable can be omitted.	User execution environment
21	`PDCLTAPNAME`	HiRDB_for_Java_PDCLTAPNAME	Specifies UAP identification information (UAP identifier) of the UAP that accesses the HiRDB server.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
24	PDDBLOG	HiRDB_for_Java_PDDBLOG	Specifies whether or not the database update log is to be retrieved when the UAP is executed.	
25	PDEXWARN	HiRDB_for_Java_PDEXWARN	Specifies whether return codes with warnings are to be accepted from the server.	
26	PDSUBSTRLEN	HiRDB_for_Java_PDSUBSTRLEN	Specifies the maximum number of bytes representing one character.	
30	PDCLTGRP	HiRDB_for_Java_PDCLTGRP	Specifies a client group name when the connection frame guarantee facility for client groups is used.	
32	PDAUTORECONNECT	HiRDB_for_Java_PDAUTORECONNECT	Specifies whether or not the automatic reconnect facility is to be used.	
33	PDRCCOUNT	HiRDB_for_Java_PDRCCOUNT	Specifies the number of times the CONNECT statement is retried by the automatic reconnect facility.	
34	PDRCINTERVAL	HiRDB_for_Java_PDRCINTERVAL	Specifies the CONNECT retry interval at which the automatic reconnect facility executes reconnect processing.	
35	PDUAPENVFILE	HiRDB_for_Java_PDUAPENVFILE	Specifies the UAP environment definition file that defines the execution environment if the UAP is to be executed in a separate environment.	
36	PDDBBUFLRU	HiRDB_for_Java_PDDBBUFLRU	Specifies whether the LRU method is to be applied to processing when a page accessed by the UAP is cached to the global buffer.	
37	PDHATRNQUEUING	HiRDB_for_Java_PDHATRNQUEUING	Specifies that the client does not use the transaction queuing facility.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
38	PDCLTBINDLOOPBACKADDR	HiRDB_for_Java_PDCLTBINDLOOPBACKADDR	Specifies whether bind() is to be executed at a loopback address when a receive port to be used for communication with the HiRDB server is created.	
48	PDCWAITTIME	HiRDB_for_Java_PDCWAITTIME	Specifies the maximum time that the HiRDB client waits for a response from the HiRDB server after sending a request to the HiRDB server.	System monitoring
49	PDSWAITTIME	HiRDB_for_Java_PDSWAITTIME	Specifies the maximum time that the HiRDB server waits for the next request from the HiRDB client to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time during transaction processing.	
50	PDSWATCHTIME	HiRDB_for_Java_PDSWATCHTIME	Specifies the maximum time that the HiRDB server waits for the next request from the HiRDB server to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time other than the transaction processing time.	
51	PDCWAITTIMEWRNPNT	HiRDB_for_Java_PDCWAITTIMEWRNPNT	Specifies the output timing of the SQL runtime warning information file when the SQL runtime warning output facility is used. The output timing is specified as a percentage of the maximum time that the HiRDB client waits, or as an amount of time.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
55	PDNBLOCKWAITTIME	HiRDB_for_Java_PDNBLOCKWAITTIME	Specifies the connection establishment monitoring time in nonblock mode when monitoring is performed for completion of the connection between the HiRDB server and the client.	
56	PDCONNECTWAITTIME	HiRDB_for_Java_PDCONNECTWAITTIME	Specifies the maximum time that the HiRDB client waits for a response from the HiRDB server when it connects with the HiRDB server.	
57	PDCLTPATH	HiRDB_for_Java_PDCLTPATH	Specifies the storage directory for the SQL trace file and client error log file created by the HiRDB client.	Troubleshooting
58	PDSQLTRACE[#2]	HiRDB_for_Java_PDSQLTRACE	Specifies the size (bytes) of the SQL trace file into which SQL trace information for the UAP is to be output.	
61	PDPRMTRC	HiRDB_for_Java_PDPRMTRC	Specifies whether parameter information and retrieval data are to be output in the SQL trace information.	
62	PDPRMTRCSIZE	HiRDB_for_Java_PDPRMTRCSIZE	Specifies the maximum data length of the parameter information and retrieval data to be output in the SQL trace information.	
64	PDUAPREPLVL	HiRDB_for_Java_PDUAPREPLVL	Specifies output information for UAP statistical reports.	
65	PDREPPATH	HiRDB_for_Java_PDREPPATH	Specifies whether UAP statistical report files are to be output to a different directory from the directory specified by PDCLTPATH.	
66	PDTRCPATH	HiRDB_for_Java_PDTRCPATH	Specifies the storage directory for dynamic SQL trace files.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
68	PDSQLTEXTSIZE	HiRDB_for_Java_PDSQLTEXTSIZE	Specifies the size of the SQL statement to be output to the SQL trace.	
70	PDRCTRACE	HiRDB_for_Java_PDRCTRACE	Specifies the size of the output file for the UAP reconnect trace information.	
71	PDWRTLNPATH	HiRDB_for_Java_PDWRTLNPATH	Specifies the storage directory for files to which value expression values of WRITE LINE statements are to be output.	
72	PDWRTLNFILSZ	HiRDB_for_Java_PDWRTLNFILSZ	Specifies the maximum size of the files to which value expression values of WRITE LINE statements are to be output.	
73	PDWRTLNCOMSZ	HiRDB_for_Java_PDWRTLNCOMSZ	Specifies the total size of the value expression values in WRITE LINE statements.	
78	PDVWOPTMODE	HiRDB_for_Java_PDVWOPTMODE	Specifies whether the access path information file is to be retrieved.	Access path information file for the access path display utility
82	PDSTJTRNOUT	HiRDB_for_Java_PDSTJTRNOUT	Specifies whether UAP statistical information is to be output to a statistical log file for each transaction.	Output unit for UAP statistical information
83	PDLOCKLIMIT	HiRDB_for_Java_PDLOCKLIMIT	Specifies the maximum number of lock requests that a UAP can issue to one server.	Lock
84	PDDLKPRIO	HiRDB_for_Java_PDDLKPRIO	Specifies the deadlock priority value of the UAP.	
85	PDLOCKSKIP	HiRDB_for_Java_PDLOCKSKIP	Specifies whether an unlocked conditional search is to be performed.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
86	PDFORUPDATEEXLOCK	HiRDB_for_Java_PDFORUPDATEEXLOCK	Specifies whether WITH EXCLUSIVE LOCK is to be applied to the lock option of SQL statements in which the FOR UPDATE clause is specified (or assumed).	
87	PDISLLVL	HiRDB_for_Java_PDISLLVL	Specifies the data guarantee level of an SQL statement	SQL-related
88	PDSQLOPTLVL	HiRDB_for_Java_PDSQLOPTLVL	Specifies optimization methods (SQL optimization options) for determining the most efficient access path by taking the database status into consideration.	
89	PDADDITIONALOPTLVL	HiRDB_for_Java_PDADDITIONALOPTLVL	Specifies optimization methods (SQL extension optimizing methods) for determining the most efficient access path by taking the database status into consideration.	
90	PDHASHTBLSIZE	HiRDB_for_Java_PDHASHTBLSIZE	Specifies the hash table size when hash join, subquery hash execution is applied in SQL optimization.	
91	PDDFLNVAL	HiRDB_for_Java_PDDFLNVAL	Specifies whether a default value is to be set into an embedded variable when table data is to be fetched into the embedded variable and the fetched value is a null value.	
92	PDAGGR	HiRDB_for_Java_PDAGGR	Specifies the maximum number of groups allowed in each server so that the memory size used in GROUP BY processing can be determined.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
93	PDCMMTBFDDL	HiRDB_for_Java_PDCMMTBFDDL	When a definition SQL is to be executed in a transaction that is executing a data manipulation SQL, specifies whether the transaction is to be committed automatically before the definition SQL is executed.	
94	PDPRPCRCLS	HiRDB_for_Java_PDPRPCRCLS	Specifies whether an open cursor is to be closed automatically if a PREPARE statement reuses the SQL identifier that is using that open cursor.	
96	PDDDLDEAPRPEXE	HiRDB_for_Java_PDDDLDEAPRPEXE	Ignores the preceding transaction's preprocessing results and executes definition transactions.	
97	PDDDLDEAPRP	HiRDB_for_Java_PDDDLDEAPRP	Specifies whether definition information of a table being used by a closed holdable cursor can be changed by another UAP between transactions.	
98	PDLCKWAITTIME	HiRDB_for_Java_PDLCKWAITTIME	Specifies the maximum amount of time the HiRDB client is to monitor for release of a lock request, beginning when the lock request is placed in wait status.	
100	PDDELRSVWDFILE	HiRDB_for_Java_PDDELRSVWDFILE	Specifies the name of the SQL reserved word deletion file when the SQL reserved word deletion facility is used.	
101	PDHJHASHINGMODE	HiRDB_for_Java_PDHJHASHINGMODE	Specifies the hashing method when application of hash join, subquery hash execution is selected as the SQL extension optimizing option.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
102	PDCALCMDWAITTIME	HiRDB_for_Java_PDCALCMDWAITTIME	Specifies the maximum amount of time the HiRDB client is to wait for termination of a command executed by the CALL COMMAND statement or a utility, beginning when command execution starts.	
103	PDSTANDARDSQLSTATE	HiRDB_for_Java_PDSTANDARDSQLSTATE	Specifies whether the details of the SQLSTATE value are to be displayed.	
104	PDBLKF	HiRDB_for_Java_PDBLKF	Specifies the number of rows to be sent in one transfer when the HiRDB server transfers retrieval results to the HiRDB client.	Block transfer facility
105	PDBINARYBLKF	HiRDB_for_Java_PDBINARYBLKF	Specifies whether the block transfer facility is to be applied when a table with a BINARY-type selection expression with a defined length exceeding 32,00 bytes is searched.	
106	PDBLKBUFFSIZE	HiRDB_for_Java_PDBLKBUFFSIZE	Specifies the size of the server-client communication buffer used by the block transfer facility.	
113	PDDBACCS	HiRDB_for_Java_PDDBACCS	When the inner replica facility is being used and an RDAREA that is not the current RDAREA is to be accessed, specifies that RDAREA's generation number.	Inner replica facility
114	PDDBORGUAP	HiRDB_for_Java_PDDBORGUAP	Specifies whether to execute a UAP on the original RDAREA that is in online reorganization hold status.	Updatable online reorganization
115	PDSPACELVL	HiRDB_for_Java_PDSPACELVL	Specifies the space conversion level for data storage, comparison, and search processing.	Data space conversion

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
116	PDCLTRDNODE	HiRDB_for_Java_PDCLTRDNODE	Specifies the identifier of the XDM/RD E2 database to be connected when the XDM/RD E2 connection facility is used.	XDM/RD E2 connection facility
119	PDCNSTRNTNAME	HiRDB_for_Java_PDCNSTRNTNAME	Specifies the position of the constraint name definition when a referential or check constraint is defined.	Referential and check constraints
120	PDBESCONHOLD	HiRDB_for_Java_PDBESCONHOLD	Specifies whether the BES connection holding facility is to be used.	BES connection holding facility
121	PDBESCONHTI	HiRDB_for_Java_PDBESCONHTI	Specifies the BES connection holding period when the BES connection holding facility is used.	
129	PDPLGIXMK	HiRDB_for_Java_PDPLGIXMK	Specifies whether delayed batch creation of plug-in indexes is to be used.	Plug-ins
130	PDPLUGINNSUB	HiRDB_for_Java_PDPLUGINNSUB	For details, see the manual for the target plug-in.	
131	PDPLGPFSZ	HiRDB_for_Java_PDPLGPFSZ	Specifies the initial size of the index information file for delayed batch creation of plug-ins.	
132	PDPLGPFSZEXP	HiRDB_for_Java_PDPLGPFSZEXP	Specifies the extension size of the index information file for delayed batch creation of plug-ins.	
133	PDJDBFILEDIR	--	Specifies the log file output destination for Exception trace logs in the Type4 JDBC driver.	JDBC driver
134	PDJDBFILEOUTNUM	--	Specifies the number of Exception trace logs that the Type4 JDBC driver outputs to the log file.	
135	PDJDBONMEMNUM	--	Specifies the number of Exception trace logs acquired in memory by the Type4 JDBC driver.	

No.	Environment variable name	Corresponding system property[#1]	Function	Environment variable type
136	PDJDBTRACELEVEL	--	Specifies the trace acquisition level for Exception trace logs in the Type4 JDBC driver.	
137	PDXDSHOST	--	Specifies the host name of the XDS to be connected.	XDS client only
138	PDXDSPORT	--	Specifies the port number of the XDS to be connected.	
139	PDXVWOPT	--	Specifies whether the access path of the SQL statement used to access a table to be expanded to the memory database is to be output.	

Legend:

--: Not applicable

#1

You can use the system properties to specify the same connection information as in the client environment definitions. For details about the specification priorities, see *18.11 Connection information priorities*. For details about the system property settings for connection information related to trace logs, see *18.15.1(2) Setup for acquisition of the Exception trace log*.

Note that for an internal driver, system property specifications are ignored.

#2

The name of the SQL trace file is `pdjsql`*xxxxxxxx_ppppp*`_1.trc` or `pdjsql`*xxxxxxxx_ppppp*`_2.trc`.

xxxxxxxx: Name of connected server (up to eight characters)

ppppp: Receive port number (5 characters) at the client side

This format is used even when the SQL trace file is acquired by the UAP statistical report facility (`PDREPPATH` specification) or by the SQL trace dynamic acquisition facility (`PDTRCPATH` specification). However, if the SQL trace file is acquired before connection to the FES or SDS, the file name becomes `pdjsql1.trc` or `pdjsql2.trc`.

Using a HiRDB.INI file to enable client environment variables

Client environment variables defined in a `HiRDB.INI` file (that is stored in any desired directory) can be applied in any of the following cases:

- HiRDB_for_Java_HiRDB_INI is specified in the Properties argument of DriverManager.getConnection.
- HiRDB_INI is specified in the URL of DriverManager.getConnection.
- The absolute path of the HiRDB_INI file is specified in the setHiRDBINI method of a DataSource interface.

For details about the specification priorities, see *18.11 Connection information priorities*.

Note that for an internal driver, specifications in the HiRDB.INI file are ignored.

18.11 Connection information priorities

(1) List of connection information priorities

The JDBC driver enables you to specify synonymous connection information by using multiple setup methods (for example, DBHOST specified in the URL and PDHOST specified in HiRDB client environment definition). The following table lists the connection information items that have multiple setup methods; it also shows the priorities when items are set concurrently by multiple setup methods.

Table 18-82: Priorities for connection information

Meaning of connection information	Setup method	Priority		
		A	B	C
HiRDB host name	HiRDB_for_Java_PDHOST specified in system properties	1	1	1
	HiRDB_for_Java_DBHOST property in the Properties argument of DriverManager.getConnection	2	--	--
	DBHOST in the URL	3	--	--
	PDHOST in the HiRDB client environment definition specified in HiRDB_for_Java_ENV_VARIABLES in the Properties argument of DriverManager.getConnection	4	--	--
	PDHOST in the HiRDB environment variable group specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection	5	--	--
	PDHOST in the HiRDB environment variable group specified in DBID in the URL	6	--	--
	PDHOST in the HiRDB.ini file specified in HiRDB_for_Java_HiRDB_INI in the Properties argument of DriverManager.getConnection	7	--	--
	PDHOST in the HiRDB.ini file specified in HiRDB_INI in the URL	8	--	--
	setDBHostName method of DataSource interface	--	2	2
	PDHOST in the HiRDB client environment definition specified in the setEnvironmentVariables method of the DataSource interface	--	3	3
	PDHOST in the HiRDB environment variable group specified by setDescription method of the DataSource interface	--	4	--
	PDHOST in the HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	4
	PDHOST in a HiRDB.ini file specified in the setHiRDBINI method of the DataSource interface	--	5	5

Meaning of connection information	Setup method	Priority		
		A	B	C
HiRDB port number	HiRDB_for_Java_ PDNAMEPORT specified in system properties	1	1	1
	HiRDB_for_Java_DBID property in the Properties argument of DriverManager.getConnection	2	--	--
	DBID in the URL	3	--	--
	PDNAMEPORT in the HiRDB client environment definition specified in HiRDB_for_Java_ENV_VARIABLES in the Properties argument of DriverManager.getConnection	4	--	--
	PDNAMEPORT in the HiRDB environment variable group specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection	5	--	--
	PDNAMEPORT in the HiRDB environment variable group specified in DBID in the URL	6	--	--
	PDNAMEPORT in the HiRDB.ini file specified in HiRDB_for_Java_HiRDB_INI in the Properties argument of DriverManager.getConnection	7	--	--
	PDNAMEPORT in the HiRDB.ini file specified in HiRDB_INI in the URL	8	--	--
	setDescription method of the DataSource interface	--	2	2
	PDNAMEPORT in the HiRDB client environment definition specified in the setEnvironmentVariables method of a DataSource interface	--	3	3
	PDNAMEPORT in the HiRDB environment variable group specified by setDescription method of the DataSource interface	--	4	--
	PDNAMEPORT in the HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	4
	PDNAMEPORT in the HiRDB.ini file specified in the setHiRDBINI method of the DataSource interface	--	5	5
User name used to establish connection, password[#1]	HiRDB_for_Java_PDUSER specified in system properties	1	1	1

Meaning of connection information	Setup method	Priority		
		A	B	C
	user and password arguments in DriverManager.getConnection or user and password in the Properties argument	2	--	--
	USER and PASSWORD in the URL	3	--	--
	PDUSER in the HiRDB client environment variable specified in HiRDB_for_Java_ENV_VARIABLES in the Properties argument of DriverManager.getConnection	4	--	--
	PDUSER in the HiRDB environment variable group specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection	5	--	--
	PDUSER in the HiRDB environment variable group specified in DBID in the URL	6	--	--
	PDUSER in the HiRDB.ini file specified in HiRDB_for_Java_HiRDB_INI in the Properties argument of DriverManager.getConnection	7	--	--
	PDUSER in the HiRDB.ini file specified in HiRDB_INI in the URL	8	--	--
	Argument in the getConnection method of the DataSource interface or argument in the getPooledConnection method of the ConnectionPoolDataSource interface	--	2	--
	Argument in the getXAConnection method of XADataSource interface	--	--	2
	setUser and setPassword methods of the DataSource interface	--	3	3
	PDUSER in the HiRDB client environment variable specified in the setEnvironmentVariables method of the DataSource interface	--	4	4
	PDUSER in the HiRDB environment variable group specified in the setDescription method of the DataSource interface	--	5	--
	PDUSER in the HiRDB environment variable group specified in XADataSource.setXAOpenString	--	--	5
	PDUSER in the HiRDB.ini file specified in the setHiRDBINI method of a DataSource interface	--	6	6

Meaning of connection information	Setup method	Priority		
		A	B	C
UAP name#2	HiRDB_for_Java_ PDCLTAPNAME specified in system properties	1	1	1
	UAPNAME property in the Properties argument of the DriverManager.getConnectionoperties	2	--	--
	UAPNAME in the URL	3	--	--
	PDCLTAPNAME in the HiRDB client environment definition specified in HiRDB_for_Java_ENV_VARIABLES in the Properties argument of DriverManager.getConnection	4	--	--
	PDCLTAPNAME in the HiRDB environment variable specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection	5	--	--
	PDCLTAPNAME in the HiRDB environment variable group specified in DBID in the URL	6	--	--
	PDCLTAPNAME in the HiRDB.ini file specified in HiRDB_for_Java_HiRDB_IN in the Properties argument of DriverManager.getConnection	7	--	--
	PDCLTAPNAME in the HiRDB.ini file specified in HiRDB_INI in the URL	8	--	--
	setUapName method of DataSource interface	--	2	2
	PDCLTAPNAME in the HiRDB client environment definition specified in the setEnvironmentVariables method of the DataSource interface	--	3	3
	PDCLTAPNAME in the HiRDB environment variable group specified by the setDescription method of DataSource interface	--	4	--
	PDCLTAPNAME in the HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	4
	PDCLTAPNAME in the HiRDB.ini file specified in the setHiRDBINI method of the DataSource interface	--	5	5
Conversion character set	ENCODELANG property in the Properties argument of DriverManager.getConnection	1	--	--
	ENCODELANG in the URL	2	--	--
	setEncodeLang of the DataSource interface	--	1	1

Meaning of connection information	Setup method	Priority		
		A	B	C
Cursor operation mode	HIRDB_CURSOR property in the Properties argument of DriverManager.getConnection	1	--	--
	HIRDB_CURSOR in the URL	2	--	--
	setHiRDBCursorMode of the DataSource interface	--	1	1
Status after statement commit execution	HiRDB_for_Java_DAB_STATEMENT_COMMIT_BEHAVIOR specified in system properties	1	1	1
	HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR property in the Properties argument of DriverManager.getConnection	2	--	--
	STATEMENT_COMMIT_BEHAVIOR in the URL	3	--	--
	setStatementCommitBehavior of the DataSource interface	--	2	2
Login wait time	HiRDB_for_Java_PDCONNECTWAITTIME specified in system properties	1	1	1
	DriverManager.setLoginTimeout	2	--	--
	PDCONNECTWAITTIME in the HiRDB client environment definition specified in HiRDB_for_Java_HiRDB_IN in the Properties argument of DriverManager.getConnection	3	--	--
	PDCONNECTWAITTIME in the HiRDB environment variable group specified in DBID in the Properties argument of DriverManager.getConnection	4	--	--
	PDCONNECTWAITTIME in the HiRDB environment variable group specified by DBID in the URL	5	--	--
	PDCONNECTWAITTIME in the HiRDB.ini file specified in HiRDB_for_Java_HiRDB_INI in the Properties argument of DriverManager.getConnection	6	--	--
	PDCONNECTWAITTIME in the HiRDB.ini file specified in HiRDB_INI in the URL	7	--	--
	setLoginTimeout of the DataSource interface	--	2	2
	PDCONNECTWAITTIME in the HiRDB client environment definition specified in the setEnvironmentVariables method of the DataSource interface	--	3	3

Meaning of connection information	Setup method	Priority		
		A	B	C
	PDCONNECTWAITTIME in the HiRDB environment variable group specified by the setDescription method of the DataSource interface	--	4	--
	PDCONNECTWAITTIME in the HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	4
	PDCONNECTWAITTIME in the HiRDB.ini file specified in the setHiRDBINI method of the DataSource interface	--	5	5
Discarding of SQL preprocessing results during execution of the close method of Statement	HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR specified in system properties	1	1	1
	HiRDB_for_Java_STATEMENT_CLOSE_BEHAVIOR property in the Properties argument of DriverManager.getConnection	2	--	--
	STATEMENT_CLOSE_BEHAVIOR in the URL	3	--	--
	setStatementCloseBehavior of the DataSource interface	--	2	2

Legend:

A: For connection that uses DriverManager

B: For non-XA connection that uses the DataSource interface

C: For XA connection that uses the XADataSource interface

--: Cannot be specified for the connection method.

#1

If you specify a user name but omit a password, the JDBC driver assumes that no password is specified. If you omit a user name but specify a password, the specified password is ignored and the specification with the next highest priority takes effect.

#2

If this information cannot be set with the setting method shown in this table, the JDBC driver operates with the information set by the HiRDB_Type4_JDBC_Driver, which is the product name of the JDBC driver.

(2) Priorities for other client environment definitions

The following table shows the priorities for other HiRDB client environment definitions:

Priority	Client environment variable
1	System property
2	HiRDB client environment definition specified in HiRDB_for_Java_ENV_VARIABLES in the Properties argument of DriverManager.getConnection
3	Client environment variable in a HiRDB environment variable group file specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection
4	Client environment definition in a HiRDB environment variable group file specified in DBID in the URL in DriverManager.getConnection
5	HiRDB client environment variable specified in a HiRDB.ini file under the directory specified in HiRDB_for_Java_HiRDB_INI in the Properties argument of DriverManager.getConnection
6	HiRDB client environment variable specified in a HiRDB.ini file in the directory specified in HiRDB_INI in the URL in DriverManager.getConnection

Notes

- If a HiRDB environment variable group file is specified in HiRDB_for_Java_DBID in the Properties argument of DriverManager.getConnection and in DBID in the URL, the DBID specification in the URL is ignored.
- If HiRDB_for_Java_HiRDB_INI is specified in the Properties argument of DriverManager.getConnection and HiRDB_INI is specified in the URL, the HiRDB_IN specification in the URL is ignored.

18.12 Migration from a Type2 JDBC driver

This section discusses migration to a Type4 JDBC driver of Java stored procedures that have been used with a Type2 JDBC driver.

If you use any of the platforms listed below, you can migrate Java stored procedures from a Type2 JDBC driver to a Type4 JDBC driver without having to change the programs:

- 64-bit mode HP-UX(PA-RISC)
- 64-bit mode Solaris(PA-RISC)
- 64-bit mode AIX(PA-RISC)
- Linux(EM64T)
- Windows (x64)

If you use any other platform, you must change the settings as shown below when you migrate the internal driver from Type2 JDBC to Type4 JDBC.

Item requiring change	Type2 JDBC driver	Type4 JDBC driver
Driver name	"JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver"	"JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver"
Protocol name, subprotocol name, and subname specified in the URL when connection is established with HiRDB	jdbc:hitachi:PrdbDrive	jdbc:hitachi:hirdb
Class name of `DataSource` class	JdbhDataSource	PrdbDataSource

Item requiring change	Type2 JDBC driver	Type4 JDBC driver
Cursor operation mode[#]	Specify by using one of the following methods: • COMMIT_BEHAVIOR in the URL that is specified when connection is established • COMMIT_BEHAVIOR property that is specified when connection is established • setCommit_Behavior method of the DataSource class	Specify by using one of the following methods: • Combination of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR in the URL that is specified when connection is established • Combination of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR properties that are specified when connection is established • Combination of setStatementCommitBehavior and setHiRDBCursorMode methods of the DataSource class
HiRDB's array update, array insertion, and array deletion facilities	Supported if this function is set to be used by one of the following methods: • BLOCK_UPDATE property that is specified when connection is established • HiRDB_for_Java_BLOCK_UPDATE system property • setBlockUpdate method of the DataSource class • setBlockUpdate method of the JdbcDbpsvPreparedStatement class	Supported unconditionally. However, the setBlockUpdate method of the JdbcDbpsvPreparedStatement class is not supported.
Maximum number of input or input-output ? parameters in the SQL statements to be executed	Specify by using one of the following methods: • HiRDB_for_Java_SQL_IN_NUM property that is specified when connection is established • HiRDB_for_Java_SQL_IN_NUM system property • setSQLInNum method of the DataSource class The default value is 64.	Specify by using one of the following methods: • HiRDB_for_Java_SQL_IN_NUM property that is specified when connection is established • setSQLInNum method of the DataSource class The default value is 300.
Maximum number of output items for the SQL statement to be executed	Specify by using one of the following methods: • HiRDB_for_Java_SQL_OUT_NUM property that is specified when connection is established • HiRDB_for_Java_SQL_OUT_NUM system property • setSQLOutNum method of the DataSource class The default value is 64.	Specify by using one of the following methods: • HiRDB_for_Java_SQL_OUT_NUM property that is specified when connection is established • setSQLOutNum method of the DataSource class The default value is 300.

#

Whether ResultSet and Statement objects become valid after commit execution depends on the cursor's operation mode.

The table below shows the correspondence of the cursor operation mode settings between Type2 JDBC and Type4 JDBC drivers.

Note that the default values are different between the Type2 JDBC and Type4 JDBC drivers.

ResultSet object status after commit execution	Statement object status after commit execution	Type2 JDBC driver	Type4 JDBC driver

Invalid	Invalid	COMMIT_BEHAVIOR= "DELETE" (default value)	HIRDB_CURSOR=FALSE (default value)
			STATEMENT_COMMIT_BEHAVIOR=FALSE

	Valid	COMMIT_BEHAVIOR= "CLOSE"	HIRDB_CURSOR=FALSE (default value)
			STATEMENT_COMMIT_BEHAVIOR=TRUE (default value)

Valid	Valid	COMMIT_BEHAVIOR= "PRESERVE"	HIRDB_CURSOR=TRUE
			STATEMENT_COMMIT_BEHAVIOR=TRUE or STATEMENT_COMMIT_BEHAVIOR=FALSE

Client environment definitions in Java applications

With a Type2 JDBC driver, the PDHOST and PDNAMEPORT specifications in the OS's environment variables are effective if you omit information (such as URL) when connection is established.

A Type4 JDBC driver does not use OS environment variables, so when you migrate from a Type2 JDBC driver you must modify your UAPs so that PDHOST and PDNAMEPORT in the client environment definitions are used for URL when connection is established.

18.13 Migration from DABroker for Java

If you migrate your JDBC driver from DABroker for Java to a Type4 JDBC, you might need to change UAPs. A facility called the DABroker for Java-compatible facility is provided for running a Type4 JDBC driver in a mode compatible with DABroker for Java by specifying system properties. This section describes the DABroker for Java-compatible facility.

18.13.1 System properties related to the DABroker for Java-compatible facility

You use system properties to specify connection information related to the DABroker for Java-compatible facility. The table below lists and describes the system properties to be specified.

Note that for an internal driver, system property specifications are ignored.

Table 18-83: System properties related to the DABroker for Java-compatible facility

Property name	Description	Specification

HiRDB_for_Java_DAB_CONVERT_NULL	Specifies whether the `setString` and `getString` methods of the `CallableStatement` class are to be run in the DABroker for Java-compatible mode. If this property is omitted, `FALSE` is assumed. The specified value is not case sensitive.
	`TRUE`
	Runs the methods in the DABroker for Java-compatible mode.
	If the data type of a `?` parameter in the `setString` method is `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`, and a character string with a length of 0 is specified in the argument, `null` is set in the `?` parameter.
	If the `?` parameter value acquired by the `getString` method is a character string with a length of 0, `null` is set as the return value.
	`FALSE`
	Does not run the methods in the DABroker for Java-compatible mode.
	If the data type of a `?` parameter in the `setString` method is `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`, and a character string with a length of 0 is specified in the argument, a character string with a length of 0 is set in the `?` parameter.
	If the `?` parameter value acquired by the `getString` method is a character string with a length of 0, a character string with a length of 0 is set as the return value.
	Other
	The driver assumes that `FALSE` is specified.
		Optional

Property name	Description	Specification

HiRDB_for_Java_DAB_STATEMENT_COMMIT_BEHAVIOR	Specifies whether the processing for disabling the objects of `Statement` during commit execution is to be run in the DABroker for Java-compatible mode. If this property is omitted, `FALSE` is assumed. The specified value is not case sensitive.
	`TRUE`
	Makes the processing compatible with DABroker for Java.
	If commit processing is performed, the objects of `Statement` are disabled.
	`FALSE`
	Does not make the processing compatible with DABroker for Java.
	The driver uses a value specified by another method. For details about the specification method, see *18.11(1) List of connection information priorities*.
	Other
	The driver assumes that `FALSE` is specified.
		Optional

HiRDB_for_Java_DAB_EXECUTESQL_NOCHK	Specifies whether the `executeQuery` and `executeUpdate` methods of `Statement` are to be run in the DABroker for Java-compatible mode. If this property is omitted, `FALSE` is assumed. The specified value is not case sensitive.
	`TRUE`
	Runs the methods in the DABroker for Java-compatible mode.
	• The `executeQuery` method of `Statement` can be used to execute non-retrieval SQL statements.
	• The `executeUpdate` method of `Statement` can be used to execute retrieval SQL statements.
	• If a non-retrieval SQL statement is executed by the `execute`, `executeQuery`, or `executeUpdate` method of `Statement` and then the `getResultSet` method is executed, the driver returns a `ResultSet` containing no columns.
	`FALSE`
	Does not run the methods in the DABroker for Java-compatible mode.
	• If the `executeQuery` method of `Statement` is used to execute a non-retrieval SQL statement, the driver throws an `SQLException`.
	• If the `executeUpdate` method of `Statement` is used to execute a retrieval SQL statement, the driver throws an `SQLException`.
	• If a non-retrieval SQL statement is executed by the `execute` or `executeUpdate` method of `Statement` and then the `getResultSet` method is executed, the driver returns `null`.
	Other
	The driver assumes that `FALSE` is specified.
		Optional

Property name	Description	Specification

HiRDB_for_Java_DAB_OUTPARMSIZE_MAX	Specifies whether the `registerOutParameter` method of the `CallableStatement` class is to be run in the DABroker for Java-compatible mode. If this property is omitted, `FALSE` is assumed. The specified value is not case sensitive.
	`TRUE`
	Runs the method in the DABroker for Java-compatible mode.
	If the `VARCHAR`, `NVARCHAR`, or `MVARCHAR` data type has been acquired for `INOUT` or `OUT` parameters by `CALL` statement preprocessing and this data type is registered as `java.sql.Types.CHAR` by the `registerOutParameter` method of the `CallableStatement` class, the `getString` method of the `CallableStatement` class returns the maximum size of data acquired by preprocessing.
	If the data set by the stored procedure is smaller than the maximum size of data acquired by preprocessing, spaces are added to the data set by the stored procedure to match the maximum size acquired by preprocessing.
	`FALSE`
	Does not run the method in the DABroker for Java-compatible mode.
	If the `VARCHAR`, `NVARCHAR`, or `MVARCHAR` data type has been acquired for `INOUT` or `OUT` parameters by `CALL` statement preprocessing and this data type is registered as `java.sql.Types.CHAR` by the `registerOutParameter` method of the `CallableStatement` class, the `getString` method of the `CallableStatement` class returns the data set by the stored procedure as is. No spaces are added to the data set by the stored procedure.
	Other
	The driver assumes that `FALSE` is specified.
		Optional

18.13.2 Items that are not compatible with a Type4 JDBC driver

The tables in this subsection list the items that are not compatible with a Type4 JDBC driver for the various versions of DABroker for Java and Cosminexus.

DABroker for Java and Cosminexus versions:

- DABroker for Java Version 2 02-10 or earlier
- Cosminexus Studio Version 5 05-05-/E or earlier
- Cosminexus Application Server Version 5 05-05-/E or earlier
- Cosminexus Developer Version 5 05-05-/E or earlier

Table 18-84: Items that are not compatible with a Type4 JDBC driver (part 1)

Incompatible item		DABroker for Java	Type4 JDBC driver
Handling of character string with a length of 0 by the `setString` and `getString` methods of the `CallableStatement` class	`setString` method	If the data type of a `?` parameter is `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, or `MVARCHAR` and a character string with a value of 0 is set in the argument, `null` is set in the `?` parameter.	If the data type of a `?` parameter is `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, or `MVARCHAR` and a character string with a value of 0 is set in the argument, a character string with a length of 0 is set in the `?` parameter.
	`getString` method	If the acquired `?` parameter value is a character string with a length of 0, `null` is set as the return value.	If the acquired `?` parameter value is a character string with a length of 0, a character string with a length of 0 is set as the return value.
Status of `Statement` after commit execution		If commit processing is performed, the objects of `Statement` are disabled.	If commit processing is performed, the objects of `Statement` are enabled.

DABroker for Java and Cosminexus versions:

- DABroker for Java Version 2 02-07 or earlier
- Cosminexus Studio Version 5 05-05 or earlier
- Cosminexus Application Server Version 5 05-05 or earlier
- Cosminexus Developer Version 5 05-05 or earlier

Table 18-85: Items that are not compatible with a Type4 JDBC driver (part 2)

Incompatible item		DABroker for Java	Type4 JDBC driver
• Types of SQL statements that can be executed by the `executeQuery` and `executeUpdate` methods of `Statement` • Return value of the `getResultSet` method after execution of a non-retrieval SQL statement	`executeQuery` method	• All SQL statements can be executed. • If a non-retrieval SQL statement is executed, a `ResultSet` containing no columns is returned.	Only retrieval SQL statements can be executed.
	`executeUpdate` method	• All SQL statements can be executed. • If a retrieval SQL statement is executed, `-1` is returned.	Only non-retrieval SQL statements can be executed.
	`getResultSet` method	If the `getResultSet` method is executed after a non-retrieval SQL statement was executed by the `execute`, `executeQuery`, or `executeUpdate` method of `Statement`, a `ResultSet` containing no columns is returned.	If the `getResultSet` method is executed after a non-retrieval SQL statement was executed by the `execute` or `executeUpdate` method of `Statement`, `null` is returned.

DABroker for Java and Cosminexus versions:

- Cosminexus DABroker 03-00-/D or earlier
- Cosminexus DABroker for Java 02-06-/B or earlier
- Cosminexus Studio Version 5 05-00-/D or earlier
- Cosminexus Application Server Version 5 05-00-/D or earlier
- Cosminexus Developer Version 5 05-00-/D or earlier

Table 18-86: Items that are not compatible with a Type4 JDBC driver (part 3)

Incompatible item	DABroker for Java	Type4 JDBC driver
Data received by the `getString` method of the `CallableStatement` class when the `VARCHAR`, `NVARCHAR`, or `MVARCHAR` data type has been acquired for `INOUT` or `OUT` parameters by `CALL` statement preprocessing and this data type is registered as `java.sql.Types.CHAR` by the `registerOutParameter` method of the `CallableStatement` class	The maximum size of data acquired by preprocessing is returned. If the data set by the stored procedure is smaller than the maximum size of data acquired by preprocessing, spaces are added to the data set by the stored procedure to match the maximum size acquired by preprocessing.	The data set by the stored procedure is returned. No spaces are added to the data set by the stored procedure.

18.14 JDBC interface method trace

You can acquire a JDBC interface method trace as troubleshooting information when you call a method of the JDBC interface.

18.14.1 Setup for trace acquisition

(1) Connection with the DriverManager class

Specify a valid log writer by using the `setLogWrite` method of the `DriverManager` class, and specify acquisition of the JDBC interface method trace in the arguments (`Properties info`) of the `getConnection` method.

For details, see *18.2.2(2)(d) JDBC_IF* and *18.2.2(2)(e) TRC_NO*.

(2) Connection with the DataSource class

Specify a valid log writer by using the `setLogWriter` method provided by the `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` interfaces, and specify the `setJDBC_IF_TRC` method provided by the `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` classes, which are provided by the JDBC2.0 Optional Package.

For details, see *18.7.5 setJDBC_IF_TRC* and *18.7.7 setTRC_NO*.

18.14.2 Acquisition rules

This section describes the rules for acquisition of the JDBC interface method trace.

- Trace information is acquired when a method of the JDBC interface is called and when processing is returned from that method.

  However, trace information is not acquired for methods executed before connection to the database.

  Trace information is not acquired for the following methods:

  Driver interface

  - `acceptsURL(String url)`
  - `getMajorVersion()`
  - `getMinorVersion()`
  - `getPropertyInfo(String url, Properties info)`
  - `jdbcCompliant()`

  DataSource interface

  - `getLoginTimeout()`

- getLogWriter()
- setLoginTimeout(int seconds)
- setLogWriter(PrintWriter out)

- Trace information is stored for the number of entries and is output to the specified log writer when the Connection.close method is called (normal termination), or when an SQLException, XAException, or BatchUpdateException is thrown (error occurrence).
- If the number of trace information items exceeds the number of entries, the stored trace information is discarded in chronological sequence and the newest trace information is retained.
- A JDBC interface method trace uses a single-entry trace area for each Entry and each Return.

18.14.3 Output example

Shown below is an output example of a JDBC interface method trace.

Output example

```
[1]                          [2]                        [3]
[HiRDB_Type4_JDBC_Driver][JDBC Interface Entry ][PrdbStatement.executeQuery]
                                                  [4]
[HiRDB_Type4_JDBC_Driver]                         sql=select * from pp
[HiRDB_Type4_JDBC_Driver][JDBC Interface Return][PrdbStatement.executeQuery]
                                                  [5]
[HiRDB_Type4_JDBC_Driver]
Return=JP.co.Hitachi.soft.HiRDB.JDBC.Prdb...
[HiRDB_Type4_JDBC_Driver][JDBC Interface Entry ][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC_Driver][JDBC Interface Return][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC_Driver]
Return=JP.co.Hitachi.soft.HiRDB.JDBC.Prdb...
```

Explanation

1. [HiRDB_Type4_JDBC_Driver]

   Name of the JDBC driver

2. [JDBC Interface Entry], [JDBC Interface Return]

   [JDBC Interface Entry]: Calling of the JDBC method

   [JDBC Interface Return]: Return from the JDBC method

3. [*XXXXX*.*YYYYY*]

   YYYYY method of the *XXXXX* class

4. `select * from pp`

   Argument of the JDBC method (for the argument indicating the password, an asterisk (`*`) is output, as in `password=*`)

5. `JP.co.Hitachi.soft.HiRDB.JDBC.Prdb`

   Return value of the JDBC method

18.15 Exception trace log

You can acquire an Exception trace log as troubleshooting information. If a failure caused by an exception occurs in the JDBC driver, the failure cause is output to the Exception trace log.

The following constitute the output contents:

- Information (such as error messages) when an exception occurs
- Execution record of JDBC's API methods up to the point where an exception occurred

When this function is used, information about JDBC's API methods that are called from the UAP is stored in the JDBC driver memory. Then if an `SQLException`, `BatchUpdateException`, or `XAException` occurs, the information stored in memory can be output to a file before the exception is thrown.

18.15.1 Methods to be acquired and setup for log acquisition

(1) Methods to be acquired in the Exception trace log

The information to be acquired in the Exception trace log is the calling and return of methods described in the java.sql package found in the API specifications of Java 2 Platform, Standard Edition, Version 1.4.

Methods that satisfy all of the following conditions are acquired:

- The methods listed in the table below, when a trace acquisition level required for acquisition is specified for each method.
- Methods of the `Blob` and `InputStream` classes when `LOCATOR` is specified in `LONGVARBINARY_ACCESS`.

Methods that only look up and return information found in objects or only store information into objects, such as the `ResultSet.get`*XXX*, `PreparedStatement.set`*XXX*, and `Connection.isClosed` methods, are not acquisition targets.

The table below lists the methods that are acquisition targets of the Exception trace log. The table also provides the trace acquisition levels of the methods.

Table 18-87: Methods that are acquisition targets of the Exception trace log and their trace acquisition levels

Class	Method	Trace acquisition level				
		1	2	3	4	5[#1]
CallableStatement	void close()[#6]	Y	Y	Y	Y	Y
	boolean execute()[#2 #6]	--	Y	Y	Y	Y
	ResultSet executeQuery()[#2, #6]	--	Y	Y	Y	Y
	int executeUpdate()[#2, #6]	--	Y	Y	Y	Y
	boolean execute(String sql)[#3, #6]	Y	Y	Y	Y	Y
	int[] executeBatch()[#6]	--	Y	Y	Y	Y
	ResultSet executeQuery(String sql)[#3, #6]	Y	Y	Y	Y	Y
	int executeUpdate(String sql)[#3, #6]	Y	Y	Y	Y	Y
Connection	void close()	Y	Y	Y	Y	Y
	void commit()	--	Y	Y	Y	Y
	Statement createStatement()[#2]	Y	Y	Y	Y	Y
	Statement createStatement(int resultSetType, int resultSetConcurrency)[#3]	Y	Y	Y	Y	Y
	DatabaseMetaData getMetaData()	--	Y	Y	Y	Y
	CallableStatement prepareCall(String sql)[#2]	Y	Y	Y	Y	Y
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)[#3]	Y	Y	Y	Y	Y
	PreparedStatement prepareStatement(String sql)[#2]	Y	Y	Y	Y	Y
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)[#3]	Y	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
	void rollback()#2	--	Y	Y	Y	Y
	void setAutoCommit(boolean autoCommit)	--	Y	Y	Y	Y
DatabaseMetaData	ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	--	Y	Y	Y	Y
	ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	--	Y	Y	Y	Y
	ResultSet getCatalogs()	--	Y	Y	Y	Y
	ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	--	Y	Y	Y	Y
	ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	--	Y	Y	Y	Y
	Connection getConnection()	--	Y	Y	Y	Y
	ResultSet getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema)	--	Y	Y	Y	Y
	ResultSet getExportedKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getImportedKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	--	Y	Y	Y	Y
	ResultSet getPrimaryKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	--	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5[#1]
	ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	--	Y	Y	Y	Y
	ResultSet getSchemas()	--	Y	Y	Y	Y
	ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	--	Y	Y	Y	Y
	ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	--	Y	Y	Y	Y
	ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	--	Y	Y	Y	Y
	ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	--	Y	Y	Y	Y
	ResultSet getTableTypes()	--	Y	Y	Y	Y
	ResultSet getTypeInfo()	--	Y	Y	Y	Y
	ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	--	Y	Y	Y	Y
	ResultSet getVersionColumns(String catalog, String schema, String table)	--	Y	Y	Y	Y
Driver	Connection connect(String url, Properties info)	Y	Y	Y	Y	Y
PreparedStatement	boolean execute()[#2]	--	Y	Y	Y	Y
	ResultSet executeQuery()[#2]	--	Y	Y	Y	Y
	int executeUpdate()[#2]	--	Y	Y	Y	Y
	ResultSetMetaData getMetaData()	--	Y	Y	Y	Y
	boolean execute(String sql)[#3, #5]	Y	--	Y	Y	Y
	int[] executeBatch()[#5]	--	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5[#1]
	ResultSet executeQuery(String sql)[#3, #5]	Y	Y	Y	Y	Y
	int executeUpdate(String sql)[#3, #5]	Y	Y	Y	Y	Y
ResultSet	boolean absolute(int row)	--	Y	Y	Y	Y
	void afterLast()	--	Y	Y	Y	Y
	void beforeFirst()	--	Y	Y	Y	Y
	void close()	--	Y	Y	Y	Y
	boolean first()	--	Y	Y	Y	Y
	ResultSetMetaData getMetaData()	--	Y	Y	Y	Y
	Statement getStatement()	--	Y	Y	Y	Y
	boolean last()	--	Y	Y	Y	Y
	boolean next()	--	Y	Y	Y	Y
	boolean relative(int rows)	--	Y	Y	Y	Y
	boolean isAfterLast()	--	Y	Y	Y	Y
	boolean isBeforeFirst()	--	Y	Y	Y	Y
	boolean isLast()	--	Y	Y	Y	Y
Statement	void cancel()	--	Y	Y	Y	Y
	void close()	Y	Y	Y	Y	Y
	boolean execute(String sql)	Y	Y	Y	Y	Y
	int[] executeBatch()	--	Y	Y	Y	Y
	ResultSet executeQuery(String sql)	Y	Y	Y	Y	Y
	int executeUpdate(String sql)	Y	Y	Y	Y	Y
	ResultSet getResultSet()	--	Y	Y	Y	Y
Blob	long position(Blob pattern,long start)[#2]	--	Y	Y	Y	Y
	long position(byte[] pattern, long start)[#3]	--	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5[#1]
	long length()	--	Y	Y	Y	Y
	byte[] getBytes(long pos, int length)	--	Y	Y	Y	Y
InputStream	int read()[#2]	--	Y	Y	Y	Y
	int read(byte[] data, int data_offset,int data_len)[#3]	--	Y	Y	Y	Y
	int read(byte[] data, int data_offset,int data_len)[#4]	--	Y	Y	Y	Y
DataSource	getConnection()[#2]	Y	Y	Y	Y	Y
	getConnection(String username, String password)[#3]	Y	Y	Y	Y	Y
ConnectionPoolDataSource	getPooledConnection()[#2]	Y	Y	Y	Y	Y
	getPooledConnection(String username, String password)[#3]	Y	Y	Y	Y	Y
PooledConnection	close()	Y	Y	Y	Y	Y
	getConnection()	Y	Y	Y	Y	Y
XADataSource	getXAConnection()[#2]	Y	Y	Y	Y	Y
	getXAConnection(String username, String password)[#3]	Y	Y	Y	Y	Y
XAConnection	getXAResource()	Y	Y	Y	Y	Y
XAResource	commit(Xid xid, boolean onePhase)	--	--	Y	Y	Y
	end(Xid xid, int flags)	--	--	Y	Y	Y
	forget(Xid xid)	--	--	Y	Y	Y
	isSameRM(XAResource xares)	--	--	Y	Y	Y
	prepare(Xid xid)	--	--	Y	Y	Y
	recover(int flag)	--	--	Y	Y	Y
	rollback(Xid xid)	--	--	Y	Y	Y
	start(Xid xid, int flags)	--	--	Y	Y	Y

Legend:

Y: An Exception trace log is acquired.

--: An Exception trace log is not acquired.

#1

If the trace acquisition level is 5, an Exception trace log that includes internal calling is acquired.

#2

method-name(1) is output as the method name.

#3

method-name(2) is output as the method name.

#4

method-name(3) is output as the method name.

#5

This method overrides the method of the `Statement` class.

#6

This method overrides the methods of the `Statement` and `PreparedStatement` classes.

(2) Setup for acquisition of the Exception trace log

Use the system properties or the client environment definitions to set the file output destination of the Exception trace log, the number of outputs to the file, the number of information items to be acquired in memory, and the trace acquisition level. The priorities are as follows:

1. System properties
2. Client environment definitions

(a) Setting client environment definitions

Specify the following items in the client environment definitions:

- `PDJDBFILEDIR`
- `PDJDBFILEOUTNUM`
- `PDJDBONMEMNUM`
- `PDJDBTRACELEVEL`

For details about the specification values, see *6.6.4 Environment definition information*.

If invalid values are specified in these client environment definitions, the facility for controlling the Exception trace log acquired when an SQLException is thrown assumes that values were not specified for these client environment definitions. In this situation, the defaults shown in the table below are assumed.

(b) Setting system properties

Specify in the system properties the items shown in the following table.

Table 18-88: System property settings for acquisition of the Exception trace log

Item	System property	Description	Default#
File output destination	HiRDB_for_Java_FileDIR	Specify the absolute path of the directory to which the Exception trace log is to be output. The Exception trace log is output immediately under the specified directory.	Current directory
Number of outputs to the file	HiRDB_for_Java_FileOutNUM	Specify the maximum number of information items to be output to one file. Specify a value in the range from 1 to 50. The maximum number of information items to be output to one file is actually *number of outputs to the file* x *number of acquisition items to be acquired in memory*. For the number of outputs to the file, the formats from Format 2 to Format 4 shown in *18.15.2 Output formats* are each counted as one output. The information items are output to memory in the sequence they were stored. If information items exceeding the maximum value are to be output to a file, the items are wrapped around into two files. The file names are as follows: • pdexc1.trc • pdexc2.trc However, the output destination file does not change between Format 1 and Format 2 shown in *18.15.2 Output formats*.	5
Number of information items to be acquired in memory	HiRDB_for_Java_OnMemNUM	Specify the maximum number of information items to be stored in memory. You can specify a value in the range from 500 to 10,000. For the information acquired in memory, each method shown in *Table 18-87* is counted as one item. If the number of information items to be stored exceeds the maximum value, old information items are overwritten with new information items in chronological sequence.	1,000

Item	System property	Description	Default#
Trace acquisition level	`HiRDB_for_Java_TraceLevel`	Specify a trace acquisition level. You can specify a level in the range from 0 to 5. If you specify `5`, all methods that are trace acquisition targets, including internally called methods, are acquired. If you specify `0`, an Exception trace log is not acquired.	1

#

When the Exception trace log is acquired in the following cases, the JDBC driver assumes that values were not specified in the system properties (and the defaults are assumed):

- When an invalid value is specified in the system properties and an `SQLException` is thrown during connection to the database
- When the Java Virtual Machine denies the JDBC driver permission to exchange system properties because of security manager reasons
- Before initial connection of the Java Virtual Machine is established

18.15.2 Output formats

The Exception trace log has the following four formats.

Format 1: Header section

```
[AA....AA] HiRDB_Type4_JDBC_Driver BB-CC
```

Format 2: Method execution history (execution start of a method)

```
AAAAAAAAAAAAAAAAAAAAAA  BB....BB:[C][DD....DD]
                      ConnectionID(EE....EE) : SID(FF....FF)
                      GG....GG
```

Format 3: Method execution history (normal termination of a method)

```
AAAAAAAAAAAAAAAAAAAAAA  BB....BB:[C][DD....DD]
                      ConnectionID(EE....EE) : SID(FF....FF)
                      HH....HH
```

Format 4: Timing when output occurred

```
AAAAAAAAAAAAAAAAAAAAAAA BB....BB:Exception:
II....II
```

Format 2 and Format 3 are output repeatedly in time series sequence for each method executed.

(1) Explanation of variables in Format 1

AA....AA

Indicates the sequence number of the output information.

The sequence number is incremented by 1 for each output (including failures caused by output errors). After the value reaches 2,147,483,647, the sequence returns to 0.

BB

Indicates the version of the JDBC driver.

CC

Indicates the revision of the JDBC driver.

(2) Explanation of variables in Format 2, Format 3, and Format 4

AAAAAAAAAAAAAAAAAAAAAAA

Indicates the acquisition date and time of the Exception trace log, in the following format (a value from 0 to 9 is set in each variable):

```
YYYY/MM/DD hh:mm:ss.sss
```

YYYY: Year (Western calendar)

MM: Month

DD: Day

hh: Hour (24-hour clock format)

mm: Minute

ss.sss: Second (includes 3 digits after the decimal point)

BB....BB

Indicates thread identification information for the target thread, in the following format:

```
Thread[aa....aa]@bb....bb
```

aa....aa: Thread information, including the thread name, priority sequence, and thread group name. The Java Virtual Machine determines the format.

bb....bb: Hash code of the object. The Java Virtual Machine determines the format.

C

Indicates call identification information for the method:

`E`: Indicates that the information is history information for when the method was started.

`R`: Indicates that the information is history information for when the method terminated normally.

DD....DD

Indicates the object identifier and the method name, in the following format:

```
aa....aa.bb....bb
```

aa....aa: Object identifier (up to 32 characters)

The Java Virtual Machine determines the format.

bb....bb: Method name

EE....EE

Indicates the connection ID, in the following format:

```
aa....aa:bb....bb:cc....cc
```

aa....aa: Front-end server name or single-server name (up to 32 characters).

If this information cannot be retrieved, an asterisk (`*`) is output.

bb....bb: Connection sequence number (up to 10 characters) of the server identified by *aa....aa*.

If this information cannot be retrieved, an asterisk (`*`) is output.

cc....cc: Process ID (up to 10 characters) of the server identified by *aa....aa*.

If this information cannot be retrieved, an asterisk (`*`) is output.

FF....FF

Indicates the section ID (up to 4 characters).

GG....GG

Indicates the method arguments, in the following format (this information is not output for methods without arguments):

```
aa....aa=bb....bb
aa....aa=bb....bb
          :
aa....aa=bb....bb
```

aa....aa: Argument name.

bb....bb: Argument contents (up to 256 characters).

For reference type values, the object determines the format.

One asterisk (`*`) is output to *bb....bb* for the `password` argument of the following methods:

- `getConnection(String username, String password)` of the `DataSource` class
- `getPooledConnection(String username, String password)` of the `ConnectionPoolDataSource`
- `getXAConnection(String username, String password)` of `XADataSource`

For the `info` argument in `connect(String url, Properties info)` of the `Driver` class, the value of the each of the following properties is replaced by one asterisk (`*`) and then output:

- `password`
- `HiRDB_for_Java_ENV_VARIABLES`

HH....HH

Indicates the return value of the method, in the following format:

```
Return=aa....aa
```

aa....aa: Argument name.

This item is not output for methods that do not have a return value. If the return value is a reference-type value, the Java Virtual Machine determines the format.

II....II

Indicates troubleshooting information, in the following format:

```
ExceptionClass: aa....aa
UapEnvironment: bb....bb
Message: cc....cc
ErrorInfo: kk....kk
ErrorCode: dd....dd
SQLState: eeeee
UpdateCounts: ff....ff, ....<omitted>,ff....ff
SocketInfo: ll....ll
Etc.: gg....gg, hh....hh, iiii
jj....jj
```

aa....aa: Execution class name of the exception object that was thrown.

bb....bb: Client environment definitions being used in the connection of the exception object. The definitions are output in the following format (if no definitions are to be output, this variable is replaced by an asterisk (`*`) and then output):

```
yy....yy (zz....zz), ...<omitted>, yy....yy (zz....zz)
```

yy....yy: Name of the client environment definition without the initial `PD` characters. The following client environment definitions are the output targets.

Note that the information that is output depends on the connection-target server, as shown in the following table:

No.	Client environment definition	Connection target	
		Server providing primary functions	XDS
1	PDUSER	Y	Y
2	PDNAMEPORT	Y	Y
3	PDCWAITTIME	Y	Y

No.	Client environment definition	Connection target	
		Server providing primary functions	XDS
4	PDSWAITTIME	Y	Y
5	PDHOST	Y	Y
6	PDFESHOST	Y	Y
7	PDSERVICEGRP	Y	--
8	PDSWATCHTIME	Y	Y
9	PDSERVICEPORT	Y	Y
10	PDSRVTYPE	Y	Y
11	PDCLTRCVPORT	Y	--
12	PDCLTRCVADDR	Y	--
13	PDFESGRP	Y	--
14	PDXDSHOST	--	Y
15	PDXDSPORT	--	Y

Legend:

Y: Output

--: Not output

zz....zz: Contents of the client environment definition. The password portion of `PDUSER` is not output.

cc....cc: Message of the exception object.

dd....dd: `SQLCODE` error code (for `XAException`, error code indicated by the `errorCode` field of the `XAException` object) (up to 11 characters).

This item is output when the execution class of the thrown exception object is one of the following classes or subclasses:

- `SQLException`
- `XAException`

eeeee: `SQLSTATE` (5 characters).

This item is output when the execution class of the thrown exception object is `SQLException` or a subclasss of `SQLException`.

ff....ff: Number of update rows for each update statement in a batch update that was executed normally before this exception occurred (up to 11 characters).

This item is output when the execution class of the exception object is `BatchUpdateException`.

If the number of update rows cannot be obtained, an asterisk (`*`) is output.

gg....gg: SQL counter value (up to 6 characters).

This information can be used for coordinating with the trace information output by the SQL trace facility.

If the SQL counter cannot be obtained, an asterisk (`*`) is output.

hh....hh: Failure information of the HiRDB server when an error occurs in the HiRDB server (up to 22 characters).

The failure information is used by maintenance personnel.

If no errors have occurred in the HiRDB server, an asterisk (`*`) is output.

iiii: Type of request (operation code) that the JDBC driver issued to the HiRDB server when an error occurred in the HiRDB server.

If no errors have occurred in the HiRDB server, an asterisk (`*`) is output.

jj....jj: Stack trace in which the exception-throwing method is set as the base point.

The Java Virtual Machine determines the format.

kk....kk: Additional troubleshooting information is output in message format.

ll....ll: Socket information used to connect the object resulting in the exception, in the following format:

```
aa....aa(bb....bb), ...omitted..., aa....aa(bb....bb)
```

The contents of the socket information are as follows:

Name of socket information (aa....aa)	Contents of socket information (bb....bb)
LocalAddress	Local IP address
LocalPort	Local port number
SendBufferSize#	Send buffer size
RecvBufferSize#	Receive buffer size
SoLinger	SO_LINGER

Name of socket information (aa....aa)	Contents of socket information (bb....bb)
KeepAlive#	SO_KEEPALIVE
ReuseAddr#	SO_REUSEADDR

Note: If nothing is specified for socket information, only the parentheses () are output. If no socket information has been specified, item *ll....ll* is not output.

#: Output when the connection target is a server that provides primary functions.

18.15.3 Output example and analysis method

(1) Output example

An output example of the Exception trace log is shown below:

```
[1] HiRDB_Type4_JDBC_Driver 08-00
2006/07/06 23:07:09.129
Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.createStatement(1)]
                      ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:09.160
Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.createStatement(1)]
                      ConnectionID(sds:23:20484) : SID(0)
                      Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement@1e4cbc4
2006/07/06 23:07:09.160
Thread[main,5,main]@1259414:[E][PrdbStatement@1e4cbc4.execute]
                      ConnectionID(sds:23:20484) : SID(0)
                      sql=DELETE FROM SEINO_TABLE
2006/07/06 23:07:14.285 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                      ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                      ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301
Thread[main,5,main]@1259414:[R][PrdbStatement@1e4cbc4.execute]
                      ConnectionID(sds:23:20484) : SID(1)
                      Return=false
2006/07/06 23:07:14.301
Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.prepareStatement(1)]
                      ConnectionID(sds:23:20484) : SID(0)
                      sql=INSERT INTO SEINO_TABLE VALUES(?, ?)
2006/07/06 23:07:14.348
Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.prepareStatement(1)]
                      ConnectionID(sds:23:20484) : SID(0)
                 Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@15d56d5
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                      ConnectionID(sds:23:20484) : SID(0)
```

```
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:26.567
Thread[main,5,main]@1259414:[E][PrdbStatement@1e4cbc4.executeQuery]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:26.676
Thread[main,5,main]@1259414:[R][PrdbStatement@1e4cbc4.executeQuery]
                        ConnectionID(sds:23:20484) : SID(1)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbResultSet@3eca90
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(1)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332
Thread[Thread-0,5,main]@30090737:[E][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:28.332
Thread[Thread-0,5,main]@30090737:[R][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                    Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@2808b3

2006/07/06 23:07:28.348
Thread[Thread-1,5,main]@5462872:[E][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=DELETE FROM SEINO_TABLE WHERE I1=?
2006/07/06 23:07:28.358
Thread[Thread-1,5,main]@5462872:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:29.672
Thread[Thread-1,5,main]@5462872:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:30.098
Thread[Thread-1,5,main]@5462872:[R][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                    Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@922804
2006/07/06 23:07:30.332
Thread[Thread-2,5,main]@25253977:[E][PrdbConnection@82c01f.rollback(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[R][PrdbConnection@82c01f.rollback(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[E][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[R][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
```

```
2006/07/06 23:07:42.535 Thread[Thread-1,5,main]@5462872:Exception:
ExceptionClass: SQLException
UapEnvironment: *
Message: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
ErrorCode: -1020006
SQLState: R2400
Etc.: *,*,****
java.sql.SQLException: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
at
JP.co.Hitachi.soft.HiRDB.JDBC.JdbMakeException.generateSQLException(JdbMakeException
.java:31)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement.generateClosedSQLException(PrdbStatement
.java:3005)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement.setInt(PrdbPreparedStatement.jav
a:1170)
at Exception1.run(ExceptionTraceSample.java:57)
[2] HiRDB_Type4_JDBC_Driver 08-00
2006/07/06 23:07:25.723
Thread[Thread-3,5,main]@13249998:[E][PrdbConnection@119cca4.prepareStatement(1)]
                        ConnectionID(sds:24:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:25.770
Thread[Thread-4,5,main]@25839584:[E][PrdbConnection@119cca4.rollback(1)]
                        ConnectionID(sds:24:20484) : SID(0)
2006/07/06 23:07:25.770
Thread[Thread-4,5,main]@25839584:[R][PrdbConnection@119cca4.rollback(1)]
                        ConnectionID(sds:24:20484) : SID(0)
2006/07/06 23:07:25.770
Thread[Thread-5,5,main]@24431647:[E][PrdbConnection@119cca4.prepareStatement(1)]
                        ConnectionID(sds:24:20484) : SID(0)
                        sql=SELECT ** FROM SEINO_TABLE
2006/07/06 23:07:25.863 Thread[Thread-5,5,main]@24431647:Exception:
ExceptionClass: SQLException
UapEnvironment: USER(USER1), NAMEPORT(20249), CWAITTIME(0), SWAITTIME(600),
HOST(dragon2), FESHOST( ),
SERVICEGRP(sds), SWATCHTIME( ), SERVICEPORT( ), SRVTYPE(WS), CLTRCVPORT( ),
CLTRCVADDR( ), FESGRP( )
Message: KFPA11105-E Invalid token "*" after token "*"[PrdbStatement.prepare]
ErrorCode: -105
SQLState: R0000
Etc.: 4,sqapyac1.c(651),SET
java.sql.SQLException: KFPA11105-E Invalid token "*" after token
"*"[PrdbStatement.prepare]
at JP.co.Hitachi.soft.HiRDB.JDBC.CltSection.prepare(CltSection.java:1497)
at JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement.prepare(PrdbStatement.java:2834)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement.<init>(PrdbPreparedStatement.jav
a:109)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbConnection.prepareStatement(PrdbConnection.java:10
41)
at Exception1.run(ExceptionTraceSample.java:64)
```

(2) Analysis method

This item explains the analysis method of the Exception trace log. You can use a text editor to reference the Exception trace log.

Described below is an example of analyzing the Exception trace log shown in *(1) Output example*.

Analysis example

To analyze the Exception trace log:

1. Extract the sequentially numbered information, including the exception to be investigated.
2. Categorize the information by using the thread identification information, and separate the information by thread.
3. Arrange the information in time sequence based on the acquisition time.

   The following table shows what the results look like.

   Table 18-89: Example in which the Exception trace log is arranged in time sequence

Date and time	Thread 1	Thread 2	Thread 3	Thread 4
	Thread[main,5,main] @1259414	Thread[Thread-0,5,main] @30090737	Thread[Thread-1,5,main] @5462872	Thread[Thread-2,5,main] @25253977
`2006/07/06 23:07:09.129`	`PrdbConnection @82c01f.createStatement(1)`			
`2006/07/06 23:07:09.160`	`PrdbStatement @1e4cbc4.execute`			
`2006/07/06 23:07:14.285`	`PrdbConnection @82c01f.commit`			
`2006/07/06 23:07:14.301`	`PrdbConnection @82c01f.prepareStatement(1)`			
`2006/07/06 23:07:26.567`	`PrdbConnection @82c01f.commit`			
`2006/07/06 23:07:26.567`	`PrdbStatement @1e4cbc4.executeQuery`			

Date and time	Thread 1	Thread 2	Thread 3	Thread 4
	Thread[main,5,main] @1259414	Thread[Thread-0,5,main] @30090737	Thread[Thread-1,5,main] @5462872	Thread[Thread-2,5,main] @25253977
2006/07/06 23:07:26.567	PrdbStatement @1e4cbc4.execute			
2006/07/06 23:07:28.332	PrdbResultSet @3eca90.close	PrdbConnection @82c01f.prepareStatement(1)		
2006/07/06 23:07:28.332	PrdbConnection @82c01f.commit			
2006/07/06 23:07:28.348			PrdbConnection @82c01f.prepareStatement(1)	
2006/07/06 23:07:28.358			PrdbConnection @82c01f.commit	
2006/07/06 23:07:30.332				PrdbConnection @82c01f.rollback
2006/07/06 23:07:42.098				PrdbConnection @82c01f.close
2006/07/06 23:07:42.535			SQLException occurred KFPJ20006-E Connection closed	

4. Check the contents of the exception error.

   The information indicates that an SQLException occurred in Thread 3 at 2006/07/06 23:07:42.535, and that a Statement or Connection object had already been closed.

5. Check the operation of the object in the time sequence.

   Because the object ID of the `Connection` object in the next thread is the same, we know that the four threads were being processed in the same connection.

   - Thread 1 at 2006/07/06 23:07:09.129
   - Thread 2 at 2006/07/06 23:07:28.332
   - Thread 3 at 2006/07/06 23:07:28.348
   - Thread 4 at 2006/07/06 23:07:30.332

6. Search for the location of the error cause.

   Because we know that the four threads have the same connection, we can search for the locations where the `Statement.close` or `Connection.close` method was executed, and learn that Thread 4 executed the `Connection.close` method at 2006/07/06 23:07:42.098. From this, we know that the cause of the SQLException that occurred in Thread 3 at 2006/07/06 23:07:42.535 was that Thread 4 executed the `Connection.close` method at 2006/07/06 23:07:42.098.

18.15.4 Required memory size and file size

(1) Required memory size

The memory size required for acquiring the Exception trace log is determined from the following formula:

Formula

$$\uparrow 360 \times n/1024 \uparrow \text{ (kilobytes)}$$

Explanation

n: Number of information items to be acquired in memory

(2) Required file size

The file size for acquiring the Exception trace log is determined from the following formula:

Formula

$$\uparrow 180 \times n \times m/1024 \uparrow + 1 \text{ (kilobytes)}$$

Explanation

n: Number of information items to be acquired in memory

m: File output information

18.15.5 Notes

(1) If the system properties and client environment definition settings are different

The following table shows how the method execution history that was accumulated in the JDBC driver memory before establishment of the first HiRDB connection is transferred when the system properties and client environment definition settings are different.

Table 18-90: Transfer of the method execution history accumulated in the JDBC driver memory

Item	Relationship between system properties and client environment definition	Transfer operation
Number of information items to be acquired in memory	`HiRDB_for_Java_OnMemNUM` < `PDJDBONMEMNUM`	The JDBC driver re-allocates memory for accumulation of the method execution history based on the `PDJDBONMEMNUM` specification, and then copies the execution history accumulated up to that point to the re-allocated area. However, if the `PDJDBTRACELEVEL` specification is `0`, memory is not re-allocated.
	`HiRDB_for_Java_OnMemNUM` > `PDJDBONMEMNUM`	The JDBC driver re-allocates memory for accumulation of the method execution history based on the `PDJDBONMEMNUM` specification. The driver then destroys any accumulated execution history information that cannot be stored in the re-allocated area, and copies the remaining information to the re-allocated area. However, if the `PDJDBTRACELEVEL` specification is `0`, memory is not re-allocated.

Item	Relationship between system properties and client environment definition	Transfer operation
Trace acquisition level	HiRDB_for_Java_TraceLevel < PDJDBTRACELEVEL	The driver simply transfers the execution history that was accumulated up to that point.
	HiRDB_for_Java_TraceLevel > PDJDBTRACELEVEL	If the PDJDBTRACELEVEL specification is 1 or greater, the JDBC driver simply transfers the execution history accumulated up to that point. The driver also transfers the execution history of methods that are not targeted by the trace acquisition level specified by PDJDBTRACELEVEL. If the PDJDBTRACELEVEL specification is 0, the JDBC driver destroys the accumulated execution history for each accumulation memory.

(2) First output after startup of the Java Virtual Machine

The first time the Exception trace log is output to a file after the Java Virtual Machine is started, the log is output to the file with the older update date and time. If the date and time are the same for both files, the log is output to `pdexc1.trc`.

(3) Specification of the file output destination

If the same file output destination is specified when Exception trace logs are being acquired from multiple processes, trace information for the different processes is output to the same file. To acquire a trace for each process, specify a different file output destination for each process.

The JDBC driver uses the facilities of the Java Virtual Machine to create log files in the file system provided by the OS. Therefore, the following items depend on the Java Virtual Machine and file system being used:

- Prefix for the absolute path name
- Path delimiter character
- Maximum number of characters for the output destination file (absolute path)
- Size per file

(4) Processing when an error occurs

Information is not output to the Exception trace log when file creation or output fails. An error message may be returned to the UAP and file output may be retried.

(5) Character encoding

The Exception trace log is output with the default conversion character set of the Java Virtual Machine being used.

18.16 Example UAP that uses a JDBC driver

This section presents an example of a UAP that uses a JDBC driver.

This UAP executes SQL statements that display the rows that satisfy specified conditions. The UAP checks the SQL statements for errors and acquires error information when errors are detected.

```
import java.sql.*;
import java.io.*;
import java.util.Properties;

public class SAMPLE {

    public static void main(String[] args) {

        //Connection object variables
        String url = "jdbc:hitachi:hirdb://DBID=22200,DBHOST=host1";
        String user = "USER1";
        String passwd = "USER1";
        String driver = "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver";
        try {
            Class.forName(driver);
        }catch(Exception e){e.printStackTrace();return;}

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;


        // *** Environment variables ***
        java.util.Properties info = new java.util.Properties();
        info.setProperty("user",user);
        info.setProperty("password",passwd);
        info.setProperty("HiRDB_for_Java_ENV_VARIABLES",
                     "PDCLTPATH=C:\\tmp;PDSQLTRACE=0;PDPRMTRC=INOUT;");
        // ************

        // Register and load the driver
        System.setProperty("jdbc.drivers",
                           "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");

        // Configure an environment
        try {
            // Connect .................................................1
            con = DriverManager.getConnection(url,info);
            // *********************************************
       // Example of retrieval by SAMPLE1(C1 INT,C2 INT,C3 VARCHAR(30)).........2
            // *********************************************
            // Acquire PreparedStatement
            pstmt = con.prepareStatement
                            ("SELECT C2,C3 FROM SAMPLE1 WHERE C1 = ? ");
            // Set a ? parameter (C1 = 200)
            pstmt.setInt(1,200);
            // Acquire ResultSet
            rs = pstmt.executeQuery();
            int cnt=1;
            System.out.println("**** Retrieving ****");
            while(rs.next()){
                System.out.println("**** Retrieving row "+cnt+" ***");
                // Acquire and display C2 data
                int i_data = rs.getInt(1);
                System.out.println("C2="+i_data);
                // Acquire and display C3 data
                String c_data = rs.getString(2);
                System.out.println("C3="+c_data);
                cnt++;
            }
            // Release ResultSet
            rs.close();
            // Release PreparedStatement
            pstmt.close();
            // Disconnect.................................................3
            con.close();
            }


        catch(SQLException e){  .....................................4
            // Output error information
            e.printStackTrace();
            // Acquire individual information item by the following processing
            // Output SQLSTATE
            System.out.println("SQLSTATE=" + e.getSQLState());
            // Output SQLCODE
            System.out.println("SQLCODE=" + e.getErrorCode());
            // Output SQLERRM (SQL message)
            System.out.println("SQLERRM=" + e.getMessage());
            return;
        }
    }
}
```

Explanation:

1. Uses the `getConnection` method to connect to HiRDB.
2. Executes the SQL statement that displays the rows that satisfy the specified conditions.
3. Uses the `close` method to disconnect from HiRDB.

4. If an error occurs, the UAP returns `SQLException` and outputs error information.

18.17 Estimating the memory requirements for using a JDBC driver

This section describes how to estimate the size of the Java heap to be used by the JDBC driver.

18.17.1 Estimating the Connection object size

The following shows the formulas for estimating the `Connection` object size.

Formula (for 32-bit mode)

Connection object size =
70000 + (11000 + *send buffer size* + *receive buffer size*) x *number of Connection instances*
(bytes)

Formula (for 64-bit mode)

Connection object size =
304000 + (14000 + *send buffer size* + *receive buffer size*) x *number of Connection instances*
(bytes)

Send buffer size: Use the formula shown below to determine the value. If multiple SQL statements are executed, you must obtain the send buffer size for each SQL statement and then use the largest value.

MAX(32768,
↑(240 + *length of SQL statement* + *length of user-added information*) ÷ 4096↑ x 4096,
↑(240 + *length of input parameter information* + *length of output parameter information*
+ *length of user-added information*) ÷ 4096↑ x 4096)

Length of SQL statement: Length of SQL statement (`String`) that has been converted to the HiRDB server's character codes

Length of user-added information: Sum of the lengths of user-added information 1, user-added information 2, and user-added information 3. Add this value if you use `Connection.setHiRDB_Audit_Info`.

User-added information 1: Use the following formula to determine the value:

16 + length of user-added information 1 (`String`) converted to HiRDB

server's character codes

User-added information 2: Use the following formula to determine the value:

16 + length of user-added information 2 (String) converted to HiRDB server's character codes

User-added information 3: Use the following formula to determine the value:

16 + length of user-added information 3 (String) converted to HiRDB server's character codes

Length of input parameter information: If there are input parameters, obtain this value from the following formula and then add the resulting value:

$$24 + \sum_{i=1}^{\textit{Number of input parameters}} (16 + \textit{input data length}_i)$$

Number of input parameters: See the following table:

Object	Number of input parameters
PreparedStatement	Number of ? parameters
CallableStatement	Number of IN and INOUT parameters

Input data length: Real data length set by a set*XXXX* method

If the length was set by the setString method, this is the length of input data that has been converted to the HiRDB server's character codes.

Length of output parameter information: If the values of OUT and INOUT parameters are to be acquired by executing retrieval SQL statements or stored procedures, add the value obtained from the following formula:

24 + 16 x number of output parameters

Number of output parameters: If you execute retrieval SQL statements, this is the number of columns to be retrieved. If you execute stored procedures, this is the total number of OUT and INOUT parameters.

Receive buffer size: Use the formula shown below to determine the value. If you execute multiple SQL statements, obtain a receive buffer size for each SQL statement and then use the largest value.

$$\begin{aligned}&\mathrm{MAX}(32768,\\ &\ \uparrow(240 + 52 \times (\textit{number of input parameters} + \textit{number of output parameters})) \div 4096\uparrow \times 4096,\\ &(\uparrow(240 + \textit{length of output data during data access by the locator facility}) \div 4096\uparrow \times 4096)^{\#}\end{aligned}$$

#: Use one of the following specifications to obtain this value when you use the locator facility to access data:

- Specification of `LOCATOR` in the `url` argument of `DriverManager.getConnection` or in the `LONGVARBINARY_ACCESS` user property
- Specification of `LOCATOR` in the `setLONGVARBINARY_Access` method of a `DataSource` interface

Number of input parameters: See the following table:

Object	Number of input parameters
Statement	0
PreparedStatement	Number of `?` parameters
CallableStatement	Number of `IN` and `INOUT` parameters

Number of output parameters: The value is the same as for the send buffer size.

Length of output data during data access by the locator facility: Use either of the following, as appropriate:

- When a non-zero value is specified in `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE`:

  Use the following formula to determine the value:

  `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE` value x 1024
- When `0` is specified in `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE` or `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE` is omitted:

  Real length of data stored in `BLOB`-type or `BINARY`-type columns

18.17.2 Estimating the Statement object size

The following shows the formulas for estimating the `Statement` object size.

Formula (for 32-bit mode)

Statement object size =
38000 + (2000 + *retrieval results storage area size*) x *number of Statement instances* (bytes)

Formula (for 64-bit mode)

Statement object size =
38000 + (2100 + *retrieval results storage area size*) x *number of Statement instances* (bytes)

Retrieval results storage area size: The value is `0` for non-retrieval SQL statements.

For retrieval SQL statements, use the following formula to determine the value:

$$\sum_{i=1}^{\textit{Number of columns to be retrieved}} (120 + \textit{definition length of data in the columns to be retrieved}_i)$$

Definition length of data in the column to be retrieved: If the column to be retrieved is `BLOB` or `BINARY` and the locator facility is used, the value is `4`. If the locator facility is not used, the value is one of the following:

- If `HiRDB_for_Java_MAXBINARYSIZE` is omitted:

  8 + definition length

- If `HiRDB_for_Java_MAXBINARYSIZE` is specified:

  MIN(8 + *definition length*, 8 + *MAXBINARYSIZE value*)

18.17.3 Estimating the PreparedStatement object size

The following shows the formulas for estimating the `PreparedStatement` object size.

Formula (for 32-bit mode)

PreparedStatement object size =
39000 + (2100 + *input data storage area size* + *retrieval results storage area size*)
x *number of PreparedStatement instances* (bytes)

Formula (for 64-bit mode)

PreparedStatement object size =
40000 + (2600 + *input data storage area size* + *retrieval results storage area size*)
x *number of PreparedStatement instances* (bytes)

Input data storage area size: If `?` parameters are used, use the following formula to determine the value:

$$\sum_{i=1}^{\text{Number of ? parameters}} (120 + \textit{input data length}_i)$$

The retrieval results storage area size is the same as for the `Statement` object.

18.17.4 Estimating the CallableStatement object size

The following shows the formulas for estimating the `CallableStatement` object size.

Formula (for 32-bit mode)

CallableStatement object size =
39000 + (2100 + *input data storage area size* + *output data storage area size*)
x *number of CallableStatement instances* (bytes)

Formula (for 64-bit mode)

CallableStatement object size =
40000 + (2600 + *input data storage area size* + *output data storage area size*)
x *number of CallableStatement instances* (bytes)

Input data storage area size: If the `IN` and `INOUT` parameters are used, use the following formula to determine the value:

$$\sum_{i=1}^{\text{Number of IN and INOUT parameters}} (120 + \textit{data length for IN and INOUT parameters}_i)$$

The method for determining the data length for `IN` and `INOUT` parameters is the same as for the method for determining the length of input parameter information for `Connection` objects.

Output data storage area size: If the `INOUT` and `OUT` parameters are used, use the following formula to determine the value:

$$\sum_{i=1}^{\text{Number of INOUT and OUT parameters}} (120 + \textit{definition length of data for INOUT and OUT parameters}_i)$$

Definition length of data for `INOUT` and `OUT` parameters: If the `INOUT` and `OUT` parameters are `BLOB` or `BINARY` and the locator facility is used, the value is 4. If the locator facility is not used, the value is one of the following:

- If `HiRDB_for_Java_MAXBINARYSIZE` is omitted:

  8 + definition length

- If `HiRDB_for_Java_MAXBINARYSIZE` is specified:

  MIN(8 + *definition length*, 8 + *HiRDB_for_Java_MAXBINARYSIZE value*)

18.17.5 Estimating the ResultSet object size

The following shows the formulas for estimating the `ResultSet` object size.

Formula (for 32-bit mode)

ResultSet object size =
16000 + (900 + *locator access object length* + *receive buffer size*)
x *number of ResultSet instances* + *retrieval results accumulation area size x number of ResultSet instances 2*
(bytes)

Formula (for 64-bit mode)

ResultSet object size =
17000 + (1300 + *locator access object length* + *receive buffer size*)
x *number of ResultSet instances* + *retrieval results accumulation area size* x *number of ResultSet instances 2*
(bytes)

Locator access object length: If the locator facility is used (`LOCATOR` specified in the `url` argument of `DriverManager.getConnection` or in the `LONGVARBINARY_ACCESS` user property, or `LOCATOR` specified by the `setLONGVARBINARY_Access` method of a `DataSource` interface), add 1,600.

Receive buffer size: Use the following formula to determine the value:

$$\text{MAX}(32768,\ \uparrow(240 + \textit{data length of retrieved row} \times \textit{number of rows for block transfer}) \div 4096 \uparrow \times 4096)$$

Data length of retrieved row: Use the following formula to determine the value:

$$8 + \sum_{i=1}^{\textit{Number of columns to be retrieved}} (\textit{definition length of column data}_i)$$

Definition length of column data: If the data is `BLOB` or `BINARY` and a locator is used, the value is 4. If a locator is not used, the value is one of the following:

- If `HiRDB_for_Java_MAXBINARYSIZE` is omitted:

  8 + definition length

- If `HiRDB_for_Java_MAXBINARYSIZE` is specified:

  MIN(8 + *definition length*, 8 + *HiRDB_for_Java_MAXBINARYSIZE value*)

Number of rows for block transfer: If the block transfer facility is not used, the value is 1.

Retrieval results accumulation area size: Use the following formula to determine the value:

$$\sum_{i=1}^{\textit{Number of columns to be retrieved}} (100 + \textit{real length of column data}_i) \times \textit{number of rows to be retrieved}$$

Real length of column data: See the following table:

Output data attribute	Real length of column data
INTEGER	4
SMALLINT	2
DECIMAL(m, n)	$\downarrow m \div 2 \downarrow + 1$

Output data attribute	Real length of column data
FLOAT	8
SMALLFLT	4
BLOB or BINARY	If the locator facility is used, the value is 4; if not, the value is one of the following: • If HiRDB_for_Java_MAXBINARYSIZE is omitted: 8 + *real length* • If HiRDB_for_Java_MAXBINARYSIZE is specified: MIN(8 + *real length*, 8 + *MAXBINARYSIZE value*)
Other	Real length

Number of ResultSet instances 2: Number of ResultSet instances whose result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE

18.17.6 Estimating the size of trace objects

The total size of trace objects is equal to the total amount of memory required for collecting the following troubleshooting information:

- Interface method traces
- Exception trace logs
- SQL traces

(1) Interface method traces

Add this value if you collect interface method traces.

Formula

Size of interface method traces = 300 x *number of entries* x *number of Connection instances* (bytes)

Number of entries: Either of the following values, as appropriate:

- Value of the url argument in DriverManager.getConnection or the value of the TRC_NO user property
- Value specified in the setTRC_NO method of a DataSource interface

(2) Estimating the size of Exception trace logs

Add this value when the acquisition level for Exception trace logs is not 0.

Formula

Size of Exception trace logs = 360 x *number of data items to be collected in memory* (bytes)

Number of data items to be collected in memory: The value is one of the following:

- Value of the `HiRDB_for_Java_OnMemNUM` system property
- Value of the `PDJDBONMEMNUM` client environment definition

(3) Estimating the size of SQL traces

Add this value if you collect SQL traces.

Formula

Size of SQL traces = 10,000 x *number of Connection instances* (bytes)

Chapter

19. SQLJ

This chapter explains how to use SQLJ to develop a UAP. Note that SQLJ cannot be used in the Linux for AP8000 edition client.

19.1 Overview
19.2 SQLJ Translator
19.3 UAP coding rule
19.4 Native Runtime

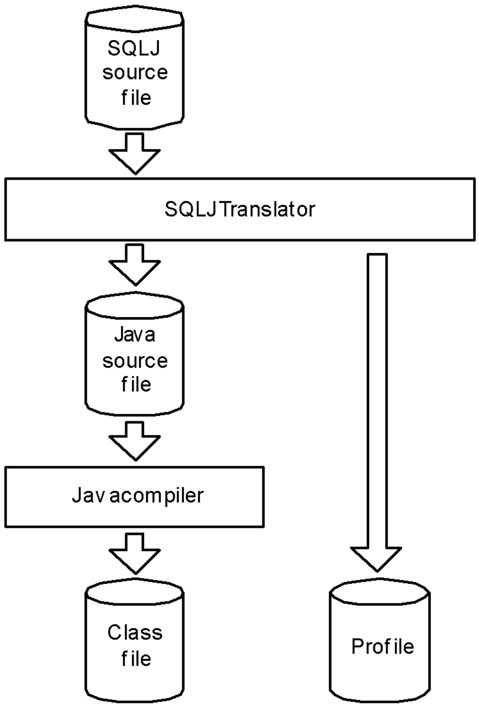
19.1 Overview

19.1.1 What is SQLJ?

SQLJ is a language specification for coding a static SQL statement as an embedded SQL statement in Java and executing it.

The following figure shows the flow of UAP development when SQLJ is used.

Figure 19-1: Flow of UAP development that uses SQLJ



SQLJ consists of SQLJ Translator and SQLJ Runtime Library.

SQLJ Translator

SQLJ Translator analyzes an SQLJ source program and replaces SQL statements with standard Java instructions for accessing a database through SQLJ Runtime Library.

SQLJ Translator generates a Java source file and a profile that stores SQL information. The user uses the Java compiler to compile the Java source file to create a `class` file (executable file).

SQLJ Runtime Library

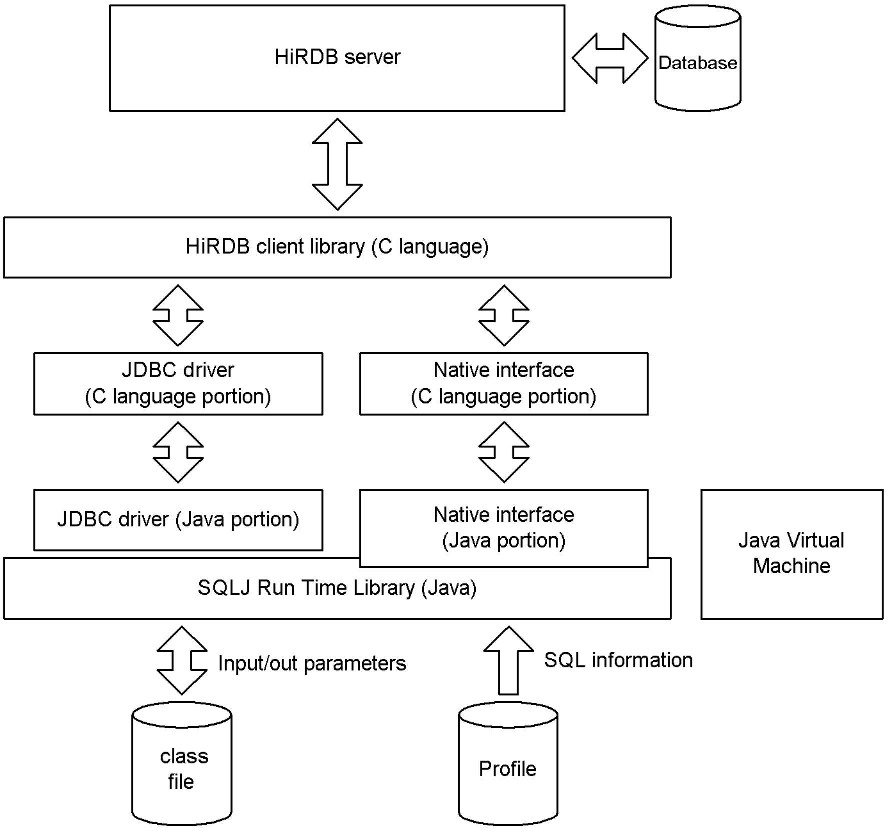
SQLJ Runtime Library is used for executing a compiled `class` file.

SQLJ Runtime Library can be used in either of the following ways, depending on the access interface used:

- Invoke the JDBC interface API (standard interface edition) and execute the SQL statements.
- Invoke an original interface (native interface edition), not the JDBC interface, and execute the SQL statements.

The following figure shows UAP execution when SQLJ is used.

Figure 19-2: Execution of a UAP that uses SQLJ



Explanation:

- A `class` file of the SQLJ source file compiled by the Java compiler accesses a database through the SQLJ Runtime Library.

- When the native interface is used, SQLJ Runtime Library directly invokes a HiRDB client library instead of invoking JDBC. In this case, you cannot use coding that directly invokes the JDBC API and shares connection and result sets with JDBC.
- Because the SQLJ Runtime Library loads a profile during execution, the `class` file and profile must be stored in the same directory. Also, when the `class` file is stored in the `jar` file, you must also store the profile in the `jar` file.

19.1.2 Environment settings

The environment variable settings required for SQLJ operation are shown below. Since SQLJ uses the JDBC driver, environment settings for the JDBC driver must also be specified.

(1) Environment settings for the UNIX edition

Set the following information in the environment variables for the execution environment.

(a) HiRDB/Developer's Kit

```
CLASSPATH=$CLASSPATH:[installation-directory]/client/lib/pdsqlj.jar#
```

#: For the 32-bit mode HP-UX (IPF) edition, specify `pdsqlj32.jar`.

(b) HiRDB/Run Time

```
CLASSPATH=$CLASSPATH:[installation-directory]/client/lib/
pdruntime.jar#1
CLASSPATH=$CLASSPATH:[installation-directory]/client/lib/
pdnativert.jar#2
```

Note

When using the 32-bit mode HP-UX (IPF) edition, do not specify the following pairs of files at the same time:

- `pdsqlj.jar` and `pdsqlj32.jar`
- `pdruntime.jar` and `pdruntime32.jar`
- `pdnativert.jar` and `pdnativert32.jar`

#1: For the 32-bit mode HP-UX (IPF) edition, specify `pdruntime32.jar`.

#2: For the 32-bit mode HP-UX (IPF) edition, specify `pdnativert32.jar`.

(2) Environment settings in the Windows edition

In sequence, choose **Control Panel**, **System**, **System Properties**, and **Environment**, and then set the contents shown below.

(a) HiRDB/Developer's Kit

```
CLASSPATH=%CLASSPATH%:[installation-directory]\UTL\pdsqlj.jar
```

(b) HiRDB/Run Time

```
CLASSPATH=%CLASSPATH%:[installation-directory]\UTL\pdruntime.jar
CLASSPATH=%CLASSPATH%:[installation-directory]\UTL\pdnativert.jar
```

19.2 SQLJ Translator

SQLJ Translator analyzes an SQLJ source program and generates a Java source file and a profile.

SQL statements are replaced with Java instructions, including the invocation of the JDBC API, and are output as a Java source file.

SQL character strings, number of parameters, types and modes of individual parameters, and the description of the columns to be output to a profile. The profile is referenced from the SQLJ Runtime Library. The entity of a profile is an instance of the `java.sql.runtime.Profile` class.

The following table lists the files that are generated and referenced by the SQLJ Translator.

Table 19-1: Files that are generated and referenced by the SQLJ Translator

File classification	File name format	Explanation	Type
SQLJ source file	*file-name*`.sqlj`	Indicates an SQLJ source file.	Referenced
Java source file	*file-name*`.java`	Indicates a Java source file.	Generated
Profile	*file-name*`_SJProfile` *profile-number*`.ser`	Stores the information of each SQL statement extracted from an SQLJ source file. A profile number is assigned to each context. The cardinal number is `0` for all.	Generated

The prefixes of the classes and variables internally generated by SQLJ Translator are as follows:

`_sJT_`: Names of the variables internally generated

`_SJ`: Names of the classes and profiles internally generated

19.3 UAP coding rule

This section explains the coding rule for SQLJ source files.

19.3.1 Labeling rule

The following labels cannot be used:

- Label that begins with `_sJT_`
- Label that begins with `_SJf`
- Label that begins with `p_rdb`

Other rules are based on the Java language rules.

19.3.2 SQL coding rule

(1) SQL statement coding rule

Each SQL statement must be enclosed between the SQL leading character string (`#sql`) and the SQL trailing character (`;`). The SQL statement itself must further be enclosed between curly brackets. Connection class and cursor declarations must also be enclosed between the SQL leading character string and the SQL trailing character.

The following table shows the SQL statement coding formats.

Table 19-2: SQL statement coding formats

Function	Format	Purpose
SQL execution	`#sql [`*context*`] {` *SQL-statement* `} ;`	Executes an SQL statement. The SQL statements that can be used differ for the standard interface edition and the native interface edition. For details, see *19.3.3 SQL statements that can be used in SQLJ*.
Declaration of an iterator class with column specification	• Standard interface edition `#sql` *modifier* `iterator` *class-name* `(`*data-type column-name*`,...) ;` • Native interface edition This function cannot be used.	Declares the class to be used for cursor declaration. Cannot be used in a `FETCH` statement.

Function	Format	Purpose
Declaration of an iterator class with a position specification	• Standard interface edition #sql *modifier* iterator *class-name* (*data-type*, ...) ; • Native interface edition #sql *modifier* iterator *class-name* [implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime. ForUpdate] [with (*keyword*=*value*, ...)] (*data-type*, ...) ;	Declares the class to be used in the cursor declaration. This function is used in a FETCH statement.
Declaration of a connection class	#sql *modifier* context *class-name* ;	Declares the class to be used for connection.
Declaration of a cursor	#sql *iterator-object* = { *SELECT-statement* } ;	Defines and opens a cursor.
Conversion of a result set	• Standard interface edition #sql [*context*] *iterator-object* = {CAST :*JDBC-result-set*} ; • Native interface edition This function cannot be used.	Converts a JDBC result set into one that can be used by SQLJ.

Notes

modifier

Combination of `private`, `public`, `protected`, `final`, `abstract`, `protected`, `static`, `native`, `synchronized`, `transient`, and `volatile`.

context

{*connection-context* | *connection-context*,*execution-context* | *execution-context*}

keyword

`holdability` or `updateColumns`

value

`true`, `false`, or "*column-name-1*,*column-name-2*,..."

data-type

Java data type

column-name

Retrieval item

(2) Explicitly specifying connection context when using the multi-connection facility

When you use the multi-connection facility, insert the connection context surrounded by square brackets between the SQL leading character string and the SQL statement, to explicitly specify the connection to be used. An example follows:

```
#sql [connCtx] { DELETE FROM EMP WHERE SAL > 1000};
```

If no connection context is explicitly specified, the default connection context is assumed.

(3) Explicitly specifying an execution environment

In SQLJ, a user can explicitly specify an execution environment instead of using the default one. To specify an execution environment, insert the execution connection context surrounded by square brackets between the SQL leading character string and the SQL statement.

If SQL statements are simultaneously being executed in multiple threads for a single connection, using separate multiple execution environments can prevent an execution result from being overwritten by another SQL statement. An example follows:

```
ExecutionContext execCtx = new ExecutionContext();
try {
 #sql [execCtx] { DELETE FROM STOCK WHERE PCODE > 1000 };
 System.out.println
   ("removed " + execCtx.getUpdateCount() + "goods");
}
catch(SQLException e){
 System.out.println("SQLException has occurred with "+ " exception " + e);
}
```

If no execution connection context is explicitly specified, the default execution environment is used.

The values described in the following table are maintained in the execution environments. These values are set using the `set`<*name*> method and determined using the `get`<*name*> method.

Name	Details
`MaxRows`	Maximum number of rows to be returned from a search.
`MaxFieldSize`	Maximum size of data in units of bytes to be returned in columns and `OUTPUT` variable value.
`QueryTimeout`	Maximum wait time until SQL execution is completed. This is invalid in HiRDB.
`UpdateCount`	Number of updated, inserted, or deleted rows (reference only).

Name	Details
`SQLWarnings`	Correspond to `SQLWARN0`-`SQLWARNF` (reference only).

If multi-connection is also specified, the connection context and execution connection context must be specified in that order, delimited by a comma. An example follows:

```
#sql [connCtX, execCtx] { DELETE FROM STOCK WHERE PCODE > 1000 };
```

(4) Specifying embedded variables

In SQLJ, `BEGIN DECLARE SECTION` for declaring embedded variables is not used.

Any variables, parameters, and object fields can be used as embedded variables. In an SQL statement, a variable is described as `:`*variable-name* with a colon at the front. The colon can be separated by spaces from the variable name.

The `IN`, `OUT`, or `INOUT` parameter of a `CALL` statement is described as `:{IN|OUT|INOUT}`*variable-name*.

Additionally, in SQLJ, you can use `:(`*expression*`)` as an embedded variable. The expression must be enclosed by parentheses. This is a Java method and not an SQL method. An example follows:

```
#sql { SELECT COL1, COL2 FROM TABLE1 WHERE :(x[--i]) > COL3 };
```

(5) Specifying indicator variables

SQLJ has no indicator variable. Therefore, to set a null value for an embedded variable, use the `Wrapper` type defined in the `sql.lang` package instead of the basic data type. If a null value is received by a Java variable of the basic data type, the `SQLNullException` exception occurs.

(6) Exception handling

SQLJ cannot handle exceptions from an embedded SQL `WHENEVER` statement. Therefore, Java exception handling (`try...catch`) is used instead of `WHENEVER`. An example follows:

```
try{
   #sql { DELETE FROM STOCK WHERE PCODE > 1000 };
}
catch(SQLException e){
   System.out.println("SQLCODE:" + e.getErrorCode() +
                      "\nERRMSG:" + e.getMessage() );
}
```

If an error occurs during SQL execution, the JDBC exception object (`java.sql.SQLException`) is issued.

SQLCODE, SQLSTATE, and error messages are stored in exception objects, and their values can be obtained using the getErrorCode, getSQLState, and getMessage methods.

(7) Static SQL statements and dynamic SQL statements

In SQLJ, only static SQL statements can be described. Dynamic SQL statements cannot be described.

To use a dynamic SQL statement, use the JDBC API.

(8) Reading out the result set of a dynamic cursor

You can use a CAST statement to convert and read out the result set of a dynamic cursor created using the JDBC API as the result set of an SQLJ cursor. An example follows:

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String Query="SELECT pname, pcode FROM stock WHERE pcode > 1000";
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps ={CAST :rs };
```

The CAST statement cannot be used with the native interface edition. If the statement is used, a translation error results.

(9) Connecting to and disconnecting from a HiRDB server

The CONNECT and DISCONNECT statements can be used in the native interface edition but not in the standard interface edition. For both the standard interface and native interface editions, Java instructions can be used to connect to or disconnect from a HiRDB server.

(10) Exception generation conditions

In HiRDB embedded SQL statements, an alarm is issued in the following cases. In contrast, exceptions occur in SQLJ.

- In a single-row SELECT statement, the number of search items does not match the number of variables specified in an INTO clause.
- In a single-row SELECT statement, the retrieval result has zero rows.
- In a single-row SELECT statement, the retrieval result has multiple rows.
- In a FETCH statement, the number of search items does not match the number of variables specified in an INTO clause.
- The number of columns defined by an iterator with a position specification does not match the number of retrieval items.
- The number of columns defined by an iterator with a column name specification

is greater than the number of retrieval items.

(11) Comments and handling of SQL optimization specification

Comments (/*-*/) described between the SQL leading character string and the SQL trailing character are deleted. However, in cursor declaration and SQL statement execution, the SQL optimization specification (/*>>-<<*/) described between curly brackets is not deleted and handled as an SQL statement. All other SQL optimization specifications (/*>>-<<*/) are treated as comments. For details on comments and SQL optimization specifications inside SQL statements, see the manual *HiRDB Version 9 SQL Reference*.

19.3.3 SQL statements that can be used in SQLJ

The following table lists the SQL statements that can be used in SQLJ.

Table 19-3: SQL statements that can be used in SQLJ

Type	SQL statement	Usability		Alternate means
		Standard interface edition	Native interface edition	
Definition SQL statement	All	Y	Y	None
Data manipulation SQL statements	`ASSIGN LIST` statement	N (SQLJ)	N (SQLJ)	Use JDBC.
	`CALL` statement	Y	Y	None
	`CLOSE` statement	N (SQLJ)	N (SQLJ)	Use an iterator.
	`DECLARE CURSOR`	N (SQLJ)	N (SQLJ)	
	`DELETE` statement	Y	Y	None
	`DESCRIBE` statement	N (JDBC)	N (JDBC)#	Use JDBC.
	`DESCRIBE TYPE` statement	N (JDBC)	N (JDBC)#	
	`DROP LIST` statement	N (JDBC)	N (JDBC)#	
	`EXECUTE` statement	N (JDBC)	N (JDBC)#	
	`EXECUTE IMMEDIATE` statement	N (JDBC)	N (JDBC)#	
	`FETCH` statement (Format 1 or 3)	Y	Y	None
	`FETCH` statement (Format 2)	N	N	None
	`INSERT` statement	Y	Y	None

Type	SQL statement	Usability		Alternate means
		Standard interface edition	Native interface edition	
	OPEN statement (Format 1)	N (SQLJ)	N (SQLJ)	Use an iterator.
	OPEN statement (Format 2)	N (SQLJ)	N (SQLJ)	Use JDBC.
	PREPARE statement	N (SQLJ)	N (SQLJ)	
	PURGE TABLE statement	Y	Y	None
	Single-row SELECT statement	Y	Y	None
	Dynamic SELECT statement	N (JDBC)	N (JDBC)#	Use JDBC.
	UPDATE statement	Y	Y	None
Control SQL statements	COMMIT statement	Y	Y	None
	COMMIT statement (RELEASE specified)	N (SQLJ)	N (SQLJ)	Split into COMMIT and DISCONNECT.
	CONNECT statement	N	Y	None
	DISCONNECT statement	N	Y	None
	LOCK statement	Y	Y	None
	ROLLBACK statement	Y	Y	None
	ROLLBACK statement (RELEASE specified)	N (SQLJ)	N (SQLJ)	Split into ROLLBACK and DISCONNECT.
	SET SESSION AUTHORIZATION statement	N	N	None
Embedded language syntax	BEGIN DECLARE SECTION	N	N	None
	END DECLARE SECTION	N	N	None
	ALLOCATE CONNECTION HANDLE	N (SQLJ)	N (SQLJ)	Use a connection context.
	DECLARE CONNECTION HANDLE	N (SQLJ)	N (SQLJ)	
	FREE CONNECTION HANDLE	N (SQLJ)	N (SQLJ)	

Type	SQL statement	Usability		Alternate means
		Standard interface edition	Native interface edition	
	GET CONNECTION HANDLE	N	N	None
	COPY	N	N	None
	GET DIAGNOSTICS	N	N	None
	WHENEVER	N (SQLJ)	N (SQLJ)	Implement using try ... catch.

Legend:

Y: Can be used in SQLJ.

N (SQLJ): Cannot be used in SQLJ, but a similar function is available in the functions provided by SQLJ or JAVA.

N (JDBC): Cannot be used in SQLJ, but a similar function is available when JDBC is used.

N: Cannot be used in SQLJ.

None: There is no alternate means.

Note

SQLJ cannot use HiRDB functions that are not provided by the JDBC driver. The following functions cannot be used:

- UPDATE statement and DELETE statement that use an iterator
- Specification of a keyword in a WITH clause during the declaration of an iterator
- INSERT function that uses an array

#: If you use a JDBC connection object to create a connection context, you can also use the alternate means with the native interface. If you do not use a JDBC connection object to create a connection context, you cannot use the alternate means.

19.3.4 Correspondence between HiRDB data types and SQLJ data types

The table below shows the correspondence between the HiRDB data types and the SQLJ data types. To use embedded variables in SQLJ, declare variables according to this table.

Table 19-4: Correspondence between HiRDB data types and SQLJ data types

HiRDB data types	SQLJ data types (Java data types)	
	When a null value is included	When a null value is not included
CHAR[#1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR[#4]	N/A
VARCHAR	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR[#4]	N/A
NCHAR[#1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR[#4]	N/A
NVARCHAR[#1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR[#4]	N/A
MCHAR[#1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR[#4]	N/A
MVARCHAR[#1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR[#4]	N/A
DECIMAL[#2]	java.math.BigDecimal	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL[#4]	N/A
SMALLINT	java.lang.Short	short
INTEGER	java.lang.Integer	int
REAL, SMALLFLT	java.lang.Float	float
FLOAT, DOUBLE PRECISION	java.lang.Double	double
DATE	java.sql.Date	N/A
TIME	java.sql.Time	N/A

HiRDB data types	SQLJ data types (Java data types)	
	When a null value is included	When a null value is not included
TIMESTAMP	java.sql.Timestamp	N/A
INTERVAL HOUR TO SECOND	N/A	N/A
INTERVAL YEAR TO DAY	N/A	N/A
BLOB#3	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB#4	byte[]
BINARY	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY#4	byte[]

Legend:

N/A: Cannot be used or not applicable

Note

Repetition columns cannot be used.

#1: When `java.lang.String` is specified in the native interface edition, the data type requested to the server is `VARCHAR`. When the data type is specified in an output variable, the length of the data acceptance area is assumed to be 32,000 bytes.

#2: When `java.math.BigDecimal` is used as an output variable in the native interface edition, the precision is set to `15` and the scale to `0`.

#3: When the data type is specified with `byte[]` in the native interface edition, the data type requested to the server is `BINARY` type. If the HiRDB server is version 06-02 or earlier and the `BLOB` type is to be used, specify `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB`, which is a HiRDB data type. An error occurs if `byte[]` is specified.

#4: This type can be specified for the native interface edition.

19.3.5 Output variable settings (limited to the native interface edition)

When an execution request is sent to the server, the data length set in the SQL descriptor area for an output variable used in single-line searches, and in the `OUT` parameters of `CALL` statements when an execution request is sent to the server, differs depending on the initial value of the output variable. The following table lists the initial value and the data length set in the SQL descriptor area for each data type.

Table 19-5: Initial value for each data type and the data length set in SQL Descriptor Area

Data type	Initial value	Length of data set in SQL Descriptor Area
`HiRDBCHAR`	*variable* = `null;`	30,000 bytes
	variable = `new HiRDBCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 30,000)
	variable = `new HiRDBCHAR(String` *t*`);`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)
`HiRDBVARCHAR`	*variable* = `null;`	32,000 bytes
	variable = `new HiRDBVARCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 32,000)
	variable = `new HiRDBVARCHAR(String` *t*`)`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)
`HiRDBNCHAR`	*variable* = `null;`	30,000 bytes (15,000 double-byte characters)
	variable = `new HiRDBNCHAR(int` *n*`);`	(*n**2) bytes (*n* double-byte characters) (1 ≤ *n* ≤ 15,000)
	variable = `new HiRDBNCHAR(String` *t*`)`	Length of *t* (length of (`byte` array/2) obtained with `t.getBytes()`)
`HiRDBNVARCHAR`	*variable* = `null;`	32,000 bytes (16,000 double-byte characters)
	variable = `new HiRDBNVARCHAR(int` *n*`);`	(*n* x 2) bytes (*n* double-byte characters) (1 ≤ *n* ≤ 16,000)
	variable = `new HiRDBNVARCHAR(String` *t*`)`	Length of *t* (length of (`byte` array/2) obtained with `t.getBytes()`)
`HiRDBMCHAR`	*variable* = `null;`	30,000 bytes
	variable = `new HiRDBMCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 30,000)
	variable = `new HiRDBMCHAR(String` *t*`)`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)
`HiRDBMVARCHAR`	*variable* = `null;`	32,000 bytes
	variable = `new HiRDBMVARCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 32,000)
	variable = `new HiRDBMVARCHAR(String` *t*`)`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)

Data type	Initial value	Length of data set in SQL Descriptor Area
`HiRDBDECIMAL`	*variable* = `null;`	Precision 15, scale 0
	variable = `new HiRDBDECIMAL(int` *p*`,int` *s*`);`	Precision *p*, scale *s* (1 ≤ *p* ≤ 29, 0 ≤ *s* ≤ *p*)
	variable = `new HiRDBDECIMAL(String` *t*`)`	The precision is the character string length obtained when the sign and period characters are subtracted from *t*. The scale is the character string length after the period (excluding the period).
	variable = `newHiRDBDECIMAL(java.math.BigDecimal` *t*`)`	The precision is the character string length obtained when the flag and period characters of the character string retrieved with `toString()` are subtracted from *t*. The scale is the value retrieved by the `scale()` method of the `BigDecimal` object.
`HiRDBBLOB`	*variable* = `null;`	1 megabyte
	variable = `new HiRDBBLOB(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)
	variable = `new HiRDBBLOB(byte[]` *t*`)`	Length of *t*
`HiRDBBINARY`	*variable* = `null;`	1 megabyte
	variable = `new HiRDBBINARY(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)
	variable = `new HiRDBBINARY(byte[]` *t*`)`	Length of *t*
`Java.math.BigDecimal`	*variable* = `null;`	Precision 15, scale 0
	variable = `new java.math.BigDecimal;`	The precision is set to the character string length obtained when the flag and period characters in the character string retrieved by `toString()` are subtracted from the `BigDecimal` object. The scale is set to the value retrieved by the `scale()` method.
`byte[]`	*variable* = `null;`	1 megabyte
	variable = `new byte[int` *n*`]`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)

19.3.6 Using data types when a cursor is declared (limited to the native interface edition)

To use HiRDB data types with a cursor declared, you must specify the data types as shown in the following table.

Table 19-6: Descriptions and acceptance area settings for using the HiRDB data types when a cursor is declared

Data type	Description when cursor declared	Acceptance area setting
`HiRDBCHAR`	`#sql iterator` *cursor-name*`(HiRDBCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 30,000)
`HiRDBVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBVARCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 32,000)
`HiRDBNCHAR`	`#sql iterator` *cursor-name*`(HiRDBNCHAR(int n));`	(n x 2) bytes (*n* double-byte characters) (1 ≤ n ≤ 15,000)
`HiRDBNVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBNVARCHAR(int n));`	(*n* x 2) bytes (*n* double-byte characters) (1 ≤ *n* ≤ 16,000)
`HiRDBMCHAR`	`#sql iterator` *cursor-name*`(HiRDBMCHAR(int n));`	*n* bytes (1 ≤ n ≤ 30,000)
`HiRDBMVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBMVARCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 32,000)
`HiRDBDECIMAL`	`#sql iterator` *cursor-name*`(HiRDBMVARCHAR(int p,int s));`	Precision *p*, scale *s* (1 ≤ *p* ≤ 29, 0 ≤ *s* ≤ p)
`HiRDBBLOB`	`#sql iterator` *cursor-name*`(HiRDBBLOB(int n));`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)
`HiRDBBINARY`	`#sql iterator` *cursor-name*`(HiRDBBINARY(int n));`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)

19.3.7 Description of connection to and disconnection from a HiRDB server

SQLJ has no `CONNECT` or `DISCONNECT` statement. Therefore, connection to or disconnection from a HiRDB server is coded as Java instructions.

(1) Connection to a HiRDB server

To connect to a HiRDB server, use the following coding using a connection context.

(a) Defining a connection context class

Define a class for the connection context. *Class-name* indicates a Java identifier. The defined class inherits `sqlj.runtime.ConnectionContext.`

```
#sql modifier context class-name ;
```

(b) Declaring connection context

Using the declared class, declare the connection context (as a Java variable declaration). *Connection-context* indicates a Java identifier.

```
modifier class-name connection-context ;
```

(c) Connecting to a HiRDB server

Create a connection context object using a new operator. During this step, connection is made to the HiRDB server. For the connection parameters, describe the HiRDB server at the connection destination, port number, authorization identifier, and password in the same format as that used for JDBC.

```
connection-context = new class-name(connection-parameter) ;
```

(d) Connecting to the HiRDB server when the native interface is used

When the native interface is used, there are three ways of connecting to the HiRDB server:

- Describing the connection as a Java instruction
- Using the `CONNECT` statement
- Using the JDBC connection object (`Connection`)

These connection methods are described below.

1. Describing the connection as a Java instruction

Use the `new` operator to generate a connection context object. However, since JDBC is not being used, specify an authorization identifier, a password, a server name, and a port number in the connection parameters.

```
connection-context = new class-name(connection-parameters);
```

If no connection parameters are specified, the HiRDB server checks the client environment variables.

```
connection-context = new class-name();
```

An example of creating a connection context follows:

```
#sql context Ctx;
String Userid=new String("user1");
String Passwd=new String("puser1");
String Host=new String("HiRDB_SV");
short port=22000;

Ctx con = new Ctx(:Userid,:Passwd,:Host,:port);
```

2. Using the CONNECT statement

Specify an authorization identifier and a password in the connection parameters.

The HiRDB server checks the client environment definitions for the port number and the server name.

```
#sql [connection-context]{CONNECT USER :embedded variable USING :embedded variable};
or
#sql [connection-context]{CONNECT :embedded variable IDENTIFIED BY :embedded variable};
```

If connection parameters are not specified, the HiRDB server checks the client environment definitions.

```
#sql [connection-context]{CONNECT};
```

An example of the `CONNECT` statement follows:

```
#sql context Ctx;
String Userid=new String("user1");
String Passwd=new String("puser1");
Ctx con;

#sql [con] {CONNECT USER :Userid USING :Passwd };
```

3. Using the JDBC connection object (Connection)

Use the `new` operator to generate a connection context object. In the connection parameters, specify the JDBC connection object (`java.sql.Connection`).

```
connection-context = new class-name(connection-object);
```

An example of creating a connection context follows:

```
#sql context Ctx;
java sql.Connection con =

java.sql.DriverManager.getConnection("jdbc:hitachi:PrdbDrive://DBID=22200,
DBHOST=HiRDB_SV","user1","user1");

Ctx ctx = new Ctx(con);
```

(2) Disconnecting from a HiRDB server

To disconnect from the HiRDB server, invoke the `close` method for the connection context. Note that there is no reconnection method. To reconnect, create a new object.

```
connection-context.close() ;
```

An example of invoking the `close` method for the connection context follows:

```
#sql context DeptContext;
           ...
{
    DeptContext deptCtx = new DeptContext(deptURL,true);
    #sql [deptCtx] { DELETE FROM TAB };
    deptCtx.close();
}
```

When using the native interface edition, you can use the `DISCONNECT` statement instead of invoking the close method for the connection context.

```
#sql[connection-context]{DISCONNECT};
```

An example of the `DISCONNECT` statement follows:

```
#sql context Ctx;
Ctx con;
#sql[con]{CONNECT};

#sql[con]{DISCONNECT};
```

(3) Default connection

(a) Standard interface edition

For the standard interface edition, the default connection context is assumed if no connection context is specified in an SQL statement.

To use the default connection context, a UAP must create a connection context in advance, and set it as the default connection context. Once the default connection context is set, it remains valid until the close() method for the default connection context is issued or a new connection context is set as the default connection context.

The default connection context is held by a variable inside the default connection context class (JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext).

The default connection context has multiple constructors that have the different arguments described as follows.

- Constructor that has a JDBC connection object as an argument
- Constructor that has the URL of the connection destination, authorization identifier, password, and auto commit specification as arguments
- Constructor that has the specifications of the URL of the connection destination, Properties object, and autoCommit as arguments
- Constructor that has connection context as an argument

To specify the URL of the connection destination, authorization identifier, and password, use the same format as is used for the JDBC driver of HiRDB.

In SQLJ, to use a constructor that includes a connection URL during the creation of a connection context, you must specify autoCommit, and specify TRUE to enable it and FALSE to disable it.

If the default connection context is created from the JDBC connection context, the autoCommit setting in the JDBC connection context is inherited.

■ Creating and setting the default connection context

An example for creating and setting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                         ...
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
```

■ Releasing and resetting the default connection context

An example of releasing and resetting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                           ...
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
                ...
pctx.close();
PrdbContext new_pctx = new PrdbContext(url.use,passwd,autoCommit);

PrdbContext.setDefaultContext(new_pctx);
```

- ■ Acquiring the default connection context

  When the following method is invoked, the connection context can be acquired:

  `JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();`

  An example of specifying the default context follows:

```
void print_address(String name) throws SQLException;
{
  String telno;
  sqlj.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno
               FROM PERSON
               WHERE :name = NAME } ;
}
```

(b) Native interface edition

For the native interface edition, the default connection context class is held in a variable of `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext`.

The default connection context class has the following constructors:

- Constructors with JDBC connection objects as arguments
- Constructors with the authorization identifier, password, server name, and port number of the connection destination as arguments
- Constructors with the authorization identifier and password specification of the connection destination as arguments
- Constructors with connection contexts as arguments
- Constructors without arguments

- ■ Creating and setting the default connection context

  An example of creating and setting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                         :
PrdbContext pctx = new PrdbContext();
PrdbContext.setDefaultContext(pctx);
```

- ■ Releasing and resetting the default connection context

An example of releasing and resetting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                         :
PrdbContext pctx = new PrdbContext(user,passwd,host,port);
PrdbContext.setDefaultContext(pctx);
                         :
pctx.close();
PrdbContext new_pctx = new PrdbContext(user,passwd,host,port);

PrdbContext.setDefaultContext(new_pctx);
```

- Getting the default connection context

  To get the default connection context, invoke the following method:

  `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext.getDefaultContext();`

A coding example in which the default context is implicitly specified follows:

```
void print_address(String name) throws SQLException;
{
  String telno;
  JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno FROM PERSON WHERE :name = NAME } ;
}
```

19.3.8 Description of cursor-based retrieval

Because SQLJ has no `DECLARE CURSOR`, `OPEN`, or `CLOSE` statements, cursor declaration, opening, and closing must be coded as Java instructions. During this step, an iterator object is used in place of a cursor name. Because the iterator object is declared as a reference variable to an object, the same naming rule and valid range as in the Java rules apply here.

Depending on the iterator object type used, the retrieval result can be obtained using or not using a `FETCH` statement. A `FETCH` statement uses an object in the iterator type with a position specification and cannot use an object in the iterator type with a column name specification.

(1) Retrieval using a FETCH statement

The method for describing a retrieval using a `FETCH` statement is explained as follows.

(a) Defining a class for an iterator with position specification and declaring an iterator object

- Standard interface edition

For the standard interface edition, define a class for an iterator with a position specification and declare an iterator object. *class-name* indicates a Java identifier. *data-type-N* indicates the data type of a Java variable that stores the *N*-th retrieval item in the `FETCH` statement.

```
#sql modifier iterator class-name
              (data-type-1,data-type-2,...) ;
modifier class-name iterator-object ;
```

- Native interface edition

For the native interface edition, the specification is as follows:

```
#sql modifier iterator class-name
           [ implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate ]
           [ with keyword=setting-value,...]
              (data-type-1,data-type-2,...) ;
modifier class-name iterator-object ;
```

If an iterator is used in an `UPDATE` or `DELETE` statement, the `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate` interface is inherited.

keyword in the `WITH` clause indicates the function of the iterator. Only a constant can be specified. The following table shows the combinations of *keyword* in the `WITH` clause and the setting values.

Table 19-7: Combinations of keyword in the WITH clause and setting values

Keyword in the WITH clause	Function	Setting
`holdability`	Indicates a holdable cursor.	`TRUE`
`updateColumns`	Indicates the column to be updated.	`"`*column-name,column-name*`,..."`

(b) Defining and opening a cursor

Substitute the result set from the `SELECT` statement into the declared iterator object.

```
#sql [context] iterator-object = { SELECT-statement } ;
```

(c) Extracting the retrieval result

Specify an iterator object instead of a cursor name and execute a FETCH statement. The iterator object must be preceded by a colon.

```
#sql [context] {
   FETCH :iterator-object INTO :variable-1,:variable-2,...} ;
```

(d) Determining NOT FOUND

Invoke the endFetch method for the iterator object and determine whether the result is NOT FOUND. If there is no row to be retrieved, true is returned. If the next row is found, false is returned. If the endFetch method is invoked after the cursor is closed, true is returned.

```
while(! iterator-object.endFetch()) {
      processing-on-the-extracted-row
}
```

(e) Closing the cursor

To close the cursor, invoke the close method.

```
iterator-object.close() ;
```

An example of a retrieval using a FETCH statement follows:

```
#sql public iterator ByPos(String, int);
                  :
{
  ByPos positer;
  String name = null;
  int code = 0;

  #sql positer = { SELECT PNAME,PCODE FROM STOCK };
  #sql { FETCH :positer INTO :name,:code };
  while( !positer.endFetch() ){
    System.out.println(name + ":" + code);
    #sql { FETCH :positer INTO :name,:pcode };
  }
  positer.close();
}
```

(2) Retrieval without using a FETCH statement

Using the fields in the iterator with a column name specification, read out each column of the retrieval result.

(a) Defining a class for an iterator with column name specification

Define the same name (not case sensitive) as the retrieval item as the class field. For the data type, specify the data type of the Java variable that receives the retrieval result.

This class cannot be used for the native interface.

If the retrieval item is a value expression, a column name that includes a character that cannot be used in Java, for example, use an AS clause to define an alias for the retrieval item, and use that alias.

```
#sql modifier iterator class-name
          (data-type-1 column-name-1,
           data-type-2 column-name-2,...) ;
modifier class-name iterator-object;
```

(b) Defining and opening a cursor

Substitute the result set from the SELECT statement into the declared iterator object.

```
#sql [context] iterator-object = { SELECT-statement } ;
```

(c) Extracting the next row and determining NOT FOUND

Invoke the next method for the iterator object and determine whether the result is NOT FOUND. If the result is NOT FOUND, TRUE is returned. If a row is found, FALSE is returned. After the cursor is opened, it is not positioned on the first line of the retrieval result until the first next method is executed.

```
while(iterator-object.next()){
      processing-on-the-extracted-row
}
```

(d) Acquiring the retrieval result

Read data out from each field of the iterator object. If the result is NOT FOUND or if data is read out after the cursor is closed, the result is undetermined. If data is read out when the data type of the field is the Java basic data type and the retrieval result is a null value, the SQLNullException object occurs.

Data substituted into a field is not reflected in the database.

```
variable-1 = iterator-object.column-name-1 ;
variable-2 = iterator-object.column-name-2 ;
              ...
```

(e) Closing the cursor

To close the cursor, invoke the close method for the iterator object.

```
iterator-object.close() ;
```

An example of retrieval without using a FETCH statement follows:

Example:

```
#sql public iterator ByName(String pname,
                           int pcode);
                  :
{
   ByName nameiter;
   String s;
   int i;

   #sql nameiter = { SELECT PNAME, PCODE FROM STOCK };
   while( nameiter.next() ){
     s = nameiter.pname();
     i = nameiter.pcode();
     System.out.println(s + ":" + i);
   }
   nameiter.close();
}
```

(3) Updating using the cursor

For the native interface, a cursor can be used to update data.

To use an `UPDATE` or `DELETE` statement to manipulate the row on which the cursor is positioned, specify an iterator instead of a cursor name. Note that when the class for the iterator is being defined, it must inherit the `ForUpdate` interface.

```
#sql [context] { DELETE-statement WHERE CURRENT OF :iterator-object } ;

#sql [context] { UPDATE-statement WHERE CURRENT OF :iterator-object } ;
```

An example of an update that uses an iterator follows:

```
#sql public iterator ByPos
            implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate
            (String, int);
             :

{
  ByPos positer;
  String name = null;
  int year = 0;
  int newyear;

  #sql positer = { SELECT FULLNAME, BIRTHYEAR FROM PEOPLE };
  #sql { FETCH :positer INTO :name,:year };
  while( !positer.endFetch() ){
    newyear=year+10;
    #sql { UPDATE PEOPLE SET YEAR=:newyear WHERE CURRENT OF :positer; };
  }
  positer.close();
}
```

19.3.9 Receiving a dynamic result set

To receive a dynamic result set by invoking a procedure that returns a dynamic result set, use the `getNextResultSet()` method for the execution context. For the native interface edition, a procedure that returns a dynamic result set cannot be used because JDBC result sets cannot be used.

The `getNextResultSet()` method returns a dynamic result set (`ResultSet` object) as a return value. Every time this method is invoked, it returns the next result set. After it returns the last result set, it returns a null value.

For a procedure or SQL statement that does not return a dynamic result set, a null value is returned. A null value is returned also when SQL execution is not normally terminated.

If an error occurs during the execution of the `getNextResultSet()` method, the `SQLException` occurs.

An example follows:

```
#sql  [execCtx] { CALL MULTI_RESULTS() };
 ResultSet rs;
 while((rs == execCtx.getNextResultSet() ) != null){
   processing-of-the-retrieval-result;
   rs.close();
 }
```

19.3.10 Using JDBC and SQLJ together

This subsection explains how to use JDBC and SQLJ together.

(1) Acquiring a JDBC result set from an SQLJ iterator

You can convert an SQLJ iterator into a JDBC result set (`ResultSet` object) and use the JDBC API to obtain the retrieval result. For the native interface edition, JDBC result sets cannot be obtained.

To obtain a JDBC result set, use the `getResultSet` method for the iterator class (`ResultSetiterator`). This method returns a JDBC result set as a return value. After executing the `next` method for the iterator, do not invoke the `getResultSet` method.

After you have used the `getResultSet` method to convert an SQLJ iterator into a JDBC result set, do not receive a retrieval result using the original iterator.

An example follows:

```
public void showEmployeeName() throws SQLException
{
sqlj.runtime.ResultSetIterator iter;
#sql iter = { SELECT ename FROM rmp } ;
ResultSet rs = iter.getResultSet();
while(rs.next()){
System.out.println("employee name: " + rs.getString(1));
}
iter.close();
}
```

(2) Reading a JDBC result set as an iterator result set of SQLJ (limited to the standard interface edition)

The JDBC result set (`ResultSet`) that was created using the JDBC API is converted with the `CAST` statement and read as a result set of the SQLJ cursor.

The coding example follows:

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String query="SELECT pname, pcode FROM stock WHERE pcode > 1000";
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps ={CAST :rs };
```

(3) Converting JDBC connection into SQLJ connection context

The SQLJ connection context defines a constructor for generating an object from a JDBC connection. Using this constructor, you can convert a JDBC connection into an SQLJ connection context. Note that JDBC connection is transferred as an argument of the constructor. You can also use both types of connection together.

An example follows:

```
java.sql.Connection jdbcConCtx =java.sql.DriverManager.getConnection(...);
 #sql context Inventory;
Inventory sljConCtx = new Inventory(jdbcConCtx);
```

(4) Converting SQLJ connection into JDBC connection

You can use the `getConnection` method to get the JDBC connection from an SQLJ connection. You can also use both types of connection together.

With the native interface edition, an SQLJ connection cannot be converted into a JDBC connection. To use the same connection as JDBC, you must create a connection in JDBC beforehand, and then convert the connection into an SQLJ connection context.

An example follows:

```
#sql context Inventory;
Inventory sljConCtx = new Inventory(url);
java.sql.Connection jdbcConCtx = sqljConCtx.getConnection();
```

(5) Dynamic SQL statement

SQLJ can describe only static SQL statements. Therefore, to execute a dynamic SQL statement, you must use the JDBC API.

(a) Executing a dynamic SQL statement

A dynamic SQL statement is executed using a `PreparedStatement` object in JDBC.

When the `prepareStatement` method for the connection context is executed using the SQL as an argument, a `PreparedStatement` object is returned as a return value.

To set a parameter in a dynamic SQL statement, use the `set` method of `PreparedStatement`. To execute the dynamic SQL statement, use the `execute` method of the `PreparedStatement` object.

An example of dynamic SQL execution follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO FOO_TABLE VALUES(?, ?)");
pstmt.setInt(1, 100);
pstmt.setString(2, "test");
pstmt.execute();
```

(b) Retrieving a dynamic cursor

Only static cursors can be used in SQLJ. Therefore, to use a dynamic cursor, you must use the JDBC API.

When the `prepareStatement` method for the connection context is executed for a character string that indicates a `SELECT` statement, a `PreparedStatement` object is returned as a return value.

To set a parameter, use the `set` method of `PreparedStatement`. To execute the SQL statement, use the `executeQuery` method of the `PreparedStatement` object. The `executeQuery` method returns the JDBC result set.

To receive a retrieval result, use the `get` method for result sets.

An example of retrieval using a dynamic cursor follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
          "SELECT NAME, POINT FROM FOO_TABLE WHERE BAR=100");
ResultSet rs = pstmt.executeQuery();
String name;
Integer point;
rs.next();
name = pstmt.getString(1);
point = pstmt.getInteger(2);
```

(c) Executing a DESCRIBE statement

To determine the column name and data type of each retrieval item of a dynamic cursor, use a `ResultSetMetaData` object. You can obtain a `ResultSetMetaData` object from the `getMetaData` object of the result set.

You can also use the `getColumnClassName` method of the `ResultSetMetaData` object to obtain the character string that indicates the data type of each retrieval item.

You can use the `getColumnName` method to obtain column names.

Specify the items to be retrieved using numbers (beginning with `1`). You can use the `getColumnCount` method to obtain the number of columns.

An example of executing the `DESCRIBE` statement follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
     "SELECT * FROM FOO_TABLE");
java.sql.ResultSetMetaData aMeta = pstmt.getMetaData();
int columCount = aMeta.getColumnCount();
Vector nameList = new Vector();
Vector classLis = new Vector();
for(int i = 1; i <= columnCount; i++){
     nameList.addElement(aMeta.getColumnName(i));
     classList.addElement(a.Meta.getColumnClassName(i));
}
Vector dataList = new Vector();
rs.next();
for(int i = 1; i <= columnCount; i++){
     dataList.addElement(rs.getObject(i));
}
```

19.3.11 Creating and executing a UAP

(1) Executing the SQLJ translator

1.Set environment variables.

When the HiRDB client is the UNIX IPF edition:

Set environment variables as shown below. The underlined portion is the default installation directory.

- For HiRDB/Developer's Kit

`CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdsqlj.jar`#1

- For HiRDB/Run Time

  `CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdruntime.jar`#2

  `CLASSPATH=$CLASSPATH:/HiRDB/client/lib/pdnativert.jar`#3

  #1: For the 32-bit mode HP-UX (IPF) edition, this setting becomes `pdsqlj32.jar`.

  #2: For the 32-bit mode HP-UX (IPF) edition, this setting becomes `pdruntime32.jar`.

  #3: For the 32-bit mode HP-UX (IPF) edition, this setting becomes `pdnativert32.jar`.

When the HiRDB client is Windows:

Choose **Control Panel**, **System, System Properties**, and **Environment** in that order, and then specify as shown below. The underlined portion is the default installation directory:

- For HiRDB/Developer's Kit

  `CLASSPATH=%CLASSPATH%:\HiRDB\UTL\pdsqlj.jar`

- For HiRDB/Run Time

  `CLASSPATH=%CLASSPATH%\HiRDB\UTL\pdruntime.jar`

  `CLASSPATH=%CLASSPATH%\HiRDB\UTL\pdnativert.jar`

2. Executing the SQLJ Translator

The SQLJ Translator runs on a Java virtual machine.

Format

```
pdjava [option] file-name-1.sqlj [file-name-2.java]
```

Description

option

The table below lists the SQLJ Translator options.

file-name-1

This is a UAP source file that describes SQLJ.

file-name-2

This is a post-source file.

file-name-1 and *file-name-2* may contain a path. If *file-name-2*`.java` is not specified, *file-name-1*`.java` is assumed.

Table 19-8: SQLJ Translator options

Options	Coding format	Explanation
`-dir`	`-dir=`*directory-name*	Specifies the direction in which to create the post-source file.
`-d`	`-d=`*directory-name*	
`-status`	`-status`	Displays the internal status for preprocessing. This is a debugging option.
`-J`	`-J-`*option*	Specifies a Java virtual machine option to be used during the execution of the SQLJ Translator.
`-version`	`-version`	Displays the version of the SQLJ translator. No translation is performed.
`-help`	`-help`	Specified to display an option explanation. No translation is performed.
`-native`	`-native`	Generates a post source for the native interface. If you are specifying multiple options, be sure to specify this option first.
`-d 64`	`-d 64`	Specifies that the SQLJ translator is to be executed with the 64-bit mode HP-UX (IPF) edition.

Notes

1. When specifying multiple options, use spaces to separate the options.

   Up to two options can be specified for the standard interface edition, and up to three options (including `-native`) for the native interface edition. If more options are specified, an error occurs.

2. The `-native` option for using the native interface edition must be specified first. If the `-native` option is not specified first, an error occurs.

3. If the `-help` or `-version` option is specified, the other options are ignored. However, it both `-help` and `-version` are specified at the same time, both are valid.

Execution example

Execution examples are shown below.

- For the standard interface edition

```
Example 1: pdjava file-name.sqlj
Example 2: pdjava -dir=d:\sqljsrc file-name.sqlj
Example 3: pdjava -d64 file-name.sqlj#
```

#: This example is for the 64-bit mode HP-UX (IPF) edition.

- For the native interface edition

```
Example 1: pdjava -native file name.sqlj
Example 2: pdjava -native -dir=d:\sqljsrc file name.sqlj
Example 3: pdjava -native -d64 file name.sqlj#
```

#: This example is for the 64-bit mode HP-UX (IPF) edition.

(2) Compiling and executing an UAP

1. Setting environment variables

   See step 1 in *(1) Executing the SQLJ translator*.

2. Compiling the post-source file

   Use the Java compiler to compile the post-source file generated by the SQLJ Translator. The format used for compilation follows:

```
javac file-name-2.java
```

3. Setting the path to the JDBC driver in CLASSPATH

   For details on setting up a path for the JDBC driver, see *17.1 Installation and environment setup*.

4. Using DriveManager to connect to a database

   For details about database connection using `DriverManager`, see *17.2.1 Driver class*.

5. Using the Java Virtual Machine to execute the CLASS file

   Use the Java Virtual Machine to execute the `Class` file. The execution format follows:

```
java file-name-2
```

When the 32-bit mode HP-UX (IPF) edition is used, the execution format is as follows:

```
java -d64 file-name-2
```

19.3.12 Migrating an SQLJ source from the standard interface edition to the native interface edition

Some portions must be revised to migrate an SQLJ source from the standard interface edition to the native interface edition. The following table shows where revisions are required to migrate to the native interface edition.

Table 19-9: Revisions required for an SQLJ source to be migrated from the standard interface edition to the native interface edition

Command name	Standard interface edition	Native interface edition	Revision needed?
UAP (input) source	`*file-name*.sqlj`	`*file-name*.sqlj`	N
UAP (output) source	JAVA*source-file-name*. java*profile-name*.ser	JAVA*source-file*.java	N
Option	Specification of output file name, others	Specification of output file name, others	N
SQL prefix	#sql	#sql	N
SQL terminator	;	;	N
SQL declare section	Unnecessary	Unnecessary	N
Embedded variable	:*variable-name*	:*variable-name*	N
Declaration statement	#sql context *class-name* #sql iterator *class-name*	#sql context *class-name* #sql iterator *class-name*[#1]	N[#2]
Connection context creation	A JDBC connection object can be specified in a parameter.	A JDBC connection object can be specified in a parameter.	N
	An object other than a JDBC connection object can be specified in a parameter.	There is no object that obtains the same parameter.	Y[#3]
Use of default connection context	JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext	Y[#4]
Explicit specification of execution context	sqlj.runtime.ExecutionContext	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ExecutionContext	Y[#5]

Command name	Standard interface edition	Native interface edition	Revision needed?
Use of the CAST statement (acceptance of a JDBC result set)	Can be executed.	Cannot be executed.	Y[#6]
Acceptance of dynamic result set	Can be executed.	Cannot be executed.	Y[#7]
Data type	byte[] java.math.BigDecimal java.lang.String	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR and others	Y[#8]
Execution of different SELECT statements that use the same iterator object name	Can be executed.	Cannot be executed.	Y[#9]

Legend:

Y: Need for revision.

N: No need for revision.

#1: A name iterator cannot be used. A position iterator can be used but not in an inner class.

#2: Revision becomes necessary when a name iterator or an inner class is used.

#3: Change the connection process. For details, see *19.3.7(1)(d) Connecting to the HiRDB server when the native interface is used.*

#4: Change the JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext package name to JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext.

#5: Change sqlj.runtime.ExecutionContext to JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ExecutionContext.

#6: CAST statements result in errors when translated. Therefore, delete all CAST statements. Modify the UAP so that it operates the JDBC result set directly and does not use an SQLJ iterator.

#7: Since there is no method that accepts a dynamic result set, an error occurs during Java compilation. Therefore, delete the section that issues

ExecutionContext.getNextResultSet(). To get the dynamic result set, change the UAP so that is uses JDBC directly.

#8: byte[] is requested to the HiRDB server as the BINARY type. Modification is necessary if the HIRDB server is version 06-02 or earlier.

When the BigDecimal type is specified in an acceptance variable, the precision in set to 15 and the scale to 0. Therefore, if any other precision or scale value is set, the value must be changed.

When String is specified in an input variable, it is requested to the HiRDB server as the VARCHAR type. If you want to associate the data type with a data type of the HiRDB server, you must change the data type.

#9: The same iterator object name cannot be used to execute different SELECT statements. In this case, a separate iterator object name must be specified for each SELECT statement.

```
#sql iterator pos(HiRDBCHAR(10));
   :
pos positer = null
pos positer2 = null;
HiRDBCHAR out = null;
   :
#sql positer = {SELECT * FROM T1};
#sql {FETCH :positer INTO :out}
positer.close();

#sql positer2 = {SELECT * FROM T2};
#sql {FETCH :positer2 INTO :out}
positer2.close();
```

19.3.13 Notes about UAP development

When developing a UAP that uses multiple threads, do not use the default connection text as the connection context. If multiple threads use the same connection context, an error occurs.

When using multiple threads, be sure to specify the connection context explicitly. An example in which the connection context is specified explicitly follows:

```
#sql context Ctx;

public class sample{
 public void main(String args[]){
   Ctx con = null;
   #sql [con] {CONNECT};                      //Explicit
specification of connection context
   ...
   int data = 100;
   #sql [con] {INSERT INTO T1 VALUES(:data)}; //Explicit
specification of connection context

   #sql [con] {DISCONNECT};                    //Explicit
specification of connection context
 }
}
```

When using the SQLJ native interface edition, match the number of `SELECT` statement retrieval items with the number of columns of the iterator object to be used. If the two do not match, false errors may occur.

19.4 Native Runtime

The SQLJ runtime library used by the native interface is called *Native Runtime*.

Native Runtime provides the following functions:

- Classes and interfaces used in compilation when the `-native` option is specified
- Access to HiRDB

19.4.1 Package configuration

The following table shows the configuration of the Native Runtime packages.

Table 19-10: Configuration of the Native Runtime packages

Package name	Collected contents
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	Classes and interfaces
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.error`	Error class

19.4.2 Public classes of Native Runtime

The following table lists the public classes of Native Runtime.

Table 19-11: Public classes of Native Runtime

Package	Class or interface name	Function
N/A	Connection context	This class is generated by `#sql context` *class-name*`;` of the SQLJ translator. This corresponds to a connection context of SQLJ.
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	`PrdbContext`	This is the default connection context. This corresponds to the default connection context of SQLJ.
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	`ExecutionContext`	This is an execution context. This class corresponds to an execution context of SQLJ and is used in managing SQL execution.
N/A	Iterator	This class is generated by `#sql iterator` *class-name*`;` of the SQLJ translator. This corresponds to an iterator of SQLJ.

Package	Class or interface name	Function
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	RTResultSet	This is a result set object. This class corresponds to a result set of JDBC and is used in managing results.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	ForUpdate	This interface is implemented by an iterator declaration when cursor update using an iterator is used.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBCHAR	Indicates the CHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBVARCHAR	Indicates the VARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNCHAR	Indicates the NCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNVARCHAR	Indicates the NVARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMCHAR	Indicates the MCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMVARCHAR	Indicates the MVARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBDECIMAL	Indicates the DECIMAL type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBLOB	Indicates the BLOB type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBINARY	Indicates the BINARY type of HiRDB.

Legend:

N/A: No package is available.

19.4.3 Cluster specifications

This section describes the method and field values of each class.

(1) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR class

Description

This class corresponds to the CHAR type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBCHAR`	`HiRDBCHAR(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBCHAR` class.
		If the length of the specified character string is 30,001 bytes or greater, `SQLException` is thrown.

`HiRDBCHAR`	`HiRDBCHAR(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBCHAR` class that has a length of *len*.
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 30,000, `SQLException` is thrown.

Methods

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(2) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR class

Description

This class corresponds to the `VARCHAR` type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBVARCHAR`	`HiRDBVARCHAR(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBVARCHAR` class.
		If the length of the specified character string is 32,001 bytes or greater, `SQLException` is thrown.

`HiRDBVARCHAR`	`HiRDBVARCHAR(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBVARCHAR` class that has a length of *len*.
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 32,000, `SQLException` is thrown.

Methods

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(3) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR class

Description

This class corresponds to the `NCHAR` type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBNCHAR`	`HiRDBNCHAR(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBNCHAR` class.
		If the length of the specified character string is 15,001 bytes or greater, `SQLException` is thrown.

`HiRDBNCHAR`	`HiRDBNCHAR(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBNCHAR` class that has a length of *len* (*len* is the number of double-byte characters).
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (double-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 15,000, `SQLException` is thrown.

Methods

Return value	Methods	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(4) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR class

Description

This class corresponds to the `NVARCHAR` type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBNVARCHAR`	`HiRDBNVARCHAR(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBNVARCHAR` class.
		If the length of the specified character string is 16,001 characters or greater, `SQLException` is thrown.

Return value	Method	Function description
HiRDBNVARCHAR	HiRDBNVARCHAR(int *len*) throws SQLException	Returns a HiRDBNVARCHAR class that has a length of *len* (*len* is the number of double-byte characters). This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (double-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 16,000, SQLException is thrown.

Methods

Return value	Method	Function description
String	getString()	Returns the String object.
int	length()	Returns the character string length.

(5) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR class

Description

This class corresponds to the MCHAR type of HiRDB.

Constructors

Return value	Method	Function description
HiRDBMCHAR	HiRDBMCHAR(String *s*) throws SQLException	Generates a new HiRDBMCHAR class. If the length of the specified character string is 30,001 bytes or greater, SQLException is thrown.
HiRDBMCHAR	HiRDBMCHAR(int *len*) throws SQLException	Returns a HiRDBMCHAR class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 30,000, SQLException is thrown.

Methods

Return value	Method	Function description
String	getString()	Returns the String object.
int	length()	Returns the character string length.

(6) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR class

Description

This class corresponds to the `MVARCHAR` type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBMVARCHAR`	`HiRDBMVARCHAR(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBMVARCHAR` class.
		If the length of the specified character string is 32,001 bytes or greater, `SQLException` is thrown.

`HiRDBMVARCHAR`	`HiRDBMVARCHAR(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBMVARCHAR` class that has a length of `len`.
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character x *len*) was specified. If the specified *len* value is not in the range from 1 to 32,000, `SQLException` is thrown.

Method

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(7) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB class

Description

This class corresponds to the `BLOB` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBLOB`	`HiRDBBLOB(byte[]` *b*`)`	Generates a new `HiRDBBLOB` class.

`HiRDBBLOB`	`HiRDBBLOB(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBBLOB` class that has a length of *len*.
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that the number `(0(0x30)` x *len*) was specified. If the specified *len* value is 0 or less, `SQLException` is thrown.

Methods

Return value	Method	Function description
`byte[]`	`getBytes[]`	Returns `byte[]`.
`int`	`length()`	Returns the `byte[]` length.

(8) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY class

Description

This class corresponds to the `BINARY` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBBINARY`	`HiRDBBINARY(byte[]` *b*`)`	Generates a new `HiRDBBINARY` class.

`HiRDBBINARY`	`HiRDBBINARY(int` *len*`)`
	`throws SQLException`
		Returns a `HiRDBBINARY` class that has a length of *len*.
		This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that the number `(0(0x30)` x *len*) was specified. If the specified *len* value is 0 or less, `SQLException` is thrown.

Methods

Return value	Method	Function description
`byte[]`	`getBytes()`	Returns `byte[]`.
`int`	`length()`	Returns the `byte[]` length.

(9) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL class

Description

This class corresponds to the `DECIMAL` type of HiRDB.

Constructors

Return value	Method	Function description

`HiRDBDECIMAL`	`HiRDBDECIMAL(String` *s*`)`
	`throws SQLException`
		Generates a new `HiRDBDECIMAL` class.
		If the character string of the argument contains a character string other than a number, a period, and a sign or if the precision and scale values obtained from the character string are 30 or higher, `SQLException` occurs.

Return value	Method	Function description
`HiRDBDECIMAL`	`HiRDBDECIMAL (java.math.BigDecimal) throws SQLException`	Generates a new `HiRDBDECIMAL` class. If the precision and scale values in the arguments are 30 or higher, `SQLException` occurs.
`HiRDBDECIMAL`	`HiRDBDECIMAL(int` *x*`,int` *y*`) throws SQLException`	Returns a `HiRDBDECIMAL` class with precision *x* and scale *y*. This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that `0` was specified. If *x* is not in the range from 1 to 29, *y* is not in the range from 0 to 29, and *x* is less than *y*, `SQLException` occurs.

Methods

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`java.math.BigDecimal`	`getBigDecimal()`	Returns the `java.math.BigDecimal` object.
`int`	`precision()`	Returns the precision.
`int`	`scale()`	Returns the scale.

19.4.4 Coding examples using the native interface

(1) Data insertion and retrieval

A coding example (`sample1.sqlj`) of data insertion and retrieval follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
//Declare iterator (cursor)
#sql iterator Pos(int,HiRDBCHAR(10),HiRDBNCHAR(5),HiRDBDECIMAL(10,5));

public class sample1{
  public static void main(String args[]){

    //Connect and create table
    try{
      #sql{CONNECT};     //Refer to client environment variable and connect
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};
    //Insert data
    try{
      int InInt = 100;
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("NCHAR");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());};


    //Retrieve data (FETCH)
    try{
      Pos sampleCur = null;
      int OutInt = 0;
      HiRDBCHAR OutChar = null;
      HiRDBNCHAR OutNchar = null;
      HiRDBDECIMAL OutDecimal = null;

      #sql sampleCur  = {SELECT * FROM SAMPLE1};
      while(true){
        #sql {FETCH :sampleCur INTO :OutInt ,:OutChar ,:OutNchar ,:OutDecimal };
        if(sampleCur.endFetch()) break;
          System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
             " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
      }
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(2) Data insertion and single-row retrieval

A coding example (`sample2.sqlj`) of data insertion and single-row retrieval follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
//Iterator (cursor) declaration
#sql iterator Pos(int,HiRDBCHAR(10),HiRDBNCHAR(5),HiRDBDECIMAL(10,5));

public class sample1{
  public static void main(String args[]){

    //Connection and table creation
    try{
      #sql{CONNECT};      //Refer to the client environment variables and connect.
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};
    //Insert data
    try{
      int InInt = 100;
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("NCHAR");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());};


    //Retrieve data (single-row retrieval)
    try{
      //Declare output variables
      int OutInt = 0;
      HiRDBCHAR OutChar = new HiRDBCHAR(10);
      HiRDBNCHAR OutNchar = new HiRDBNCHAR(5);
      HiRDBDECIMAL OutDecimal = new HiRDBDECIMAL(10,5);

      #sql {SELECT * INTO :OutInt,:OutChar,:OutNchar,:OutDecimal FROM SAMPLE1};
      System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
        " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(3) CALL statement execution

A coding example (`sample3.sqlj`) of `CALL` statement execution follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;

public class sample3{
  public static void main(String args[]){

    Integer PInteger1 = new Integer(99);
    Integer PInteger2 = new Integer(100);
    Integer PInteger3 = new Integer(101);try{
      #sql {CONNECT};
    }catch(SQLException e){System.out.println(e.getMessage());}
    try{
      #sql {DROP PROCEDURE PROCSQLJ};
      #sql {DROP TABLE PROCTABLE};
    }catch(SQLException e1){}

    try{
      #sql {CREATE TABLE PROCTABLE(c1 int, c2 int)};
      #sql {CREATE PROCEDURE PROC1(in p1 int,out p2 int,inout p3 int)
        begin
        insert into PROCTABLE values(p1,p3);
        select * into p2,p3 from PROCTABLE;
        end};
      #sql {COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());}


    try{
      #sql {CALL PROC1(in :PInteger1 ,out :PInteger2 ,inout :PInteger3 )};
    }catch(SQLException e){System.out.println(e.getMessage());}

    System.out.println("IN parameter PInteger1 = " + PInteger1 );
    System.out.println("OUT parameter PInteger2 = " + PInteger2 );
    System.out.println("INOUT parameter PInteger3 = " + PInteger3 );

    try{#sql {DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(4) Update using a cursor

A coding example (`sample4.sqlj`) of update using a cursor follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
#sql iterator iterP implements
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate(short);

public class sample4{
  public static void main(String args[]){
    iterP positer = null;
    iterP positer2 = null;
    short indata;
    short indata2 = 0;
    short indata3 = 999;
    try{
      #sql {CONNECT};
      #sql {DROP TABLE CURTABLE};
    }catch(SQLException e){System.out.println(e.getMessage());}
    //Create table
    try{#sql {CREATE TABLE CURTABLE(c1 smallint)};
    }catch(SQLException e){System.out.println(e.getMessage());}
    //Insert data
    for(short i = 0;i < 5;i++){
      indata = i;
      try{#sql{INSERT INTO CURTABLE VALUES(:indata)};}catch(SQLException e){}
    }

    //Execute SELECT and update using cursor
    try{
      #sql positer = {SELECT * FROM CURTABLE};
    }catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer INTO :indata2};
        if(positer.endFetch()) break;
        System.out.println(indata2);
        #sql { UPDATE CURTABLE SET C1=:indata3 WHERE CURRENT OF :positer };
      }
    }catch(SQLException e){e.getMessage();}


    //Check update results
    try{#sql positer2 = {SELECT * FROM CURTABLE};}catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer2 INTO :indata2};
        if(positer2.endFetch()) break;
        System.out.println(indata2);
      }
    }catch(SQLException e){System.out.println(e.getMessage());}
    try{#sql{DISCONNECT};}catch(SQLException e){}
  }
}
```

Appendixes

A. SQL Communications Area
B. SQL Descriptor Area
C. Column Name Descriptor Area
D. Type Name Descriptor Area
E. Character Set Descriptor Area
F. SQL Data Types and Data Descriptions
G. Data Dictionary Table Retrieval
H. Functions provided by HiRDB
I. Scalar Functions That Can Be Specified in the Escape Clause
J. Character Code Conversion Rules When Character Sets Are Used
K. HiRDB SQL Tuning Advisor Environment Setup
L. Maximum and Minimum HiRDB Values
M. List of Sample UAPs

A. SQL Communications Area

When SQL statements are executed, HiRDB sends a return code and related information to the UAP indicating whether or not the SQL statements executed normally. The area that receives this information is called the *SQL Communications Area*. This appendix explains the organization and contents of the SQL Communications Area and the expansion of the area.

For details about the use of the SQL Communications Area, see *3.6 SQL error identification and corrective measures*.

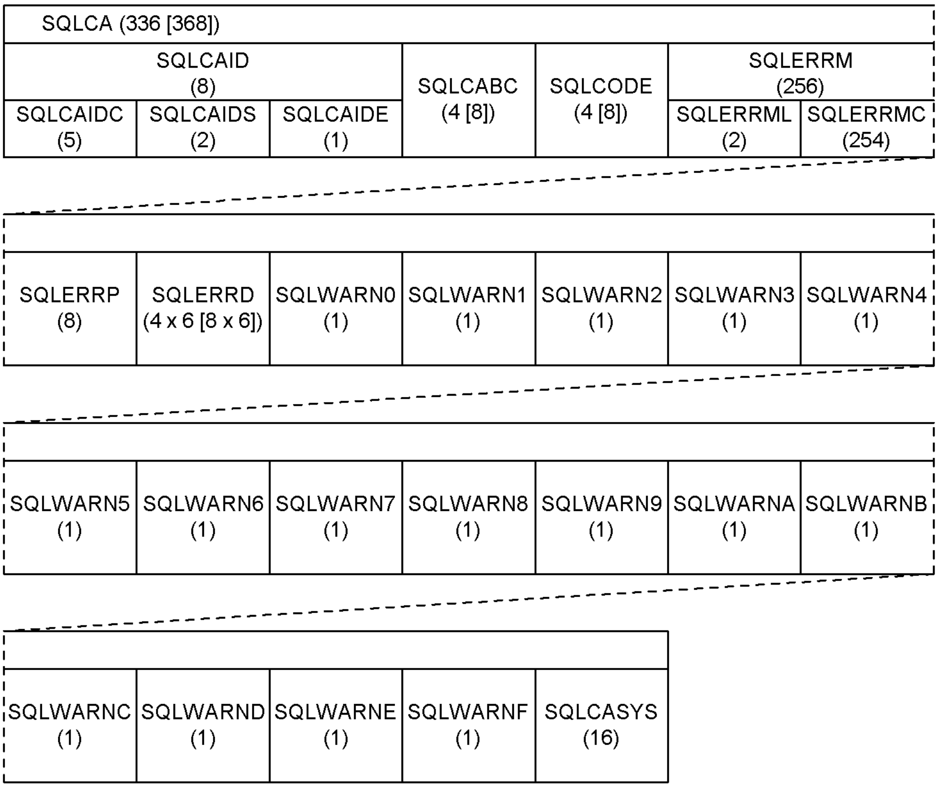
A.1 Organization and contents of the SQL Communications Area

The organization and contents of the area that receives SQL execution information are explained as follows.

(1) Organization of the SQL Communications Area

The following figure shows the organization of the SQL Communications Area.

Figure A-1: Configuration of SQL Communications Area



Note

1. Numbers in parentheses indicate length (in bytes).
2. Brackets [ ] in parentheses enclose a value for 64-bit mode. For 64-bit mode Windows, SQLCA is 336 bytes.
3. In 64-bit mode, the length of `SQLCABC`, `SQLCODE`, and `SQLERRD` becomes the size of the `long` type for each platform.

(2) Contents of the SQL Communications Area

The following table shows the contents of the SQL Communications Area.

Table A-1: Contents of the SQL Communications Area

Level number[#1]	Communications area name	Data type	Length (bytes)	Description
1	SQLCA	--	336 [368]	Denotes the overall SQL Communications Area.
2	SQLCAID	--	8	Denotes the SQLCAIDC, SQLCAIDS, and SQLCAIDE areas.
3	SQLCAIDC	char	5	Contains a character string (SQLCA) indicating that the area is the SQL Communications Area.
3	SQLCAIDS	char	2	Used by HiRDB.
3	SQLCAIDE	char	1	Used by HiRDB.
2	SQLCABC	long	4 [8][#3]	Sets the size (336 [368] bytes) of the SQL Communications Area.
2	SQLCODE	long	4 [8][#3]	Receives one of the following return codes from HiRDB after SQL statements have been executed: Negative: Abnormal termination 0: Normal termination Positive: Normal termination with a message For details about the messages associated with return codes, see the manual *HiRDB Version 9 Messages*. Return codes associated with messages are retrieved as follows: Return code Associated message ID -*yyy* KFPA11*yyy* -1*yyy* KFPA19*yyy* -3*yyy* KFPA18*yyy* *yyy* KFPA12*yyy* 3*yyy* KFPA13*yyy* Examples: Return code Message ID -125 KFPA11125 -1200 KFPA19200 -3200 KFPA18200 100 KFPA12100 3010 KFPA13010

Level number#1	Communications area name	Data type	Length (bytes)	Description
2	SQLERRM	--	256	Denotes the SQLERRML and SQLERRMC areas. The contents of these areas vary depending on whether the return code returned to the SQLCODE area is positive or negative: • If the return code is negative, a character string indicating the location or the cause of the error can be returned • If the return code is positive, a character string indicating message information can be returned.
3	SQLERRML	short	2	Contains the length of the message returned to the SQLERRMC area.
3	SQLERRMC	char	254	Contains the message associated with the return code returned to the SQLCODE area; for the contents of this area, see the manual *HiRDB Version 9 Messages*.
2	SQLERRP	char	8	Used by HiRDB.
2	SQLERRD	long	4#3 x 6	Contains the internal status of HiRDB. This area is an array of six areas of the long data type: SQLERRD[0]: Not used SQLERRD[1]: Not used SQLERRD[2]: When SQLCODE is 0 or a positive value: One of the following values. • Number of rows retrieved by the SELECT statement • Number of rows updated by the UPDATE statement • Number of rows deleted by the DELETE statement • Number of rows inserted by the INSERT statement • Number of rows fetched by the FETCH statement • Number of rows created by the ASSIGN LIST statement When SQLCODE is a negative value: A value is returned indicating the array element that resulted in an error during update processing using the array.#4 SQLERRD[3]: Used by the system. SQLERRD[4]: Used by the system. SQLERRD[5]: Not used

Level number[#] 1	Communications area name	Data type	Length (bytes)	Description
2	SQLWARN0	char	1	W is set in this area when a warning flag (W) is set in any of the areas SQLWARN1-SQLWARNF.
2	SQLWARN1	char	1	W is set in this area if an embedded variable for receiving data during character data retrieval was shorter than the data, and the truncated value was received. W is also set if the embedded variable for receiving data during repetition retrieval had a smaller element count than the data and values of the discarded elements that were received; otherwise, this area is empty.
2	SQLWARN2	char	1	W is set in this area if the null value was ignored in set function processing; otherwise, this area is empty. However, in either of the following cases, a space may be set in this area even if the null value was ignored during set function processing: • When a table that defines an index that recognizes a null value as an exception value is retrieved. • When the rapid grouping facility is used.
2	SQLWARN3	char	1	W is set in this area if the number of columns containing the results of a retrieval did not match the number of embedded variables that received the results of the retrieval; otherwise, this area is empty.
2	SQLWARN4	char	1	W is set in this area if an UPDATE or DELETE statement without a WHERE clause was executed; otherwise, this area is empty.
2	SQLWARN5	char	1	Spare
2	SQLWARN6	char	1	W is set in this area if the transaction was cancelled implicitly; otherwise, this area is empty.
2	SQLWARN7	char	1	W is set in this area if a repetition column with subscripts is specified in the SET or DELETE clause of the UPDATE statement, and the update is ignored because there are no elements in the row to be updated; otherwise, this area is empty.
2	SQLWARN8	char	1	Spare
2	SQLWARN9	char	1	Spare

Level number[#1]	Communications area name	Data type	Length (bytes)	Description
2	SQLWARNA[#2]	char	1	W is set in this area if an invalid date occurred as a result of a date operation and HiRDB modified the date automatically to the last day of the affected month; otherwise, this area is empty.
2	SQLWARNB[#2]	char	1	W is set in this area if either an overflow error or division by zero error occurred in a computation during SQL statements execution and the result of the computation was set as a null value. Otherwise, this area is empty.
2	SQLWARNC[#2]	char	1	W is set in this area when the value for a day in a date interval is more than two digits after a date operation has been completed; otherwise, this area is empty.
2	SQLWARND	char	1	Used by HiRDB.
2	SQLWARNE	char	1	Spare
2	SQLWARNF	char	1	Spare
2	SQLCASYS	char	16	Used by HiRDB.

--: Not Applicable.

Note

Value in brackets [ ] indicates the length for 64-bit mode. For 64-bite mode Windows, SQLCA is 336 bytes.

#1:

Level numbers indicate the set inclusion relationships of the SQL Communications Area. The level 1 Communications Area is composed of level 2 Communications Areas.

#2:

The first FETCH statement returns W when an SQL statement containing sort processing or an SQL statement containing the EXISTS predicate is executed.

In a HiRDB/Parallel Server environment, the row that returns W cannot be determined if a warning is generated at the WHERE clause.

#3:

In 64-bit mode, the length is the size of the long type for each platform.

#4

Such a value is set if YES is specified in the PDARYERRPOS client environment variable.

A.2 Expanding the SQL Communications Area

The SQL Communications Area need not be described in the UAP, because it is expanded by the SQL preprocessor in the source program written in a high-level language.

The format of the SQL Communications Area expanded by the SQL preprocessor in a source program is shown as follows.

(1) C

This example shows SQL Communications Area expansion when C language is used.

```
#define  SQLCAIDE sqlca.sqlcaide
#define  SQLCODE  sqlca.sqlcode
#define  SQLERRML sqlca.sqlerrml
#define  SQLERRMC sqlca.sqlerrmc
#define  SQLERRMD sqlca.sqlerrmd
#define  SQLERRD0 sqlca.sqlerrd[0]
#define  SQLERRD1 sqlca.sqlerrd[1]
#define  SQLERRD2 sqlca.sqlerrd[2]
#define  SQLERRD3 sqlca.sqlerrd[3]
#define  SQLERRD4 sqlca.sqlerrd[4]
#define  SQLERRD5 sqlca.sqlerrd[5]
#define  SQLWARN0 sqlca.sqlwarn0
#define  SQLWARN1 sqlca.sqlwarn1
#define  SQLWARN2 sqlca.sqlwarn2
#define  SQLWARN3 sqlca.sqlwarn3
#define  SQLWARN4 sqlca.sqlwarn4
#define  SQLWARN5 sqlca.sqlwarn5
#define  SQLWARN6 sqlca.sqlwarn6
#define  SQLWARN7 sqlca.sqlwarn7
#define  SQLWARN8 sqlca.sqlwarn8
#define  SQLWARN9 sqlca.sqlwarn9
#define  SQLWARNA sqlca.sqlwarna
#define  SQLWARNB sqlca.sqlwarnb
#define  SQLWARNC sqlca.sqlwarnc
#define  SQLWARND sqlca.sqlwarnd
#define  SQLWARNE sqlca.sqlwarne
#define  SQLWARNF sqlca.sqlwarnf
typedef struct sqlca {
  char     sqlcaidc[5];    /* Table ID                    */
  char     sqlcaids[2];    /* Used by HiRDB               */
  char     sqlcaide;       /* Used by HiRDB               */
  long     sqlcabc;        /* SQLCA area length           */
  long     sqlcode;        /* SQLCODE                     */
  short     sqlerrml;      /* Effective message length                  */
  char      sqlerrmc[254]; /* Message text                              */
  char      sqlerrp[8];    /* Used by HiRDB                             */
  long      sqlerrd[6];    /* HiRDB internal status                     */
  char      sqlwarn0;      /* Warning information flag                  */
  char      sqlwarn1;      /* Warning information 1                     */
  char      sqlwarn2;      /* Warning information 2                     */
  char      sqlwarn3;      /* Warning information 3                     */
  char      sqlwarn4;      /* Warning information 4                     */
  char      sqlwarn5;      /* Warning information 5                     */
  char      sqlwarn6;      /* Warning information 6                     */
  char      sqlwarn7;      /* Warning information 7                     */
  char      sqlwarn8;      /* Warning information 8                     */
  char      sqlwarn9;      /* Warning information 9                     */
  char      sqlwarna;      /* Warning information 10                    */
  char      sqlwarnb;      /* Warning information 11                    */
  char      sqlwarnc;      /* Warning information 12                    */
  char      sqlwarnd;      /* Warning information 13 (reserved)         */
  char      sqlwarne;      /* Warning information 14 (reserved)         */
  char      sqlwarnf;      /* Warning information 15 (reserved)         */
  char      sqlcasys1[16]; /* Reserved                                  */
}SQLCA;
extern SQLCA sqlca;
```

(2) COBOL

The next example shows SQL Communications Area expansion when COBOL is used.

```
01  SQLCA IS EXTERNAL.
  02  SQLCAID      PIC X(8).
  02  FILLER       REDEFINES SQLCAID.
    03  SQLCAIDC   PIC X(5).
    03  SQLCAIDS   PIC X(2).
    03  SQLCAIDE   PIC X(1).
  02  SQLCABC      PIC S9(9) COMP.
  02  SQLCODE      PIC S9(9) COMP.
  02  SQLERRM.
    03  SQLERRML   PIC S9(4) COMP.
    03  SQLERRMC   PIC X(254).
  02  SQLERRP      PIC X(8).
  02  SQLERRD      PIC S9(9) COMP OCCURS 6 TIMES.
  02  SQLWARN.
    03  SQLWARN0   PIC X.
    03  SQLWARN1   PIC X.
    03  SQLWARN2   PIC X.
    03  SQLWARN3   PIC X.
    03  SQLWARN4   PIC X.
    03  SQLWARN5   PIC X.
```

```
          03  SQLWARN6    PIC X.
          03  SQLWARN7    PIC X.
        02  SQLEXT.
          03  SQLWARN8    PIC X.
          03  SQLWARN9    PIC X.
          03  SQLWARNA    PIC X.
          03  SQLWARNB    PIC X.
          03  SQLWARNC    PIC X.
          03  SQLWARND    PIC X.
          03  SQLWARNE    PIC X.
          03  SQLWARNF    PIC X.
        02  SQLCASYS1     PIC X(16).
```

B. SQL Descriptor Area

Sometimes when SQL statements are assembled dynamically during execution of a UAP, the number and attributes of the I/O variables (data exchange areas) necessary for executing the SQL statements can be determined only when the UAP is executed. Therefore, HiRDB requires an area in which I/O variables are determined dynamically during UAP execution. The information in the area (the number, attributes, and I/O variable addresses) is posted to HiRDB via the `OPEN`, `FETCH`, or `EXECUTE` statement. The area is called the *SQL Descriptor Area*.

The area can also be used by the `DESCRIBE` statement to receive information on SQL retrieval items that were preprocessed for dynamic execution.

For details about the UAP description languages that can use the SQL Descriptor Area, see *3.2 Overview of UAPs*.

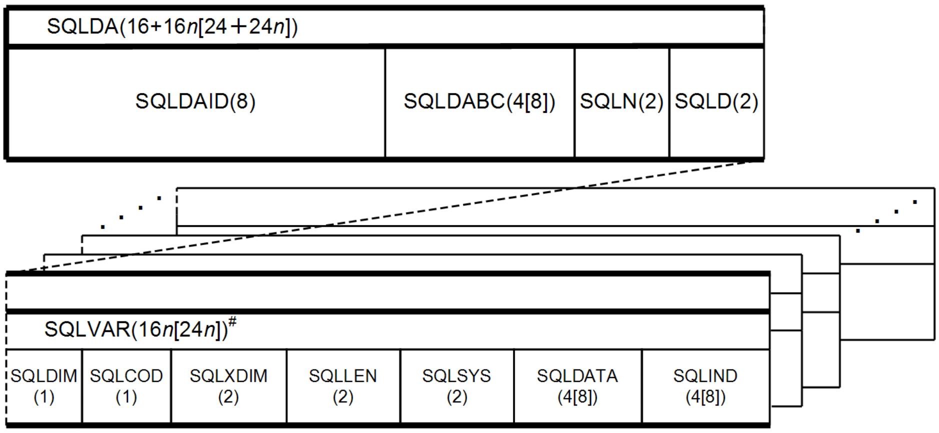
B.1 Organization and contents of the SQL Descriptor Area

This appendix explains the organization and contents of the areas that are described in the information on I/O variables, determined dynamically at the time of UAP execution.

(1) Organization of the SQL Descriptor Area

The following figure shows the organization of the SQL Descriptor Area.

Figure B-1: Organization of the SQL Descriptor Area



Notes

1. Numbers in parentheses indicate length (in bytes).
2. *n* indicates the number of `SQLVAR`s specified in `SQLN`.

3. Square brackets ([ ]) enclose the length for 64-bit mode. For 64-bit mode Windows, SQLDA is 16 + 24*n* bytes.
4. In 64-bit mode, the length of `SQLDABC` is the size of the `long` type for each platform.

#: If `BLOB`- or `BINARY`-type data is used, the area name is `SQLVAR_LOB`, which consists of `SQLDIM(1)`, `SQLCOD(1)`, `SQLXDIM(2)`, `SQLLOBLEN(4)`, `SQLDATA(4 [8])`, and `SQLLOBIND(4 [8])`.

Define the `SQLVAR_LOB` area in the `SQLVAR` area, and use it by overwriting the `SQLVAR` area during the input/output of `BLOB`-type data. For the contents of `SQLVAR_LOB`, see *Table B-3 Contents of SQLVAR_LOB*.

(2) Contents of the SQL Descriptor Area

Table B-1 shows the contents of the SQL Descriptor Area; for details about SQL data, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

Table B-1: Contents of the SQL Descriptor Area

Level number#1	Data Area name	Data type	Length (bytes)	Source of value	Description
1	`SQLDA`	--	16+16*n* [24+24*n*]	--	Denotes the overall SQL Descriptor Area.
2	`SQLDAID`	`char`	8	HiRDB	Contains the `SQLDA ID` (`SQLDA` Δ Δ Δ), indicating the `SQLDA`. This parameter is set when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
2	`SQLDABC`	`long`	4 [8]#6	HiRDB	Contains the length of `SQLDA`. This parameter is set when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
2	`SQLN`#2	`short`	2	UAP	When an `SQLDA` area is allocated or SQLDA is used, this parameter specifies the number of `SQLVAR`s (1 to 4,000) for the allocated SQLDA area.
				HiRDB	Binary `0` is set in this area if there is not enough SQLDA area (`SQLN` < `SQLD`) when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.

Level number#1	Data Area name	Data type	Length (bytes)	Source of value	Description
2	SQLD	short	2	UAP	When the OPEN or EXECUTE statement is issued, specifies the number of input ? parameters in SQLD of the SQL Descriptor Area that is specified in the USING clause. When the EXECUTE statement is issued, specifies the number of output ? parameters in SQLDA of the SQL Descriptor Area that is specified in the INTO clause. When the FETCH statement is issued, specifies the number of retrieval items (1-4,000).
				HiRDB	Binary 0, the number of retrieval items, or the number of output ? parameters is set when the DESCRIBE [OUTPUT] statement is issued: 0: The SQL statement that was preprocessed was a statement other than the SELECT statement and was not a CALL statement containing an output ? parameter Number of retrieval items: The SQL statement that was preprocessed was the SELECT statement Number of output ? parameters: The SQL statement that was preprocessed was the CALL statement

Level number#1	Data Area name	Data type	Length (bytes)	Source of value	Description
					The number of input ? parameters is set when the DESCRIBE INPUT statement is issued. The total number of user-defined type configuration elements that the system tried to receive and high-order user-defined type configuration elements being inherited is set when the DESCRIBE TYPE statement is issued. However, if the number of configuration elements exceeds 30,000, 30001 is set.
2	SQLVAR	--	16*n* [24*n*]	--	Area composed of the SQLDIM, SQLCOD, SQLXDIM, SQLLEN, SQLSYS, SQLDATA, and SQLIND areas. This set of areas should be defined at least as many times as the value specified in the SQLN area.
3	SQLDIM	unsigned char	1	--	Not used.
3	SQLCOD	unsigned char	1	UAP	A data code#3 is specified in this area when an EXECUTE, OPEN, or FETCH statement is issued.
				HiRDB	A data code#3 is set in this area after a DESCRIBE or DESCRIBE TYPE statement is issued.

Level number[#1]	Data Area name	Data type	Length (bytes)	Source of value	Description
3	SQLXDIM	short	2	UAP	One of the following values is specified, depending on the structure type of the area for the variable specified by SQLDA when the EXECUTE, OPEN, or FETCH statement is issued. Simple structure: 1 Repetition structure: 2 to 30000 (integer indicating maximum number of elements in the area) For details about data area structures, see *F. SQL Data Types and Data Descriptions*.
				HiRDB	One of the following values is set depending on the structure type of the retrieval item or ? parameter when the DESCRIBE or DESCRIBE TYPE statement is issued. Simple structure: 1 Repetition structure: 2 to 30000 (integer indicating maximum number of members in the area)
3	SQLLEN[#3, #4]	short	2	UAP	A data length[#3] is set in this area when an EXECUTE, OPEN, or FETCH statement is issued.
				HiRDB	A data length[#3] is set in this area after a DESCRIBE or DESCRIBE TYPE statement is issued.

Level number#1	Data Area name	Data type	Length (bytes)	Source of value	Description
3	`SQLSYS`	`short`	2	UAP	The following value is specified when the `EXECUTE`, `OPEN`, or `FETCH` statement is issued: • Length of the area for one element that includes a gap when a variable-length character string type (`VARCHAR`, `NVARCHAR`, `MVARCHAR`) of a repetition structure or array structure is specified. • `0` for all other cases
				HiRDB	`0` is set when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
3	`SQLDATA`#5	`unsigned char`	4 [8]	UAP	Specifies the address of the data area that stores the value of the `?` parameter when either an `EXECUTE` or an `OPEN` statement is issued.#5 When a `FETCH` statement is issued, this area specifies the address of the data area that receives the data.
3	`SQLIND`#5	`short`	4 [8]	UAP	Specifies the address of the area for receiving the value of the indicator variable only if a data code with an indicator variable is set in `SQLCODE` when an `EXECUTE`, `OPEN`, or `FETCH` statement is issued. The area for receiving the value of the indicator variable is 2 bytes. For details about indicator variable specification, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

Legend:

∆ : One space.

--: Not applicable.

Note

Square brackets ([ ]) enclose the length for 64-bit mode. For 64-bit mode Windows, SQLCA is 16 + 24*n* bytes.

#1: Level numbers indicate the set inclusion relationships of the SQL Descriptor Area. For example, the level 1 data area is composed of level 2 data areas.

#2: The number of `SQLVAR`s set by a UAP in the `SQLN` area should be either the number of `?` parameters set in the `SQLD` area or a value greater than the number of retrieval items. If the number of `SQLVAR`s is less than the number of `?` parameters or less than the number of retrieval items, HiRDB posts this fact by returning binary `0` to the `SQLN` area.

#3: For details about the data codes and data lengths, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

#4: For a packed decimal number (`DECIMAL`, `INTERVAL YEAR TO DAY`, or `INTERVAL HOUR TO SECOND`), the `SQLLEN` area is composed of the following areas:

Data area name	Data type	Length (bytes)	Description
`SQLPRCSN`	B	1	Precision (*p*)
`SQLSCALE`	B	1	Decimal scaling position (*s*)

#5: Because the `SQLDATA` and `SQLIND` areas are cleared when a `DESCRIBE` statement is executed, a value must be reset after the `DESCRIBE` statement has executed. For repetition columns, use the following structure to set a value:

Structure of variables for setting repetition columns in `SQLDATA`

4-byte, binary number area (area for storing the current element count)	Area for the first element of the data type indicated by `SQLCOD`	Area for the second element of the data type indicated by `SQLCOD`		Area for the *n*-th element of the data type indicated by `SQLCOD`

n indicates the maximum element count for the variable.

#6: In 64-bit mode, the length is the size of the `long` type for each platform. In COBOL in 64-bit mode, a declaration for `long` type is `S9(18) COMP`.

Table B-2: Data codes and data lengths set in the SQL Descriptor Area

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit
0	00	--	Data type not available in HiRDB	0	Bytes
1	01	--	Any data type[#1]	0	
48	30	No	C VARCHAR(*n*)[#4]	$1 \leq n \leq$ 3,2000[#2]	
49	31	Yes			
68	44	No	ROW	Row length L in table to be operated on: $1 \leq L \leq$ 30,000	
69	45	Yes			
100	64	No	INTERVAL YEAR TO DAY	Precision 8 Decimal scaling factor 0	Digits
101	65	Yes			
110	6E	No	INTERVAL HOUR TO SECOND	Precision 6 Decimal scaling factor 0	
111	6F	Yes			
112	70	No	DATE	4	Bytes
113	71	Yes			
120	78	No	TIME	3	
121	79	Yes			
124	7C	No	TIMESTAMP[(p)]	$7 \div \uparrow p \div 2 \uparrow$ p = 0, 2, 4, or 6	
125	7D	Yes			
131	83	--	Abstract data type[#3]	--	--

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit
144	90	No	BINARY(*n*)	1 ≤ *n* ≤ 2,147,483,647[#2]	Bytes
145	91	Yes			
146	92	No	BLOB[(*n*)]	1 ≤ *n* ≤ 2,147,483,647	
147	93	Yes			
154	9A	No	BINARY locator	4	
155	9B	Yes			
158	9E	No	BLOB locator	4	
159	9F	Yes			
160	A0	No	MVARCHAR(*n*)	1 ≤ *n* ≤ 32,000[#2]	
161	A1	Yes			
164	A4	No	MCHAR[(*n*)]	1 ≤ *n* ≤ 30,000	
165	A5	Yes			
176	B0	No	NVARCHAR(*n*)	1 ≤ *n* ≤ 16,000[#2]	Characters
177	B1	Yes			
180	B4	No	NCHAR(*n*) or NATIONAL CHAR[ACTER](*n*)	1 ≤ *n* ≤ 15,000	
181	B5	Yes			
192	C0	No	VARCHAR(*n*)	1 ≤ *n* ≤ 32,000[#2]	Bytes
193	C1	Yes			
196	C4	No	CHAR[ACTER](*n*)	1 ≤ *n* ≤ 30,000	
197	C5	Yes			
224	E0	No	FLOAT or DOUBLE PRECISION	8	
225	E1	Yes			
226	E2	No	SMALLFLT or REAL	4	
227	E3	Yes			

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit

`228`	`E4`	No	`[LARGE]DEC[IMAL]`
			`[(`*p*` [,`*s*`])]`
				Precision *p*
				Decimal scaling factor *s*
				$1 \leq p \leq 29, 0 \leq s \leq p$
					Digits

`229`	`E5`	Yes			
`234`	`EA`	No	`DISPLAY SIGN LEADING SEPARATE`[#5]	Precision *p* Decimal scaling factor *s* $1 \leq p \leq 29, 0 \leq s \leq p$	
`235`	`EB`	Yes			
`236`	`EC`	No	`DISPLAY SIGN TRAILING`[#5]	Precision *p* Decimal scaling factor *s* $1 \leq p \leq 29, 0 \leq s \leq p$	Digits
`237`	`ED`	Yes			
`240`	`F0`	No	`INT[EGER]`	4	Bytes
`241`	`F1`	Yes			
`244`	`F4`	No	`SMALLINT`	2	
`245`	`F5`	Yes			

Legend:

--: Not applicable.

#1: HiRDB sets this data code only when the `DESCRIBE INPUT` statement or the `PREPARE` statement with `INPUT` specified is executed on an SQL statement with the `NULL` predicate specified for a `?` parameter (`? IS NULL`). This data code cannot be used for any other purpose. If you use the SQL descriptor area received by the `DESCRIBE INPUT` statement or the `PREPARE` statement with `INPUT` specified in order to specify a value for the `?` parameter, you must specify the data code and data length again.

#2: When a variable-length character string of `0` length is set in the UAP, `1` must be set in the `SQLLEN` area.

#3: When the `DESCRIBE` statement is executed, a data type is returned from the server. The UAP can reference data types. Data type setup and data length setup and referencing are disabled.

#4: This data type can be set in C.

#5: This data type can be set in COBOL.

Table B-3: Contents of SQLVAR_LOB

Level number#1	Data Area name	Data type	Length (bytes)	Source of value	Description
2	`SQLVAR_LOB`	--	16*n* [24*n*]	--	Area that consists of `SQLDIM`, `SQLCOD`, `SQLXDIM`, `SQLLOBLEN`, `SQLDATA`, `SQLDATA`, and `SQLLOBIND`. Define this area in the `SQLVAR` area, and use it by overwriting the `SQLVAR` area during the input/out of `BLOB`- and `BINARY`-type data.
3	`SQLDIM`	`unsigned char`	1	--	Not used.
3	`SQLCOD`	`unsigned char`	1	UAP	Specifies a data code#2 when the `EXECUTE`, `OPEN`, or `FETCH` statement is issued.
				HiRDB	Contains a data code#2 after the `DESCRIBE` or `DESCRIBE TYPE` statement has been issued.
3	`SQLXDIM`	`short`	2	UAP	Specifies `1` when the `EXECUTE`, `OPEN`, or `FETCH` statement is issued. For details about the data area structures, see *F. SQL Data Types and Data Descriptions*.
				HiRDB	Contains the value `1` after the `DESCRIBE` or `DESCRIBE TYPE` statement has been issued.

3	`SQLLOBLEN`#2	`long`
		`[int]`
			4	UAP	Specifies the data length#2 when the `EXECUTE`, `OPEN`, or `FETCH` statement is issued.

				HiRDB	Contains the data length#2 after the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
3	`SQLDATA`#3	`unsigned char *`	4 [8]	UAP	When the `EXECUTE` or `OPEN` statement is issued, specifies the address of the data area in which a `?` parameter value is stored. When the `FETCH` statement is issued, specifies the address of the data area that receives the data.

Level number[#1]	Data Area name	Data type	Length (bytes)	Source of value	Description
3	SQLLOBIND[#3]	long * [int *]	4 [8]	UAP	Specifies the address of the area for receiving the value of the indicator variable only if a data code with an indicator variable is set in SQLCODE when an EXECUTE, OPEN, or FETCH statement is issued. The area for receiving the value of the indicator variable is 4 bytes. For details about indicator variable specification, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

--: Not applicable.

Note

The square brackets in the data type and length columns indicate the data type and length in the 64-bit mode.

#1: Level numbers indicate the set inclusion relationships of the SQL Descriptor Area. For example, the level 2 data area is composed of the level 3 data areas.

#2: For details on data length and data codes, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

#3: The SQLDATA and SQLLOBIND data areas are cleared when a DESCRIBE statement is executed. Therefore, if you use a DESCRIBE statement, reset the values for these data areas after executing the DESCRIBE statement. For the structure for setting a value in a repetition column, see footnote #5 in *Table B-1 Contents of the SQL Descriptor Area*.

B.2 Expanding the SQL Descriptor Area

The SQL Descriptor Area is allocated by means of a declaration within the UAP.

The format of the SQL Descriptor Area expanded in a source program is shown below, followed by an example.

(1) Expansion format of the SQL Descriptor Area

(a) C

This example shows SQL Descriptor Area expansion when C is used.

```
struct {
   char     sqldaid[8]; /* Table ID                           */
   long     sqldabc;    /* Table length                       */
   short    sqln;       /* Elements count in SQLVAR array     */
   short   sqld;          /* ? parameters count, retrieval items count      */
   struct  sqlvar{        /* Data information area                           */
     unsigned char  sqldim;       /* Unused                                 */
     unsigned char  sqlcod;       /* Data code                              */
     short          sqlxdim;      /* Maximum elements count                 */
     union {
       short          sqllen;     /* Data length                            */
       struct {
         unsigned char  sqlprcsn; /* Precision                              */
         unsigned char  sqlscale; /* Scale                                  */
       } s_sqllen;
     } sqllen;
     short          sqlsys;       /* Unused                                 */
     unsigned char  *sqldata;     /* Data area address                      */
     short          *sqlind;      /* Indicator variable address             */
   } SQLVAR[n];#1
} sqlda;#2
```

#1: *n* indicates the required number (`1` to `30000`).

#2: Any desired character string can be specified for the structure name (`sqlda` portion), except that no character string beginning with `SQL` is allowed.

(b) COBOL

This example shows SQL Descriptor Area expansion when COBOL is used.

- In Windows edition and 32-bit mode UNIX edition

```
01  USQLDA.#1
  02  USQLDAID         PIC  X(8)  VALUE  'SQLDA'.
  02  USQLDABC         PIC  S9(9)  COMP.
  02  USQLN            PIC  S9(4)  COMP.
  02  USQLD            PIC  S9(4)  COMP.
  02  USQLVAR   OCCURS  n  TIMES.#2
    03  USQLTYPE       PIC  S9(4)  COMP.
  03  FILLER  REDEFINES  USQLTYPE.
      04  USQLDIM      PIC  X(1).
      04  USQLCOD      PIC  X(1).
    03  USQLXDIM       PIC  S9(4)  COMP  VALUE  IS  1.
 03  USQLATTR.
      04  USQLLEN      PIC  S9(4)  COMP.
      04  FILLER  REDEFINES  USQLLEN.
        05  USQLPRCSN  PIC  X(1).
        05  USQLSCALE  PIC  X(1).
      04  USQLSYS      PIC  S9(4)  COMP.
    03  FILLER  REDEFINES  USQLATTR.
      04  USQLLOBLEN   PIC  S9(9)  COMP.
    03  USQLDATA  USAGE  IS  ADDRESS.
```

```
      03  USQLIND   USAGE  IS  ADDRESS.
```

#1: Any name can be specified as the name of the set item (USQLDA area); however, a character string that begins with SQL cannot be used for a data item.

#2: *n* indicates the required number (1 to 30000).

- In 64-bit mode UNIX edition

```
01  USQLDA.#1
  02  USQLDAID         PIC  X(8)  VALUE  'SQLDA'.
  02  USQLDABC         PIC  S9(18)  COMP. #3
  02  USQLN            PIC  S9(4)  COMP.
  02  USQLD            PIC  S9(4)  COMP.
  02  FILLER           PIC  X(4). #3
  02  USQLVAR   OCCURS  n  TIMES.#2
    03  USQLTYPE       PIC  S9(4)  COMP.
    03  FILLER  REDEFINES  USQLTYPE.
      04  USQLDIM      PIC  X(1).
      04  USQLCOD      PIC  X(1).
    03  USQLXDIM       PIC  S9(4)  COMP  VALUE  IS  1.
    03  USQLATTR.
      04  USQLLEN      PIC  S9(4)  COMP.
      04  FILLER  REDEFINES  USQLLEN.
        05  USQLPRCSN  PIC  X(1).
        05  USQLSCALE  PIC  X(1).
      04  USQLSYS      PIC  S9(4)  COMP.
    03  FILLER  REDEFINES  USQLATTR.
      04  USQLLOBLEN   PIC  S9(9)  COMP.
    03  USQLDATA  USAGE  IS  ADDRESS.
    03  USQLIND   USAGE  IS  ADDRESS.
```

#1: Any name can be specified as the name of the set item (USQLDA area); however, a character string that begins with SQL cannot be used for a data item.

#2: *n* indicates the required number (1 to 30000).

#3: The Windows edition differs from the 32-bit mode UNIX edition in the following respects:

- Contents of the PICTURE clause in USQLDABC
- Whether there is a FILLER item between USQLD and USQLVAR

(2) SQL Descriptor Area example

(a) Declaration and area allocation for using the SQL Descriptor Area

The SQL Descriptor Area is declared and allocated in the UAP.

(b) Collection of retrieval item information

This next example illustrates collecting retrieval item information. The items identified by numbers in the code are explained as follows.

```
EXEC SQL BEGIN DECLARE SECTION;   .................... 1
struct{   .......................................... 1
long#  cmd_len;   .................................. 1
char  cmd_data[1000];   ............................ 1
}XCMND;   .......................................... 1
EXEC SQL END DECLARE SECTION;   .................... 1
XCMND.cmd_len=(long#)sprintf(XCMND.cmd_data,
                 "SELECT*FROM stock WHERE GNO=1")   .. 2
EXEC SQL WHENEVER SQLERROR GO TO :RERROR;   ......... 3
EXEC SQL PREPARE ST1 FROM :XCMND;   ................. 4
EXEC SQL DESCRIBE ST1 INTO :DAREA;   ................ 5
```

Notes

1. When a `DESCRIBE` statement is executed, binary `0` or the number of retrieval items is set in the `SQLD` area:

   - `0` is set when the SQL statement that was preprocessed is not a `SELECT` statement.

   - Number of retrieval items is set when the SQL statement that was preprocessed is a `SELECT` statement.

2. The data code, data length, and maximum elements count of each retrieval item are set in `SQLCOD`, `SQLLEN`, and `SQLXDIM`, respectively.

#: In 64-bit mode, this is `int`.

Explanation

1. Declares an embedded variable (`XCMND`) for storing the SQL statements.
2. Sets the SQL statement in the variable (`XCMND`).
3. Specifies the action to be taken if an error occurs after SQL statement execution.
4. Preprocesses the SQL statements specified as the variable `XCMND` and assigns an SQL statement identifier (`ST1`).
5. Collects the information of items retrieved by the SQL statements (`ST1`) into the SQL Descriptor Area (`DAREA`).

(c) Fetching retrieval results with dynamic receive area allocation

In this example, retrieval results are fetched into areas allocated based on the information obtained using a DESCRIBE statement. The items in italics in the figure are explained as follows.

```
for(n=0;n<DAREA.sqld;n++){   .................... 1
   DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);
.. 1
    DAREA.SQLVAR[n].sqlind=&(X_IND[n]);   ....... 1
}   ............................................ 1
EXEC SQL DECLARE CR1 CURSOR FROM ST1;   ......... 2
EXEC SQL OPEN CR1   ............................ 3
EXEC SQL WHENEVER NOT FOUND GO TO:FEND;   ....... 4
for(;;){   .................................... 5
    EXEC SQL FETCH CR1 USING DESCRIPTOR:DAREA   . 5
                    :   ...................... 5,6
}   ............................................ 5
    EXEC SQL WHENEVER NOT FOUND CONTINUE;   ...... 7
FEND:EXEC SQL CLOSE CR1;   ...................... 8
```

Notes

Before the FETCH statement is executed, the following information must be set in DAREA:

- Size of SQLVAR array (SQLN)
- Number of areas to receive retrieval results (SQLD): executing a DESCRIBE statement sets this value
- Data type of receive area (SQLCOD): executing a DESCRIBE statement sets this value
- Data length of receive area (SQLLEN): executing a DESCRIBE statement sets this value.

Explanation

1. Sets the address of the allocated area in the SQL Descriptor Area (DAREA).
2. Declares a cursor (CR1) for the SQL statement identifier (ST1).
3. Opens the cursor (CR1).
4. Specifies the action to be taken (branching to FEND) at the termination of retrieval.
5. Advances the cursor (CR1) to the next line, and fetches that line into the area specified by the SQL Descriptor Area (DAREA).
6. Specifies the processing to be performed on the retrieval result (e.g., editing and output).

7. Invalidates the action at the termination of retrieval.
8. Closes the cursor (`CR1`).

(d) Dynamic allocation of a data area for specifying ? parameter values

This is an example of inserting data into a dynamically specified table. The items in italics in the figure are explained as follows.

```
char TNAME[30];    ......................................... 1
scanf("%S",TNAME);    ................................. 2
XCMND.cmd_len=(long#)sprint(XCMND.cmd_data,
                    "SELECT * FROM %S",TNAME);    ..... 3

EXEC SQL PREPARE ST1 FROM:XCMND;    .................. 3

EXEC SQL DESCRIBE ST1 INTO:DAREA;    ................. 3
         :   ........................................ 4
for(n=0;n<DAREA.sqld;n++){    ...................... 5
DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);
................................ 5
DAREA.SQLVAR[n].sqlind=&(X_IND[n]);    ................ 5
}  .......................................... 3
XCMND.cmd_len=(long#)sprit(XCMND.cmd_data,
              "INSERT INTO %S VALUES(?,...,?)",TNAME);
.......................... 6
 EXEC SQL PREPARE ST2 FROM:XCMND;    ................. 7
for(;;){    ....................................... 8
   [Input insertion data (branched to IEND if there is no data)];    ..... 8
   [Insert data in the data area and the indicator variable area];    ..... 8
  EXEC SQL EXECUTE ST2 USING DESCRIPTOR:DAREA;    ..... 8
}   ......................................... 8
IEND:
```

#: In 64-bit mode, this is `int`.

Explanation

1. Declares the variable (`TNAME`) that stores the table name.
2. Loads the table name from the input data into the variable (`TNAME`).
3. Uses the `DESCRIBE` statement to set the columns count of the table specified in `2` data type, data length, and maximum elements count of each column, as the number of `?` parameters, the data type, data length, and maximum elements count of the data area for each `?` parameter, respectively, in the SQL Descriptor Area (`DAREA`).
4. Allocates data area for each `?` parameter.
5. Sets the address of the allocated area in the SQL Descriptor Area (`DAREA`).

6. Creates an INSERT statement for inserting data into the specified table.
7. Preprocesses the INSERT statement in XCMND and assigns the SQL statement identifier (ST2).
8. Repeats data insertion on a row basis, setting in the data area, and execution using the EXECUTE statement, as long as data to be inserted exists.

(e) Retrieving DECIMAL data using a FETCH statement

In this example, DECIMAL data is retrieved using a FETCH statement.

1. Declares a data area and an indicator variable.

```
EXEC  SQL  BEGIN  DECLARE  SECTION ;

   SQL TYPE IS DECIMAL(20,0) xdec1 ;      /* Data area      */
   short                     xdec1_i ;    /* Indicator variable  */

EXEC  SQL  END  DECLARE  SECTION ;
```

2. Sets an SQL Descriptor Area.

```
PDSQLCOD(usrsqlda, 2)=PDSQL_DECIMAL_I  ;  /* Sets a data code  */

  PDSQLPRCSN(usrsqlda, 2)=20           ;  /* Sets precision     */
  PDSQLSCALE(usrsqlda, 2)= 0           ;  /* Sets a scale    */
  PDSQLDATA(usrsqlda, 2)=(void*)xdec1  ;  /* Embedded variable
address*/
                                          /* Setting          */
  PDSQLXDIM(usrsqlda, 2)=1;              /* Not a repetition column  */

PDSQLIND(usrsqlda, 2)=(void*)&xdec1_i ;  /* Indicator variable address
*/
                                          /* Setting          */
```

(3) SQL Descriptor Area expansion

The following table shows the procedure for expanding the SQL Descriptor Area.

Table B-4: SQL Descriptor Area expansion procedure

Language	When include file is used	When directly coded by user
C	`#include <pdbsqlda.h> PDUSRSQLDA(n) usrsqlda;`	Expansion of SQL Descriptor Area is coded directly.
COBOL	COPY SQLDA# [REPLACING 256 BY *n*].	Expansion of SQL Descriptor Area is coded directly. Level 01 must always be specified first.

#: In 64-bit mode, change SQLDA to SQLDA64.

Following is a COBOL coding example in which parameters are specified in the SQL Descriptor Area:

```
EXEC  SQL
    BEGIN  DECLARE  SECTION
END   EXEC
01  IN-CHR1  PIC  X(15).
01  IN-IND1  PIC  S9(4)  COMP.
EXEC  SQL
    END  DECLARE  SECTION
END-EXEC
COPY  SQLDA.
         :
COMPUTE  USQLDABC=32
COMPUTE  USQLN=1
COMPUTE  USQLD=1
COMPUTE  USQLDATA(1)=FUNCTION  ADDR(IN-CHR1)
MOVE  SQLCNST0    TO USQLDIM(1)
MOVE  SQLDCOD197  TO USQLCOD(1)
COMPUTE  USQLXDIM(1)=1
COMPUTE  USQLLEN(1)=15
COMPUTE  USQLIND(1)=FUNCTION  ADDR(IN-INT1)
EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :USQLDA
END-EXEC
```

(4) SQL Descriptor Area operation macros

Various macros for declaring the SQLDA and for setting/referencing values are defined in C language. These macros can be used by including the unique header file (pdbsqlda.h) in the UAP. *Table B-5* shows the SQL Descriptor Area operation macros, and *Table B-6* shows the macros for specifying data types.

Table B-5: SQL Descriptor Area operation macros

Macro	Function
PDUSRSQLDA(*m*)	Declares a user SQLDA.
PDSETSIZE(usrsqlda,*m*)	Specifies the SQLDA size.
PDSQLN(usrsqlda)	Specifies the ? parameter.
PDSQLD(usrsqlda)	Specifies/references the ? parameter and the number of retrieval items.
PDSQLCOD(usrsqlda,*n*)	Specifies/references the data code.
PDSQLLEN(usrsqlda,*n*)	Specifies/references the data length (other than BLOB and decimal number).

Macro	Function
PDSQPRCSN(usrsqlda,*n*)	Specifies/references the precision (decimal number only).
PDSQLSCALE(usrsqlda,*n*)	Specifies/references the scale (decimal number only).
PDSQLDATA(usrsqlda,*n*)	Specifies the address of the data area.
PDSQLIND(usrsqlda,*n*)	Specifies an indicator variable address.
PDSQLLOBLEN(usrsqlda,*n*)	Specifies/references the BLOB data length.
PDSQLDIM(usrsqldata,*n*)	Specifies/references the value in unused area.
PDSQLXDIM(usrsqldata,*n*)	Specifies/references the maximum number of elements for the repetition structure.
PDSQLSYS(usrsqldata,*n*)	Specifies the length of one element that includes the gap in variable-length character string type for the repetition structure or array structure.

Legend:

usrsqlda: User-defined SQL Descriptor Area name; any name can be specified.

m: Number of ? parameters (1 to 30,000).

n: Number of ? parameters to be specified or referenced (0 to 29,999).

Table B-6: Macros for specifying data types

Macro	Indicator variable	Corresponding data type
PDSQL_FLOAT PDSQL_FLOAT_I	No Yes	FLOAT
PDSQL_SMALLFLT PDSQL_SMALLFLT_I	No Yes	SMALLFLT
PDSQL_DECIMAL PDSQL_DECIMAL_I	No Yes	DECIMAL
PDSQL_INTEGER PDSQL_INTEGER_I	No Yes	INTEGER
PDSQL_SMALLINT PDSQL_SMALLINT_I	No Yes	SMALLINT
PDSQL_VARCHAR PDSQL_VARCHAR_I	No Yes	VARCHAR
PDSQL_CHAR PDSQL_CHAR_I	No Yes	CHAR

Macro	Indicator variable	Corresponding data type
PDSQL_NVARCHAR PDSQL_NVARCHAR_I	No Yes	NVARCHAR
PDSQL_NCHAR PDSQL_NCHAR_I	No Yes	NCHAR
PDSQL_MVARCHAR PDSQL_MVARCHAR_I	No Yes	MVARCHAR
PDSQL_MCHAR PDSQL_MCHAR_I	No Yes	MCHAR
PDSQL_DATE PDSQL_DATE_I	No Yes	DATE
PDSQL_TIME PDSQL_TIME_I	No Yes	TIME
PDSQL_YEARTODAY PDSQL_YEARTODAY_I	No Yes	INTERVAL YEAR TO DAY
PDSQL_HOURTOSEC PDSQL_HOURTOSEC_I	No Yes	INTERVAL HOUR TO SECOND
PDSQL_ROW PDSQL_ROW_I	No Yes	ROW
PDSQL_BLOB PDSQL_BLOB_I	No Yes	BLOB
PDSQL_TIMESTAMP PDSQL_TIMESTAMP_I	No Yes	TIMESTAMP
PDSQL_BINARY PDSQL_BINARY_I	No Yes	BINARY
PDSQL_BLOB_LOC PDSQL_BLOB_LOC_I	No Yes	BLOB locator
PDSQL_BINARY_LOC PDSQL_BINARY_LOC_I	No Yes	BINARY locator
PDSQL_CVARCHAR PDSQL_CVARCHAR_I	No Yes	VARCHAR for C

Following is a C coding example in which parameters are specified in the SQL Descriptor Area:

```
#include <pdbsqlda.h>                         /* Includes header file. */
```

```
EXEC  SQL  BEGIN  DECLARE  SECTION ;
short  xint1 ;
char   xchr1[16] ;
EXEC  SQL  END  DECLARE  SECTION ;
PDUSRSQLDA(2)  usrsqlda ;                   /* Declares SQL Descriptor
                                           Area. */
          :
ClearSqlda(2);                             /* Clears SQL Descriptor Area */
PDSQLCOD(usrsqlda, 0)=PDSQL_SMALLINT ;  /* Sets data code.             */
PDSQLLEN(usrsqlda, 0)=sizeof(short)  ;  /* Sets data code.             */
PDSQLDATA(usrsqlda, 0)=(void*)&xint ;   /* Sets embedded variable
                                          address.   */
PDSQLIND(usrsqlda, 0)=NULL ;            /* Sets indicator variable
                                          address. */
PDSQLCOD(usrsqlda, 1)=PDSQL_CHAR ;      /* Sets data code. */
PDSQLLEN(usrsqlda, 1)=sizeof(xchar)-1 ;/* Sets data code. */
PDSQLDATA(usrsqlda, 1)=(void*)xchr ;    /* Sets embedded variable
                                         address.   */
PDSQLIND(usrsqlda, 1)=NULL ;            /* Sets indicator variable
                                          address.  */

EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :usrsqlda ;
```

(5) Expansion format of repetition columns

During compilation, embedded variables in a repetition column are expanded into the structures shown in Table B-7 based on macro definition. The explanation here applies to the C language.

The macro for manipulating a repetition column uses the members of the expanded structures to reference the elements of the repetition column.

If the user wishes to directly set an address in the SQL Descriptor Area by securing an area, the area must be assigned to a language boundary. FLOAT ARRAY explicitly includes a free area for language boundary adjustment. However, when setting an address in the SQL Descriptor Area, you must set it by taking a free area into consideration.

Specify these expansion formats only when adjusting a boundary or determining a size during this type of area allocation. When specifying a repetition column as an embedded variable, do not specify an expansion format. Instead, use the macros described in *F. SQL Data Types and Data Descriptions*.

Table B-7: Repetition column expansion format

SQL data type	Macro name	Expansion format

SMALL INT ARRAY[m]	PD_MV_SINT(m)	struct {
		  long mcnt;
		  short data[m];
		}

INTEGER ARRAY[m]	PD_MV_INT(m)	struct{
		  long mcnt
		  long data[m];
		}

SMALL FLT ARRAY[m]	PD_MV_SFLT(m)	struct {
		  long mcnt;
		  float data[m];
		}

FLOAT ARRAY[m]	PD_MV_FLT(m)	struct
		  union {
		   double resv1;
		    struct {
		     long resv2;
		     long mcnt;
		   }mcnt_dmy2;
		  } mcnt_dmy1;
		  double data[m];
		}

CHAR(n) ARRAY[m]	PD_MV_CHAR(m, n)	struct {
		  long mcnt;
		  char data[m][(n)+1];
		}

NCHAR(n) ARRAY[m]	PD_MV_NCHAR(m, n)	struct {
		  long mcnt;
		  char data[m][2*(n)+1];
		}

SQL data type	Macro name	Expansion format

VARCHAR(*n*) ARRAY[*m*]	PD_MV_VCHAR(*m*, *n*)	struct {
		long mcnt;
		struct {
		short len;
		char str[*n*];
		} data[*m*];
		}

	PD_MV_CVCHAR(*m*, *n*)	struct {
		long mcnt;
		char data[*m*][(*n*)+1];
		}

NVARCHAR(*n*) ARRAY[*m*]	PD_MV_NVCHAR(*m*, *n*)	struct {
		long mcnt;
		struct {
		short len;
		char str[2*(*n*)+1];
		} data[*m*];
		}

DECIMAL
[(*p*[,*s*])]ARRAY[*m*]
	PD_MV_DEC(*m*, *p*, *s*)	struct {
		long mcnt;
		unsigned char data[*m*][(*p*)/2+1];
		}

C. Column Name Descriptor Area

When using the SQL Descriptor Area to receive the following information, you can also receive the column name information and routine parameter information by specifying a Column Name Descriptor Area:

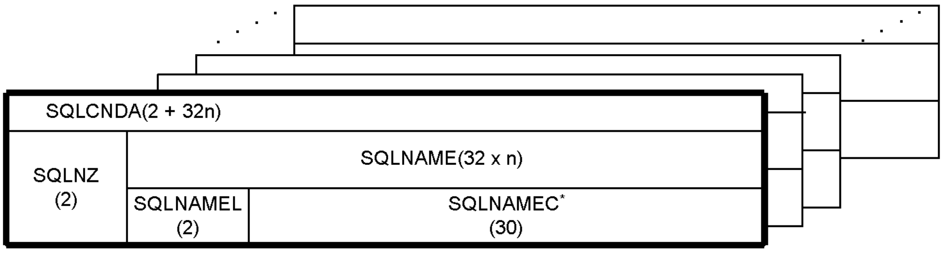
- Retrieval item information (number of retrieval items, as well as each retrieval item's data type, data length, and maximum number of elements)
- CALL statement's input/output ? parameter information (number of ? parameters, as well as the ? parameter's data type and data length)

C.1 Organization and contents of the Column Name Descriptor Area

(1) Organization of the Column Name Descriptor Area

The following figure shows the organization of the Column Name Descriptor Area (SQLCNDA).

Figure C-1: Organization of the Column Name Descriptor Area



Note

Numbers in parentheses indicate length (in bytes).

#: SQLNAMEC is an array of variable-length character strings with a maximum length of 30 bytes. The array should be the same length as the SQLVAR array in the SQL Descriptor Area. For details about the size of the SQLVAR array, see *B.1 Organization and contents of the SQL Descriptor Area*.

(2) Contents of the Column Name Descriptor Area

The following table shows the contents of the Column Name Descriptor Area.

Table C-1: Contents of the Column Name Descriptor Area

Acquired information item	Type of retrieval item	Description
Retrieval item	Column (without subscript specification)	*column-name*
	Column (with subscript specification)	*column-name* [*subscript*]
	Set function	△ △ COUNT(*) △ △ COUNT_FLOAT(*) {△ {■ |△ } *function-name* (*column-name*) | △ { ■ |△ } *function-name* (DISTINCT-*column-name*) | △ △*function-name* (△ EXP)}
	Window function	△ △ EXP (*integer*)
	Value expression (including literal)	△ △ EXP(*integer*)
	WRITE specification	△ △ EXP(*integer*)
	ROW	△ △ ROW
CALL statement's input ? parameter	? parameter	*routine's-parameter-name*
CALL statement's output ? parameter		
User-defined type configuration element	Attribute	*attribute-name*
CALL COMMAND statement's input ? parameter	? parameter	Command name: COMMAND_NAME WITH clause: ● First: WITH ● Subsequent: WITH (*argument-number*) INPUT clause: INPUT ENVIRONMENT clause: ENVIRONMENT SERVER clause: SERVER
CALL COMMAND statement's output ? parameter	? parameter	OUTPUT TO clause: OUTPUT ERROR TO clause: ERROR RETURN CODE TO clause: RETURN_CODE

Legend:

■ : X'FF'

Δ : One space

Notes

1. The *i*th element of the Column Name Descriptor Area stores column name information for the *i*th retrieval item.
2. If a retrieval item is a column, the column name is assigned to the retrieval item from the beginning of the retrieval item. If the length of the column name, including subscripts, is greater than 30 bytes, the excess bytes of the column name are truncated.

   If a retrieval item is not a column, one or two spaces (indicated by Δ in the table) are set at the beginning of the retrieval item. If the column is greater than 30 bytes, X'FF' is set in byte 2. The symbol ■ in the table denotes the value X'FF'.
3. Integer indicates the ordinal number of a retrieval item.
4. For UNION[ALL] or EXCEPT[ALL], the contents of the retrieval item in the query specified first are set in the Column Name Descriptor Areas.
5. If an AS column name is specified, the specified column name is set.
6. The routine's parameter name is set only when the? parameter is specified independently in the CALL statement's argument. If a value expression including the? parameter is specified, SQLNAMEL is set to 0.
7. If the retrieval item is a column of a derived table, the derived column list is omitted after the derived table, and the derived column has no column name in the query selection expression, ΔNONAME is set.

C.2 Expanding the Column Name Descriptor Area

The Column Name Descriptor Area is allocated as static area by declaring it in the UAP.

(1) C

The following code shows the format of the Column Name Descriptor Area that is to be expanded in the source program when C language is used:

```
struct {
    short        sqlnz;          /* Effective arrays count  */
    struct {
        short    sqlnamel;       /* Effective column name length */
        char     sqlnamec[30];   /* Column name storage area    */
    } SQLNAME[n];#1
  }XXXXX;#2
```

#1: *n* indicates the same number (1 to 30000) as the size of the SQLVAR array in the SQL Descriptor Area.

#2: Any desired character string can be specified as the structure name (*XXXXX* portion), except that a character string beginning with `SQL` cannot be specified. When Column Name Descriptor Areas are specified using a `DESCRIBE` statement, the name of the allocated areas must be specified.

(2) COBOL

The following code shows the format of the Column Name Descriptor Area that is to be expanded in the source program when COBOL is used:

```
01  SQLCNDA.#1
  02  SQLNZ  PIC  S9(4) COMP.
  02  SQLNAME  OCCURS  1  TIMES  n.#2
    03  SQLNAMEL  PIC  S9(4)  COMP.
    03  SQLNAMEC  PIC  X(30).
```

#1: Any name can be specified as the name of the set item (`SQLCNDA` area); however, a character string that begins with `SQL` cannot be used for a data item. In addition, the set item level must always be set to `01`.

#2: *n* indicates the required number (1 to 30,000).

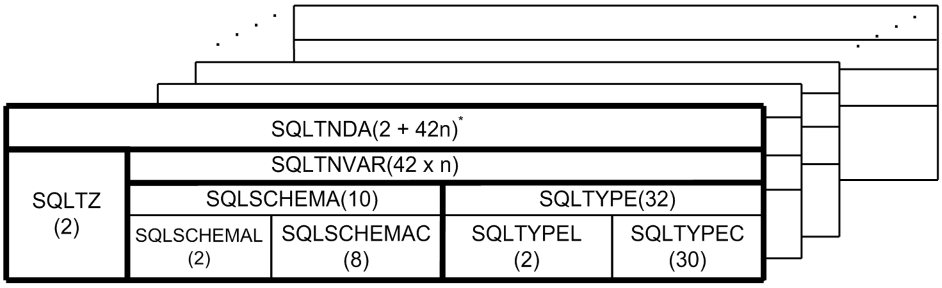
D. Type Name Descriptor Area

When the SQL Descriptor Area is used to receive retrieval item information and user-defined type definition information, user-defined type data type names can also be received by specifying a Type Name Descriptor Area (`SQLTNDA`).

D.1 Organization of the Type Name Descriptor Area

The following figure shows the organization of the Type Name Descriptor Area (`SQLTNDA`).

Figure D-1: Organization of the Type Name Descriptor Area



Note

Parentheses enclose a length in bytes.

#: `SQLTNVAR` is an array of a structure composed of a variable length character string `SQLSCHEMA` with a maximum length of 10 bytes, and a variable-length character string `SQLTYPE` with a maximum length of 32 bytes. The array should be the same length as the `SQLVAR` array in the SQL Descriptor Area. For details about the size of the `SQLVAR` array, see *B.1 Organization and contents of the SQL Descriptor Area*.

D.2 Contents of the Type Name Descriptor Area

The following table shows the contents of the Type Name Descriptor Area.

Table D-1: Contents of the Type Name Descriptor Area

Level number#	Type name area name	Data type	Length (bytes)	Description
1	`SQLTNDA`	--	2+ 42 x *n*	Indicates the name of the entire Type Name Descriptor Area.
2	`SQLTZ`	`short`	2	Specifies the number of retrieval items.

Level number#	Type name area name	Data type	Length (bytes)	Description
2	SQLTNVAR	--	42 x *n*	Area composed of the authorization identifier and data type identifiers.
3	SQLSCHEMA	--	10	Area storing information about the user-defined type authorization identifier.
4	SQLSCHEMAL	short	2	The authorization identifier is set in this area. 0 is set if the data type of the corresponding retrieval item is not a user-defined type.
4	SQLSCHEMAC	char	8	The authorization identifier is set in this area.
3	SQLTYPE	--	32	Area storing information about the user-defined type data type identifier.
4	SQLTYPEL	short	2	The length of the user-defined type is set in this area. 0 is set if a data type of the corresponding retrieval item is not the user-defined type.
4	SQLTYPEC	char	30	The data type identifier of the user-defined type is set in this area.

Legend:

--: Not applicable

#: Level numbers indicate the set inclusion relationships of the Type Name Descriptor Area. For example, the level 2 data area is composed of the level 3 data areas.

D.3 Expanding the Type Name Descriptor Area

The Type Name Descriptor Area is allocated by declaring it in the UAP.

(1) C

The following code shows the format of the Type Name Descriptor Area that is to be expanded in the source program when C is used.

```
struct {
    short       sqlnz;           /* Effective array count */
    struct {
      struct {
        short   sqlchemal;       /* Effective authorization identifier length */
        char    sqlschemac[8];   /* Authorization identifier storage area */
     } sqlschema;
      struct {
        short   sqltypel;        /* Effective length of user-defined type
                                    name */
        char    sqltypec[30];    /* User-defined type name storage area */
    } sqltnvar[n];#1
} Usrsqltnda;#2
```

#1: *n* indicates the same number (`1` to `30000`) as the size of the `SQLVAR` array in the SQL Descriptor Area.

#2: Any desired character string can be specified as the structure name (`usrsqltnda` portion), except that a character string beginning with `SQL` cannot be specified. When the Type Name Descriptor Area is specified using a `DESCRIBE` statement, the name of the allocated area must be specified.

(2) COBOL

The following code shows the format of the Type Name Descriptor Area that is to be expanded in the source program when COBOL is used.

```
01  USQLTNDA.#1
  02  USQLTZ  PIC  S9(4) COMP.
  02  USQLTNVAR  OCCURS  1  TIMES  n.#2
    03  USQLSCHEMA.
      04  USQLSCHEMAL  PIC  S9(4)  COMP.
      04  USQLSCHEMAC  PIC  X(8).
    03  USQLTYPE.
      04  USQLTYPEL  PIC  S9(4)  COMP.
      04  USQLTYPEC  PIC  X(30).
```

#1: Any name can be specified as the name of the set item (`USQLTNDA` portion); however, a character string that begins with `SQL` cannot be used for a data item.

#2: *n* indicates the same number (`1` to `30000`) as the size of the `SQLVAR` array in the SQL Descriptor Area.

E. Character Set Descriptor Area

The character set descriptor area is used for setting character set names for input and output variables that are determined dynamically during UAP execution, and for reporting them to the system with the `OPEN`, `FETCH`, and `EXECUTE` statements. During dynamic execution, the character set descriptor area can also be used to receive a character set name for a `?` parameter or an item retrieved by a preprocessed SQL statement.

For details about the UAP coding languages that use the character set descriptor area, see *3.2 Overview of UAPs*.

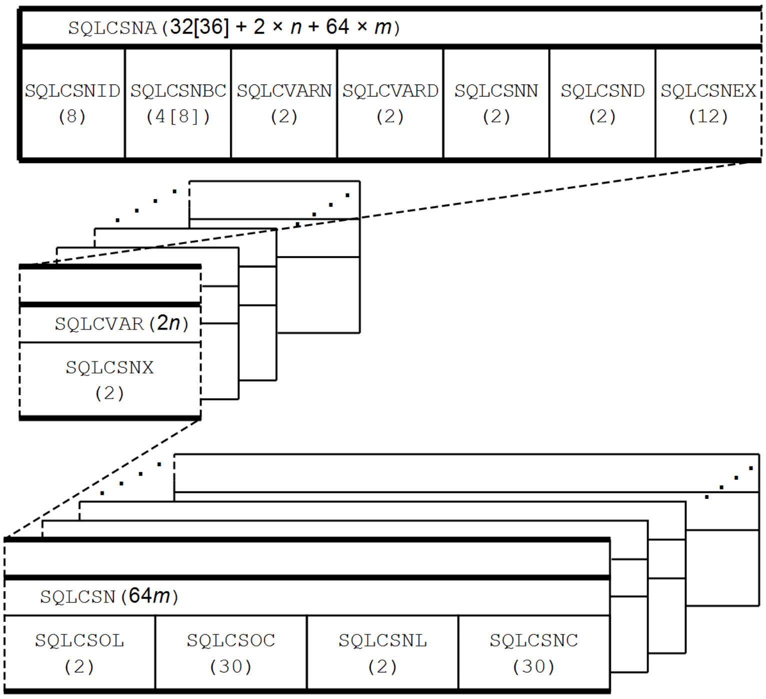
E.1 Organization and contents of the character set descriptor area

This section describes the organization and contents of the area used for setting character set names for input and output variables that are determined dynamically during UAP execution.

(1) Organization of the character set descriptor area

The following figure shows the organization of the character set descriptor area.

Figure E-1: Organization of the character set descriptor area



Note 1

Numbers in parentheses indicate area length (in bytes).

Note 2

n indicates the number of `SQLCVAR`s specified in `SQLCVARN`.

m indicates the number of `SQLCSN`s specified in `SQLCSNN`.

Note 3

A value in square brackets ([ ]) within parentheses indicates the length in 64-bit mode. The value for 64-bit mode Windows is the same as for 32-bit mode.

Note 4

In 64-bit mode, the length of `SQLCSNBC` becomes the size of the `long` type for each platform.

(2) Contents of the character set descriptor area

The following table shows the contents of the character set descriptor area.

Table E-1: Contents of the character set descriptor area

Level number[#1]	Data area name	Data type	Length (bytes)	Source of value	Description
1	SQLCSNA	--	32[36][#2] + 2 x *n* + 64 x *m*	--	Denotes the overall character set descriptor area.
2	SQLCSNID	char	8	HiRDB	Contains the ID "`SQLCSNA`Δ," indicating the `SQLCSNA`. This parameter is set when the `DESCRIBE`, `DESCRIBE CURSOR`, or `PREPARE` statement is issued.
2	SQLCSNBC	long	4 [8][#2]	HiRDB	Contains the length of the `SQLCSNA`. This parameter is set when the `DESCRIBE`, `DESCRIBE CURSOR`, or `PREPARE` statement is issued.

Level number[#] 1	Data area name	Data type	Length (bytes)	Source of value	Description
2	SQLCVARN	short	2	UAP	When an SQLCSNA area is allocated or SQLCSNA is used, specifies the number of SQLCVARs (1 to 4,000) in the area allocated for the SQLCSNA.
				HiRDB	Binary 0 is set when the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued and there is a shortage of SQLCSNA area (SQLCVARN < SQLCVARD).

Level number#1	Data area name	Data type	Length (bytes)	Source of value	Description
2	SQLCVARD	short	2	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the maximum element number of the SQLCVAR for which a character set for input ? parameters has been specified.
				HiRDB	When the DESCRIBE [OUTPUT], DESCRIBE CURSOR, or PREPARE statement is issued, binary 0, the number of retrieval items, or the maximum element number of the SQLCVAR for which a character set for output ? parameters has been specified is set. 0: This value is set in either of the following cases: • The SQL statement was a statement other than a SELECT statement and was not a CALL statement containing output ? parameters. • The retrieval items or output ? parameters do not have a character set other than the default character set. *maximum-element-number*: This value is set when both of the following conditions are satisfied: • The SQL statement was a SELECT statement or a CALL statement containing an output ? parameter. • A retrieval item or an output ? parameter has a character set other than the default character set. When the DESCRIBE INPUT or PREPARE statement is issued, binary 0 or the maximum element number of the SQLCVAR containing character set information for input ? parameters is set.

Level number#1	Data area name	Data type	Length (bytes)	Source of value	Description
					0: This value is set in either of the following cases: • The SQL statement does not contain input ? parameters. • The input ? parameters do not have a character set other than the default character set. *maximum-element-number*: This value is set when both of the following conditions are satisfied: • The SQL statement contains an input ? parameter. • An input ? parameter has a character set other than the default character set.
2	SQLCSNN	short	2	UAP	When an SQLCSNA area is allocated, specifies the number of SQLCSN areas (1) that have been allocated.
				HiRDB	Binary 0 is set when the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued and there is a shortage of SQLCSN area (SQLCSNN < SQLCSND).

Level number#1	Data area name	Data type	Length (bytes)	Source of value	Description
2	SQLCSND	short	2	UAP	When the `OPEN`, `EXECUTE`, `EXECUTE IMMEDIATE`, or `FETCH` statement is issued, specifies the number of character sets (1) that are used by input `?` parameters.
				HiRDB	When the `DESCRIBE [OUTPUT]`, `DESCRIBE CURSOR`, or `PREPARE` statement has been issued and there is a shortage of `SQLCSN` area (`SQLCSNN` < `SQLCSND`), binary `0` or the number of character sets used for retrieval items or output `?` parameters is set. `0`: This value is set in either of the following cases: • The SQL statement was a statement other than a `SELECT` statement and was not a `CALL` statement containing output `?` parameters. • The retrieval items and output `?` parameters do not have a character set other than the default character set. *maximum-element-number*: This value is set when both of the following conditions are satisfied: • The SQL statement was a `SELECT` statement or a `CALL` statement containing an output `?` parameter. • A retrieval item or an output `?` parameter has a character set other than the default character set. When the `DESCRIBE INPUT` or `PREPARE` statement is issued, binary `0` or the number of character sets used in input `?` parameters is set.

Level number[#1]	Data area name	Data type	Length (bytes)	Source of value	Description
					0: This value is set in either of the following cases: • The SQL statement does not contain input ? parameters. • The input ? parameters do not have a character set other than the default character set. *maximum-element-number*: This value is set when both of the following conditions are satisfied: • The SQL statement contains an input ? parameter. • An input ? parameter has a character set other than the default character set.
2	SQLCSNEX	--	12	--	Not used
2	SQLCVAR	--	2 x *n*	--	Area composed of SQLCSNXs. This area corresponds to SQLVAR of SQLDA and must be defined as many times as the value specified in SQLCVARN.
3	SQLCSNX	short	2	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the element number of the SQLCSN that has a character set name for input ? parameters.
				HiRDB	When the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued, the element number of the SQLCSN that has the character set name used in retrieval items or ? parameters is set. If the retrieval items or output ? parameters do not contain a character set specification, binary 0 is set. When the DESCRIBE INPUT or PREPARE statement is issued, the element number of the SQLCSN that has the character set name used in input ? parameters is set. If the input ? parameters do not contain a character set specification, binary 0 is set.

Level number[#] 1	Data area name	Data type	Length (bytes)	Source of value	Description
2	SQLCSN	--	64 x *m*	--	Area composed of SQLCSOL, SQLCSOC, SQLCSNL, and SQLCSNC areas. This area is for the information about one character set and must be defined as many times as the value specified in SQLCSNN.
3	SQLCSOL	short	2	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the length of the authorization identifier.
				HiRDB	When the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued, the length of the authorization identifier is set.
3	SQLCSOC	char	30	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the authorization identifier.
				HiRDB	When the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued, the authorization identifier is set.
3	SQLCSNL	short	2	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the length of the character set name.
				HiRDB	When the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued, the length of the character set name is set.
3	SQLCSNC	char	30	UAP	When the OPEN, EXECUTE, EXECUTE IMMEDIATE, or FETCH statement is issued, specifies the character set name.
				HiRDB	When the DESCRIBE, DESCRIBE CURSOR, or PREPARE statement is issued, the character set name is set.

Legend:

Δ : One space

--: Not applicable

#1

Level numbers indicate the set inclusion relationships of the character set descriptor area. For example, the level 1 data area is composed of level 2 data areas.

#2

Square brackets ([ ]) enclose the length for 64-bit mode. For 64-bit mode Windows, the length is the same as for 32-bit mode.

(3) Character set information that can be set in SQLCSN in the character set descriptor area

The following table shows the character set information that UAPs can set in `SQLCSN`.

Table E-2: Character set information that UAPs can set in SQLCSN

SQLCSOL	SQLCSOC	SQLCSNL	SQLCSNC
6	MASTER	6	EBCDIK#1
6	MASTER	8	UTF-16LE#2
6	MASTER	8	UTF-16BE#2
6	MASTER	5	UTF16#2

#1

This value can be used only when `sjis` is specified as the character code type by the `pdntenv` command (in the UNIX edition, `pdsetup` command).

#2

This value can be used only when `utf-8` is specified as the character code type by the `pdntenv` command (in the UNIX edition, `pdsetup` command).

If you use data descriptions in UTF-16 in C language or data descriptions containing Japanese data items (`PICTURE N`) in COBOL by using COBOL2002's Unicode functionality, you must specify the character sets UTF16, UTF-16LE, and UTF-16BE.

For details about the data descriptions in UTF-16 in C language, see *Appendix F.1 SQL data types and C data descriptions*. For details about the data descriptions containing Japanese data items (`PICTURE N`) in COBOL, see *F.2 SQL data types and COBOL data descriptions*.

The following table shows the attributes of the UTF16, UTF-16LE, and UTF-16BE character sets.

Table E-3: Attributes of the UTF16, UTF-16LE, and UTF-16BE character sets

Character set name	Usage format	Character repertoire
UTF-16LE	UTF-16, little endian	Unicode characters that can be coded by using UTF-16
UTF-16BE,UTF16	UTF-16, big endian	

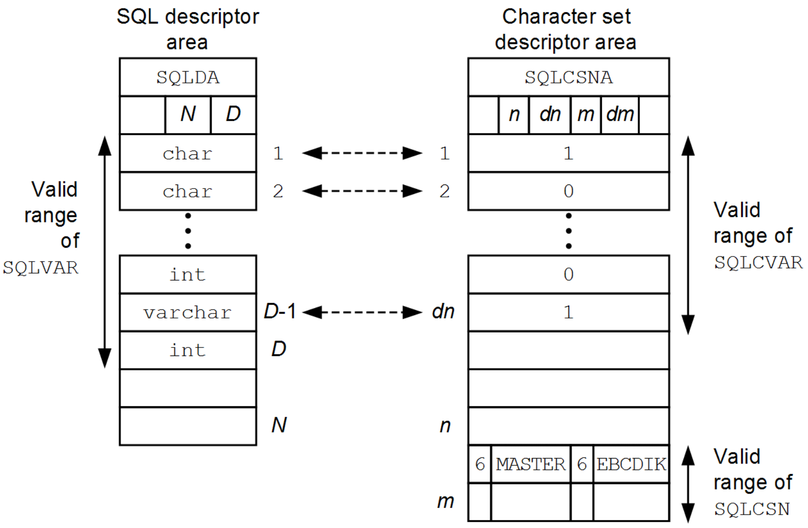
If you issue a `DESCRIBE`, `DESCRIBE TYPE`, `DESCRIBE CURSOR`, or `PREPARE` statement to process an SQL retrieval item or a `?` parameter whose resulting character set is UTF16, HiRDB sets UTF16 as the character set name in the character set descriptor area.

If you retrieve data from an SQL retrieval item or a `?` parameter whose resulting character set is UTF16 by using the `OPEN`, `EXECUTE`, `EXECUTE IMMEDIATE`, or `FETCH` statement, or if you pass data represented in UTF-16 to HiRDB by using a `?` parameter, you must specify UTF16, UTF-16LE, or UTF-16BE as the character set name.

(4) Relationship with the SQL descriptor area

The following figure shows the relationship between the SQL descriptor area and the character set descriptor area.

Figure E-2: Relationship between SQL descriptor area and character set descriptor area



Legend:

N: `SQLN` setting (maximum number of `SQLVAR`s)

D: `SQLD` setting (valid number of `SQLVAR`s) *n*: SQLCVARN setting (maximum number of SQLCVARs)

dn: SQLCVARD setting (valid number of SQLCVARs)

m: SQLCSNN setting (maximum value of SQLCSN)

dm: SQLCSND setting (valid value of SQLCSN)

E.2 Expanding the character set descriptor area

You allocate the character set descriptor area by declaring it within a UAP.

This section presents the format of a character set descriptor area that is expanded within a source program, and provides usage examples.

(1) Expansion format of character set descriptor area

(a) In C language

The following shows the expansion format of the character set descriptor area in C language:

```
struct {
     char    sqlcsnid[8];             /* name indicating SQLCSNA        */
     long    sqlcsnbc;                /* length of SQLCSNA              */
     short   sqlcvarn;     /* number of elements in the SQLCVAR array  */
     short   sqlcvard;                /* valid number of SQLCVARs       */
     short   sqlcsnn;        /* number of elements in the SQLCSN array */
     short   sqlcsnd;                 /* valid number of SQLCSNs        */
     char    sqlcsnex[12];            /* reserved                       */
     struct sqlcvar {
          short   sqlcsnx;            /* number of SQLCSN that has the  */
                                      /* corresponding character set name */
     } SQLCVAR[m]; #1
     struct sqlcsn {
        short   sqlcsol;        /* valid length of authorization identifier */
        char    sqlcsoc[30];       /* authorization identifier storage area */
        short   sqlcsnl;             /* valid length of character set name  */
        char    sqlcsnc[30];           /* character set name storage area   */
     } SQLCSN[n]; #2
} sqlcsna; #3
```

#1: m specifies the required number of items (1 to 30000).

#2: n specifies the required number of items (1).

#3: For a structure name (the sqlcsna part), specify any desired character string. A structure name cannot begin with SQL.

(b) In COBOL

The following shows the expansion format of the character set descriptor area in COBOL:

```
01  USQLCSNA.#1
    02  USQLCSNID        PIC  X(8)  VALUE  'SQLCSNA'.
    02  USQLCSNBC        PIC  S9(9)  COMP.#4
    02  USQLCVARN        PIC  S9(4)  COMP.
    02  USQLCVARD        PIC  S9(4)  COMP.
    02  USQLCSNN         PIC  S9(4)  COMP.
    02  USQLCSND         PIC  S9(4)  COMP.
    02  USQLCSNEX        PIC  X(12).
    02  USQLCVAR  OCCURS  m  TIMES. #2
      03  USQLCSNX       PIC  S9(4)  COMP.
    02  USQLCSN   OCCURS  n  TIMES. #3
      03  USQLCSOL       PIC  S9(4)  COMP.
      03  USQLCSOC       PIC  X(30).
      03  USQLCSNL       PIC  S9(4)  COMP.
      03  USQLCSNC       PIC  X(30).
```

#1: Any name can be specified for the set item (the USQLCSNA part). The data items cannot begin with SQL.

#2: m specifies the required number of items (1 to 30000).

#3: n specifies the required number of items (1).

#4: In 64-bit mode UNIX, change the PICTURE clause of USQLCSNBC to S9(18).

(2) How to expand the character set descriptor area

The following table describes how to expand the character set descriptor area.

Table E-4: How to expand the character set descriptor area

Language	Using an include file	Direct specification by the user
C	#include <pdbsqlcsna.h> PDUSRSQLCSNA(*m*,*n*) usrsqlcsna For details about *m*, *n*, and usrsqlcsna, see *(3) Character set descriptor area operation macros*.	The user directly codes the expanded format of the character set descriptor area.
COBOL	COPY SQLCSNA# [REPLACING 256 BY *m* ==OCCURS 1 == BY ==OCCURS *n* ==] .	The user directly codes the expanded format of the character set descriptor area. Coding must begin at the 01 level.

#: In 64-bit mode, change SQLCSNA to SQLCSNA64.

(3) Character set descriptor area operation macros

Various macros for declaring SQLCSNA and for setting/referencing values are defined in C language. These macros can be used by including a unique header file (pdbsqlcsna.h) in the UAP. Table E-5 lists the character set descriptor area operation macros and the macros for specifying character set names.

Table E-5: Character set descriptor area operation macros

Macro	Function
PDUSRSQLCSNA(m,n)	Declares a user SQLCSNA.
PDSETCSNASIZE(usrsqlcsna,m,n)	Specifies the size of the SQLCSNA.
PDSQLCVARN(usrsqlcsna)	Specifies the number of elements in an SQLCVAR array.
PDSQLCVARD(usrsqlcsna)	Specifies and references the valid number of SQLCVARs.
PDSQLCSNN(usrsqlcsna)	Specifies the number of elements in an SQLCSN array.
PDSQLCSND(usrsqlcsna)	Specifies and references the valid number of SQLCSNs.
PDSQLCSNX(usrsqlcsna,o)	Specifies and references the number assigned to the SQLCSN that has the corresponding character set name.
PDSQLCSOL(usrsqlcsna,p)	Specifies and references the length of the authorization identifier for a character set.
PDSQLCSOC(usrsqlcsna,p)	Specifies and references the authorization identifier for a character set.
PDSQLCSNL(usrsqlcsna,p)	Specifies and references the length of a character set's character set name.
PDSQLCSNC(usrsqlcsnaa,p)	Specifies and references a character set's character set name.

Legend:

usrsqlcsna: User-defined character set descriptor area. Specify any value.

m: Number of ? parameters (1 to 30000)

n: Number of character set names (0 or 1)

o: Number assigned to the ? parameter or retrieval item that is to be specified or referenced (0 to 29999)

p: Number assigned to the character set name that is to be specified or referenced (0)

Table E-6: Macros for specifying character set names

Macro	Function
PDSQL_CS_MASTER_L	Length of the authorization identifier `MASTER`
PDSQL_CS_MASTER_C	Character string of the authorization identifier `MASTER`
PDSQL_CS_EBCDIK_L	Length of the character set name `EBCDIK`
PDSQL_CS_EBCDIK_C	Character string of the character set name `EBCDIK`
PDSQL_CS_UTF16_L	Length of the character set name `UTF16`
PDSQL_CS_UTF16_C	Character string of the character set name `UTF16`
PDSQL_CS_UTF_16LE_L	Length of the character set name `UTF-16LE`
PDSQL_CS_UTF_16LE_C	Character string of the character set name `UTF-16LE`
PDSQL_CS_UTF_16BE_L	Length of the character set name `UTF-16BE`
PDSQL_CS_UTF_16BE_C	Character string of the character set name `UTF-16BE`

F. SQL Data Types and Data Descriptions

This appendix shows the correspondence between SQL data types and C or COBOL data descriptions.

F.1 SQL data types and C data descriptions

This section provides the correspondence between SQL data types and C data descriptions. Data can be exchanged between variables of compatible data types and between variables of either convertible or assignable data types.

(1) SQL data types and C data descriptions

The following table shows the SQL data types and C data descriptions.

Table F-1: SQL data types and C data descriptions

SQL data type	C data description	Remarks
SMALLINT	short *variable-name*;	--
INTEGER	long *variable-name*;	--
DECIMAL [(*p* [,*s*])]	SQL TYPE IS DECIMAL(*p*,*s*) *variable-name*;#5	$1 \leq p \leq 38, 0 \leq s \leq$ p
SMALLFLT, REAL	float *variable-name*;	--
FLOAT (DOUBLE PRECISION)	double *variable-name*;	--
CHAR [(*n*)] [CHARACTER SET *character-set-specification*]	char [CHARACTER SET[IS] *character-set-specification*] *variable-name*[*n*+1];#1	$1 \leq n \leq$ 30,000 *character-set-specification*: [MASTER.]EBCDIK
CHAR[(2*n*)] CHARACTER SET *character-set-specification*	char CHARACTER SET[IS] *character-set-specification* *variable-name*[2n+2];	$1 \leq n \leq$ 15,000 *character-set-specification*: [MASTER.]UTF16
	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] *character-set-specification* *character-set-specification* *variable-name*;#10	

SQL data type	C data description	Remarks
VARCHAR(*n*) [CHARACTER SET *character-set-specification*]	struct { short *variable-name-1*; char *variable-name-2*[*n*]; }[CHARACTER SET[IS] *character-set-specification*] *structure-name*;	1 ≤ *n* ≤ 32,000 *variable-name-1*: Character string length (bytes) *variable-name-2*: Character string *character-set-specification*: [MASTER.]EBCDIK
	SQL TYPE IS VARCHAR(*n*) [CHARACTER SET[IS] *character-set-specification*] *variable-name*;#6	
	VARCHAR [CHARACTER SET[IS] *character-set-specification*] *variable-name*[*n*+1]#9	
VARCHAR(2*n*) CHARACTER SET *character-set-specification*	struct { short *variable-name-1*; char *variable-name-2*[2*n*]; } CHARACTER SET[IS] *character-set-specification* *structure-name*;	1 ≤ *n* ≤ 16,000 *variable-name-1*: Character string length (number of bytes, as a multiple of 2) *variable-name-2*: Character string *character-set-specification*: [MASTER.]UTF16
	SQL TYPE IS VARCHAR(2n) CHARACTER SET[IS] *character-set-specification* *variable-name*;#11	
	VARCHAR CHARACTER SET[IS] *character-set-specification* *variable-name*[2n+2];#10	
NCHAR [(*n*)]	char *variable-name-2*[2*n*+1];#1	1 ≤ *n* ≤ 15,000

NVARCHAR(*n*)	struct {
	short *variable-name-1*;
	char *variable-name-2*[2*n*];
	} *structure-name*;
		1 ≤ *n* ≤ 16,000
		variable-name-1: Character string length (number of characters)
		variable-name-2: Character string

	SQL TYPE IS NVARCHAR(*n*) *variable-name*;#6	

SQL data type	C data description	Remarks
MCHAR [(*n*)]	char *variable-name*[*n*+1];[#1]	$1 \leq n \leq 30{,}000$

MVARCHAR(*n*)	struct {
	short *variable-name-1*;
	char *variable-name-2*[*n*];
	} *structure-name*;
		$1 \leq n \leq 32{,}000$
		variable-name-1: Character string length (number of bytes)
		variable-name-2: Character string

	SQL TYPE IS MVARCHAR(*n*) *variable-name*;[#6]	
DATE	char *variable-name*[11];[#2]	--
TIME	char *variable-name*[9];[#2]	--
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0) *variable-name*;[#5]	--
INTERVAL HOUR TO SECOND	SQL TYPE IS DECIMAL(6,0) *variable-name*;[#5]	--
TIMESTAMP[(*p*)]	char *variable-name*[*n* + 1];[#2]	If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.
ROW[#3]	char *variable-name*[*n* + 1];	$1 \leq$ total length $\leq 30{,}000$
BLOB	SQL TYPE IS BLOB(*n*[K|M|G])] *variable-name*;[#4]	Default: $1 \leq n \leq 2{,}147{,}483{,}647$ In units of K: $1 \leq n \leq 2{,}097{,}152$ In units of M: $1 \leq n \leq 2{,}048$ In units of G: $1 \leq n \leq 2$

BINARY(*n*)	struct {
	long *variable-name-1*;
	char *variable-name-2*[*n*];
	} *structure-name*;
		$1 \leq n \leq 2{,}147{,}483{,}647$
		variable-name-1: Character string length (number of bytes)
		variable-name-2: Character string

	SQL TYPE IS BINARY(*n*) *variable-name*;[#7]	
BLOB locator	SQL TYPE IS BLOB AS LOCATOR *variable-name*[#8]	--

SQL data type		C data description	Remarks
BINARY locator		SQL TYPE IS BINARY AS LOCATOR *variable-name*#8	--
Indicator variable	Other than BLOB, BINARY, BLOB locator, or BINARY locator	short *variable-name*;	--
	BLOB, BINARY, BLOB locator, or BINARY locator	long *variable-name*;	

SQL statement		struct {
		long *variable-name-1*;
		char *variable-name-2*[*n*];
		} *structure-name*;
			1 ≤ *n* ≤ 2,000,000
			variable-name-1: Character string length (number of bytes)
			variable-name-2: Character string

Legend:

--: Not applicable

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

#1: The following rules govern data conversion between SQL data types (`CHAR(`*n*`)`, `NCHAR(`*n*`)`, and `MCHAR(`*n*`)`) and C-language data types (`char[`*n*`+1]`, `char[2`*n*`+1]`, and `char[2`*n*`+1]`):

- For input (conversion from `char[`*n*`+1]` to `CHAR(`*n*`)`, conversion from `char[2`*n*`+1]` to `NCHAR(`*n*`)`, or conversion from `char[`*n*`+1]` to `MCHAR(`*n*`)`)

  The length of a fixed-length character string received by HiRDB from a C language-character string is equal to the length from the beginning of the character string to one character before the null character. If no null character is found in *n*`+1` array elements, the length is defined as *n*.

- For output (conversion from `CHAR(`*n*`)` to `char[`*n*`+1]`, conversion from `NCHAR(`*n*`)` to `char[2`*n*`+1]`, or conversion from `MCHAR(`*n*`)` to `char[`*n*`+1]`)

  A null character is appended at the end of the character string; therefore, the length of the character string known to the UAP is the SQL character string length + 1.

#2: When retrieving date data (`DATE`) using a dynamic SQL, the data code for the retrieval item information obtained by the `DESCRIBE` statement must be set as the character data type with a data length of at least 10 bytes. Similarly, when retrieving time data using a dynamic SQL, the data code for the retrieval item information obtained by the `DESCRIBE` statement must be set as the character data type with a data length of at least 8 bytes.

To retrieve time stamp data (`TIMESTAMP`) using a dynamic SQL statement, specify the following:

- Set the data code for the retrieval item information obtained using a `DESCRIBE` statement to the character data type.
- If *p* is 0, set the data size to 19 bytes or greater. If *p* is 2, 4, or 6, set the data size to 20 + *p* bytes or greater.

#3: The `ROW` type is allowed only when the HiRDB server and the HiRDB client use the same endian type.

#4: The coding of a BLOB UAP is expanded internally as follows:

```
struct{
       long           variable-name_reserved;            1
       unsigned long  variable-name_length;             2
       char           variable-name_data[m];             3
} variable-name
```

1. *variable-name*`_reserved` is not used. In the 64-bit mode, `int` *variable-name*`_reserved;` is used.
2. *variable-name*`_length` indicates the actual `BLOB` size. In the 64-bit mode, `unsigned int` *variable-name*`_length;` is used.
3. *variable-name*`_data[`*m*`]` is the BLOB data storage area (where *m* denotes the actual data length).

#5: The code for a DECIMAL UAP is internally expanded as follows:

```
unsigned char variable-name[↓p/2↓+1];
```

One byte of DECIMAL data expresses two numeric digits. A sign is expressed by four low-order bits of the trailing byte. Therefore, for DECIMAL data consisting of an even number of digits, four high-order bits of the leading byte must be padded with 0s. Do not use any number other than 0 for padding purposes.

The following table shows the standard sign representation; for details about the sign for DECIMAL type used with HiRDB, see the manual *HiRDB Version 9 SQL Reference*.

Sign in hexadecimal representation	Description
X'C'	Treated as a positive sign. Positive numbers include 0.
X'D'	Treated as a negative sign.

Coding examples

`123.4567` (odd number of digits)

```
unsigned char ex1[4]={0x12,0x34,0x56,0x7c};
```

`-123.456` (even number of digits)

```
unsigned char ex2[4]={0x01,0x23,0x45,0x6d};
```

`0` (odd number of digits)

```
unsigned char ex3[1]={0x0c};
```

#6: The following internal expansion takes place:

```
struct{
         short   len;
         char    str[n];
}variable-name
```

For `NVARCHAR`, `str[2n]` is used.

#7: The following internal expansion takes place:

```
struct{
         long  len;
         char  str[n];
}variable-name
```

In 64-bit mode, `long len;` is replaced with `int len;`.

#8: The following internal expansion takes place:
`unsigned long` *variable-name*`;`

In 64-bit mode, `unsigned long` *variable-name*`;` is replaced with `unsigned int` *variable-name*`;`.

#9: The following internal expansion takes place:

```
char variable-name[n+1];
```

The character string length is the length from the beginning of the string to the character before the NULL character. When a character string in C is accepted, an error occurs if there is no NULL character in the *n*+1-th array element.

#10

The following internal expansion takes place:

When the type specifier is not specified in the preprocessor's -XU16 option:

```
char variable-name[2n+2];
```

When the type specifier is not specified in the preprocessor's -XU16 option:

```
type-specifier  variable-name[n+1];
```

For details about the -XU16 option, see *8.2.2 Preprocessing in UNIX* or *8.2.3 Preprocessing in Windows*.

#11

The following internal expansion takes place:

When the type specifier is not specified in the preprocessor's -XU16 option:

```
struct{
         short  len;
         char   str[2n];
} variable-name;
```

When the type specifier is not specified in the preprocessor's -XU16 option:

```
struct{
         short     len;
         type-specifier  str[n];
} variable-name;
```

For details about the -XU16 option, see *8.2.2 Preprocessing in UNIX* or *8.2.3 Preprocessing in Windows*.

(2) SQL data types and C data descriptions when arrays are used

The following table shows the SQL data types and C data descriptions when arrays are used.

Table F-2: SQL data types and C data descriptions when arrays are used

SQL data type	C data description	Remarks
SMALLINT	short *variable-name*[*m*];	--
INTEGER	long *variable-name*[*m*];	--
DECIMAL[(*p*[,*s*])]	SQL TYPE IS DECIMAL(*p*,*s*) *variable-name*[*m*];	1 ≤ *p* ≤ 38, 0 ≤ *s* ≤ *p*
SMALLFLT, REAL	float *variable-name*[*m*];	--
FLOAT (DOUBLE PRECISION)	double *variable-name*[*m*];	--
CHAR[(*n*)] [CHARACTER SET *character-set-specification*]	char [CHARACTER SET[IS] *character-set-specification*] *variable-name*[*m*][*n*+1];	1 ≤ *n* ≤ 30,000 *character-set-specification*: [MASTER.]EBCDIK
CHAR[(2*n*)] CHARACTER SET *character-set-specification*	char CHARACTER SET[IS] *character-set-specification* *variable-name*[*m*][2*n*+2];	1 ≤ *n* ≤ 15,000 *character-set-specification*: [MASTER.]UTF16
	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] *character-set-specification* *variable-name*[*m*];	

VARCHAR(*n*)
[CHARACTER SET *character-set-specification*]
	struct {
	short *variable-name-1*;
	char *variable-name-2*[*n*];
	}[CHARACTER SET[IS]
	character-set-specification]
	structure-name[*m*];
		1 ≤ *n* ≤ 32,000
		character-set-specification:
		[MASTER.]EBCDIK

	SQL TYPE IS VARCHAR(*n*) [CHARACTER SET[IS] *character-set-specification*] *variable-name*[*m*];	--
	VARCHAR [CHARACTER SET[IS] *character-set-specification*] *variable-name*[*m*][*n*+1];	

SQL data type	C data description	Remarks

VARCHAR(2*n*)
CHARACTER SET
character-set-specification
	struct {
	short *variable-name-1*;
	char *variable-name-2*[2*n*];
	} CHARACTER SET[IS]
	character-set-specification
	structure-name[*m*];
		1 ≤ *n* ≤ 16,000
		variable-name-1: Character string length (number of bytes, as a multiple of 2)
		variable-name-2: Character string
		character-set-specification: [MASTER.]UTF16

	SQL TYPE IS VARCHAR(n) CHARACTER SET[IS] *character-set-specification* *variable-name*[*m*];	
	VARCHAR CHARACTER SET[IS] *character-set-specification* *variable-name*[*m*][2*n*+2]	
NCHAR[(*n*)]	char *variable-name*[*m*][2*n*+1];	1 ≤ *n* ≤ 15,000

NVARCHAR[(*n*)]	struct {
	short *variable-name-1*;
	char *variable-name-2*[2*n*];
	} *structure-name*[*m*];
		1 ≤ *n* ≤ 16,000

	SQL TYPE IS NVARCHAR(*n*) *variable-name*[*m*];	
MCHAR(*n*)	char *variable-name*[*m*][*n*+1];	1 ≤ *n* ≤ 30,000

MVARCHAR(*n*)	struct {
	short *variable-name-1*;
	char *variable-name-2*[*n*];
	} *structure-name*[*m*];
		1 ≤ *n* ≤ 32,000

	SQL TYPE IS MVARCHAR(*n*) *variable-name*[*m*];	
DATE	char *variable-name*[*m*][11];	--
TIME	char *variable-name*[*m*][9];	--
TIMESTAMP[(*p*)]	char *variable-name*[*m*][*n* + 1];	If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0) *variable-name*[*m*];	--

SQL data type		C data description	Remarks
`INTERVAL HOUR TO SECOND`		`SQL TYPE IS DECIMAL(6,0)` *variable-name*`[`*m*`];`	--
`ROW`		`char` *variable-name*`[`*m*`][`*n*`+1];`	1 ≤ *n* ≤ 30,000
`BLOB`		CN	--

`BINARY`		`struct {`
		`long` *variable-name-1*`;`
		`char` *variable-name-2*`[`*n*`];`
		`}` *structure-name*`[`*m*`];`
			- `FETCH` that uses an array 4 ≤ *n* ≤ 2,147,483,644 (*n* must be a multiple of 4.)
			- Other than `FETCH` that uses an array4 ≤ *n* ≤ 32,000 (*n* must be a multiple of 4.)

		`SQL TYPE IS BINARY(`*n*`)` *variable-name*`[`*m*`];`	
`BLOB` locator		CN	--

`BINARY` locator		`SQL TYPE IS`
		`BINARY AS LOCATOR`
		variable-name`[`*m*`];`
			--

Indicator variable	Other than `BINARY` or `BINARY` locator	`short` *variable-name*[*m*];	--
	`BINARY` or `BINARY` locator	`long` *variable-name*[*m*];	--
SQL statement		CN	--

Legend:

CN: Cannot be coded.

--: Not applicable

m: Number of array elements (1 to 4,096)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

(3) SQL data types and C data descriptions when repetition columns are used

The following table shows the SQL data types and C data descriptions when repetition columns are used.

Table F-3: SQL data types and C data descriptions when repetition columns are used

SQL data type	C data description	Remarks
SMALLINT	PD_MV_SINT(*m*) *variable-name*;	--
INTEGER	PD_MV_INT(*m*) *variable-name*;	--
DECIMAL	PD_MV_DEC(*m*,*p*,*s*) *variable-name*;	1 ≤ *p* ≤ 38, 0 ≤ *s* ≤ *p*
SMALLFLT, REAL	PD_MV_SFLT(*m*) *variable-name*;	--
FLOAT (DOUBLE PRECISION)	PD_MV_FLT(*m*) *variable-name*;	--
CHAR[(*n*)]	PD_MV_CHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 30,000
VARCHAR(*n*)	PD_MV_VCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 32,000
	PD_MV_CVCHAR(*m*,n) *variable-name*;	
NCHAR[(*n*)]	PD_MV_NCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 15,000
NVARCHAR[(*n*)]	PD_MV_NVCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 16,000
MCHAR(*n*)	PD_MV_CHAR(*m*, *n*) *variable-name*;	1 ≤ *n* ≤ 30,000
MVARCHAR(*n*)	PD_MV_CHAR(*m*, *n*) *variable-name*;	1 ≤ *n* ≤ 32,000
DATE	PD_MV_CHAR(*m*,10) *variable-name*;	--
TIME	PD_MV_CHAR(*m*,8) *variable-name*;	--
TIMESTAMP[(*p*)]	PD_MV_CHAR(*m*,*n*) *variable-name*;	If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.

SQL data type	C data description	Remarks
INTERVAL YEAR TO DAY	PD_MV_DEC(*m*,8,0) *variable-name*;	--
INTERVAL HOUR TO SECOND	PD_MV_DEC(*m*,6,0) *variable-name*;	--
ROW	CN	--
BLOB	CN	--
BINARY	CN	--
Indicator variable (other than BLOB, BINARY, BLOB locator, or BINARY locator)	PD_MV_SINT(*m*) *variable-name*;	--
SQL statement	CN	--

Legend:

CN: Cannot be coded.

--: Not applicable.

m: Maximum number of repetition array elements (2 to 30,000).

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

(4) Macros for referencing or setting embedded variables

Special macros for referencing or setting embedded variables for each data type are used in the SQL data type and C data description when repetition columns are used. The following table shows the macros for referencing or setting embedded variables.

Table F-4: Macros for referencing or setting embedded variables

SQL data type	Macro name	Data to be referenced or set	Data type
SMALLINT	PD_MV_SINT_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_SINT_DATA (*variable-name, m*)	Each repetition element	short

SQL data type	Macro name	Data to be referenced or set	Data type
INTEGER	PD_MV_INT_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_INT_DATA (*variable-name, m*)	Each repetition element	long#
DECIMAL[(*p*[,*s*])]	PD_MV_DEC_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_DEC_DATA (*variable-name*, *m*)	Start address of each repetition element in decimal	unsigned
SMALLFLT, REAL	PD_MV_SFLT_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_SFLT_DATA (*variable-name, m*)	Each repetition element	float
FLOAT (DOUBLE PRECISION)	PD_MV_FLT_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_FLT_DATA (*variable-name, m*)	Each repetition element	double
CHAR[(*n*)]	PD_MV_CHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_CHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
VARCHAR(*n*)	PD_MV_VCHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_VCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHAR_STR (*variable-name*)	Address of character string of each repetition element	char[ ]
	PD_MV_CVCHAR_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_CVCHAR_DATA(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
NCHAR[(*n*)]	PD_MV_NCHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_NCHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]

SQL data type	Macro name	Data to be referenced or set	Data type
NVARCHAR[(*n*)]	PD_MV_NVCHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_NVCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_NVCHAR_STR (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
MCHAR(*n*)	PD_MV_CHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_CHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
MVARCHAR(*n*)	PD_MV_VCHAR_CNT (*variable-name*)	Current repetition data element count	long#
	PD_MV_VCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHAR_STR (*variable-name, m*)	Address of character string of each repetition element	char[ ]
DATE	Same as CHAR(10)	--	--
TIME	Same as CHAR(8)	--	--
TIMESTAMP[(*p*)]	Same as CHAR(*n*) If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.	--	--
INTERVAL YEAR TO DAY	Same as DECIMAL(8,0)	--	--
INTERVAL HOUR TO SECOND	Same as DECIMAL(6,0)	--	--
Indicator variable	PD_MV_SINT_CNT (*variable-name*)	Indicator of the overall repetition column	long#
	PD_MV_SINT_DATA (*variable-name, m*)	Indicator of each repetition column element	short

Legend:

--: Not applicable

m: Number of each repetition column element (0 - *m*-1).

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

#: In 64-bit mode, the data type is `int`.

The following shows an example of macros used for referencing or setting embedded variables in repetition columns:

```
EXEC SQL BEGIN DECLARE SECTION;
char xname[5];
PD_MV_SINT(4) xmscore;
PD_MV_CHAR(4,5) xmsubject;
EXEC SQL END DECLARE SECTION;
     :
strcpy(xname,"SMITH")
PD_MV_SINT_DATA(xmscore,0)=90;
PD_MV_SINT_DATA(xmscore,1)=65;
PD_MV_SINT_DATA(xmscore,2)=85;
PD_MV_SINT_DATA(xmscore,3)=55;
PD_MV_SINT_CNT(xmscore)=4;
strcpy(PD_MV_CHAR_DATA(xmsubject,0),"MATHEMATICS");
strcpy(PD_MV_CHAR_DATA(xmsubject,1),"ENGLISH");
strcpy(PD_MV_CHAR_DATA(xmsubject,2),"SCIENCE");
strcpy(PD_MV_CHAR_DATA(xmsubject,3),"SOCIAL STUDIES");
PD_MV_CHAR_CNT(xmsubject)=4;
EXEC SQL
  INSERT INTO
SCORE_TABLE(NAME,SUBJECT,SCORE))VALUES(:xname,
:xmsubject;:xmscore);
```

(5) Pointer variables and the C language data description

The following table shows pointer variables and the C language data description.

Table F-5: Pointer variables and C language data description

SQL data type	C language data description	Remarks
SMALLINT	short *_variable-name_;	--
INTEGER	long *_variable-name_;	--
DECIMAL[(_p_[,_s_])]	SQL TYPE IS DECIMAL(_p_,_s_) *_variable-name_;	$1 \leq p \leq 38, 0 \leq s \leq p$
SMALLFLT, REAL	float *_variable-name_;	--
FLOAT (DOUBLE PRECISION)	double *_variable-name_;	--

SQL data type	C language data description	Remarks
CHAR[(n)] [CHARACTER SET character-set-specification]	char * [CHARACTER SET[IS] character-set-specification] *variable-name;	$1 \leq n \leq 30{,}000$# character-set-specification: [MASTER.]EBCDIK
CHAR[(2n)] CHARACTER SET character-set-specification	SQL TYPE IS CHAR(2n) CHARACTER SET[IS] character-set-specification *variable-name;	$1 \leq n \leq 15{,}000$ character-set-specification: [MASTER.]UTF16

VARCHAR(n)
[CHARACTER SET character-set-specification]
	struct {
	short variable-name-1;
	char variable-name-2[n];
	} [CHARACTER SET[IS]
	character-set-specification] *structure-name;
		$1 \leq n \leq 32{,}000$
		variable-name-1: Character string length (number of bytes)
		variable-name-2: Character string
		character-set-specification:
		[MASTER.]EBCDIK

	SQL TYPE IS VARCHAR(n) [CHARACTER SET[IS] character-set-specification] *variable-name;	
	VARCHAR [CHARACTER SET[IS] character-set-specification] *variable-name;#	

VARCHAR(2n)
CHARACTER SET character-set-specification
	struct {
	short variable-name-1;
	char variable-name-2[2n];
	} CHARACTER SET[IS]
	character-set-specification
	*structure-name;
		$1 \leq n \leq 16{,}000$
		variable-name-1: Character string length (number of bytes, as a multiple of 2)
		variable-name-2: Character string
		character-set-specification:
		[MASTER.]UTF16

	SQL TYPE IS VARCHAR(2n) CHARACTER SET[IS] character-set-specification *variable-name;	
NCHAR[(n)]	char *variable-name;	$1 \leq n \leq 15{,}000$#

SQL data type	C language data description	Remarks

NVARCHAR(*n*)	struct {
	short *variable-name-1*;
	char *variable-name-2*[2*n*];
	} **structure-name*;
		1 ≤ *n* ≤ 16,000
		variable-name-1: Character string length (number of characters)
		variable-name-2: Character string

	SQL TYPE IS NVARCHAR(*n*) **variable-name*;	
MCHAR[(*n*)]	char **variable-name*;	1 ≤ *n* ≤ 30,000#

MVARCHAR(*n*)	struct {
	short *variable-name-1*;
	char *variable-name-2*[*n*];
	} **structure-name*;
		1 ≤ *n* ≤ 32,000
		variable-name-1: Character string length (number of bytes)
		variable-name-2: Character string

	SQL TYPE IS MVARCHAR(*n*) **variable-name*;	
DATE#	char **variable-name*;	--
TIME#	char **variable-name*;	--
TIMESTAMP#	char **variable-name*;	--
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0) **variable-name*;	--
INTERVAL HOUR TO SECOND	SQL TYPE IS DECIMAL(6,0) **variable-name*;	--
ROW	char **variable-name*;	1 ≤ *total-length* ≤ 30,000#
BLOB	SQL TYPE IS BLOB(*n*[{K|M|G}]) **variable-name*;	Default: 1 ≤ *n* ≤ 2,147,483,647 In units of K: 1 ≤ *n* ≤ 2,097,152 In units of M: 1 ≤ *n* ≤ 2,048 In units of G: 1 ≤ *n* ≤ 2

BINARY(*n*)	struct {
	long *variable-name-1*;
	char *variable-name-2*[*n*];
	} **structure-name*;
		1 ≤ *n* ≤ 2,147,483,647

	SQL TYPE IS BINARY(*n*) **variable-name*;	

SQL data type		C language data description	Remarks
BLOB locator		SQL TYPE IS BLOB AS LOCATOR **variable-name*;	--
BINARY locator		SQL TYPE IS BINARY AS LOCATOR **variable-name*;	--
Indicator variable	Other than BLOB, BINARY, BLOB locator, or BINARY locator	short **variable-name*;	--
	BLOB, BINARY, BLOB locator, or BINARY locator	long **variable-name*;	

SQL statement		struct {
		long *variable-name-1*;
		char *variable-name-2*[*n*];
		} **structure-name*;
			$1 \leq n \leq$ 2,000,000

SMALLINT ARRAY *m*		PD_MV_SINT(*m*) **variable-name*;	--
INTEGER ARRAY *m*		PD_MV_INT(*m*) **variable-name*;	--
DECIMAL[(*p*[,*s*])] ARRAY *m*		PD_MV_DEC(*m*,*p*,*s*) **variable-name*;	$1 \leq p \leq 38, 0 \leq s \leq p$
SMALLFLT ARRAY *m* (REAL)		PD_MV_SFLT(*m*) **variable-name*;	--
FLOAT ARRAY *m* (DOUBLE PRECISION)		PD_MV_FLT(*m*) **variable-name*;	--
CHAR[(*n*)] ARRAY *m* and MCHAR[(*n*)] ARRAY *m*		PD_MV_CHAR(*m*,*n*) **variable-name*;	$1 \leq n \leq$ 30,000

SQL data type	C language data description	Remarks

`VARCHAR[(`*n*`)] ARRAY` *m* and `MVARCHAR[(`*n*`)] ARRAY` *m*	`PD_MV_VCHAR(`*m*`,`*n*`)`
	`*`*variable-name*`;`
		1 ≤ *n* ≤ 32,000

	`PD_MV_CVCHAR(`*m*`,`*n*`)`
	`*`*variable-name*`;`	

`NCHAR[(`*n*`)] ARRAY` *m*	`PD_MV_NCHAR(`*m*`,`*n*`)`
	`*`*variable-name*`;`
		1 ≤ *n* ≤ 15,000

`NVARCHAR[(`*n*`)] ARRAY` *m*	`PD_MV_NVCHAR(`*m*`,`*n*`)`
	`*`*variable-name*`;`
		1 ≤ *n* ≤ 16,000

`DATE ARRAY` *m*	`PD_MV_CHAR(`*m*`,10)`
	`*`*variable-name*`;`
		--

`TIME ARRAY` *m*	`PD_MV_CHAR(`*m*`,8)`
	`*`*variable-name*`;`
		--

`TIMESTAMP ARRAY` *m*	`PD_MV_CHAR(`*m*`,`*n*`)`
	`*`*variable-name*`;`
		If *p* = 0, *n* = 19.
		If *p* = 2, *n* = 21 or 22.
		If *p* = 4, *n* = 23 or 24.
		If *p* = 6, *n* = 25 or 26.

`INTERVAL YEAR TO DAY ARRAY` *m*	`PD_MV_DEC(`*m*`,8,0)`
	`*`*variable-name*`;`
		--

`INTERVAL HOUR TO SECOND ARRAY` *m*	`PD_MV_DEC(`*m*`,6,0)`
	`*`*variable-name*`;`
		--

Indicator variable for repetition column	`PD_MV_SINT(`*m*`)`
	`*`*variable-name*`;`
		--

Legend:

--: Not applicable

m: Number (0 - *m*-1) indicating each element in a repetition column

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

#: The defined length of the area cannot be determined during preprocessing. Therefore, at the time of execution, use `strlen(`*variable-name*`)` to determine the length of the character string stored in the area indicated by the pointer, and use this length in place of the area length. To receive the retrieval result, use a character other than NULL character to clear the area indicated by the pointer and enter the NULL character at the end.

(6) Macros for pointer-type repetition columns

To reference or set a variable for a pointer-type repetition column, use a dedicated macro. The following table shows the macros for pointer-type repetition columns.

Table F-6: Macros for pointer-type repetition columns

SQL data type	Macro name	Data to be referenced or set	Data type
SMALLINT ARRAY *m*	PD_MV_SINTP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_SINTP_DATA(*variable-name*, *m*)	Each repetition element	short
INTEGER ARRAY m	PD_MV_INTP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_INTP_DATA(*variable-name*,*m*)	Each repetition element	long#
DECIMAL[(*p*[,*s*])] ARRAY *m*	PD_MV_DECP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_DECP_DATA(*variable-name*,*m*)	Start address of each repetition element in decimal	char[ ]
SMALLFLT ARRAY *m* (REAL)	PD_MV_SFLTP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_SFLTP_DATA(*variable-name*, *m*)	Each repetition element	float
FLOAT ARRAY *m* (DOUBLE PRECISION)	PD_MV_FLTP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_FLTP_DATA(*variable-name*,*m*)	Each repetition element	double
CHAR[(*n*)] ARRAY *m*, or MCHAR[(*n*)] ARRAY *m*	PD_MV_CHARP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_CHARP_DATA(*variable-name*, *m*)	Leading address of character string of each repetition element	char[ ]

SQL data type	Macro name	Data to be referenced or set	Data type
VARCHAR(*n*) ARRAY *m*, or MVARCHAR(*n*) ARRAY *m*	PD_MV_VCHARP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_VCHARP_LEN(*variable-name*, *m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHARP_STR(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
	PD_MV_CVCHARP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_CVCHARP_DATA(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
NCHAR[(*n*)] ARRAY *m*	PD_MV_NCHARP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_NCHARP_DATA(*variable-name*,*m*)	Leading address of character string of each repetition element	char[ ]
NVARCHAR(*n*) ARRAY *m*	PD_MV_NVCHARP_CNT(*variable-name*)	Current repetition data element count	long#
	PD_MV_NVCHARP_LEN(*variable-name*,*m*)	Actual length of character string of each repetition element	short
	PD_MV_NVCHARP_STR(*variable-name*,*m*)	Leading address of character string of each repetition element	char[ ]
DATE ARRAY *m*	Same as CHAR(10)	--	--
TIME ARRAY *m*	Same as CHAR(8)	--	--
TIMESTAMP[(*p*)] ARRAY *m*	Same as CHAR(*n*) If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.	--	--
INTERVAL YEAR TO DAY	Same as DECIMAL(8,0)	--	--
INTERVAL HOUR TO SECOND	Same as DECIMAL(6,0)	--	--

SQL data type	Macro name	Data to be referenced or set	Data type
Indicator variable	`PD_MV_SINTP_CNT(`*variable-name*`)`	Indicator of the overall repetition column	`long`#
	`PD_MV_SINTP_DATA(`*variable-name*`,` *m*`)`	Indicator of each repetition column element	`short`

Legend:

--: Not applicable

m: Number of each repetition column element (0 - *m*-1)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

#: In 64-bit mode, the data type is `int`.

(7) Structures to be specified in batches

The following table shows the structures to be specified in batches.

Table F-7: Structures to be specified in batches

SQL data type	C language data description	Item coding	Remarks
Multiple items	Structure that contains the data types listed in Tables F-1 to F-3 as members	Specifies multiple embedded variables in a batch.	Pointers can be declared.
Indicator variable for multiple items	Structure that contains as members the indicator variables listed in Tables F-1 to F-3	Specifies multiple indicator variables in a batch.	Pointers can be declared.

F.2 SQL data types and COBOL data descriptions

This section provides the correspondence between SQL data types and COBOL data descriptions.

Data can be exchanged between variables of compatible data types and between variables of either convertible or assignable data types.

(1) SQL data types and COBOL data descriptions

The table below shows the SQL data types and COBOL data descriptions. Note that the data descriptions in these tables can also be coded as follows:

```
PICTURE:
    PIC
COMPUTATIONAL:
    COMP
COMPUTATIONAL-n:
    COMP-n
9(n):
    99   9
X(n):
    XX   X
OCCURS n TIMES:
    OCCURS 1 TO n TIMES 0
    OCCURS 1 TO n
    OCCURS n
```

Table F-8: SQL data types and COBOL data descriptions

SQL data type	COBOL data description	Item coding	Remarks

`SMALLINT`	`L1` *elementary-item-name*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
		elementary-item or *independent-item*	--

`INTEGER`	`L1` *elementary-item-name*
	`PICTURE S9(9)`
	`COMPUTATIONAL.`
		elementary-item or *independent-item*	--

`DECIMAL`
`[(`*p*`[,`*s*`])]`
	`L1` *elementary-item-name*
	`PICTURE S9(`*p-s*`) [V9(`*s*`)]`
	`COMPUTATIONAL-3.`
		elementary-item or *independent-item*	If 1 ≤ *p* ≤ 38#10, 0 ≤ *s* ≤ *p*, and *p* = *s*, then `SV9(`*s*`)`.
			If *s* = 0, then `[V9(`*s*`)]` is omitted.

	`L1` *elementary-item-name*
	`PICTURE S9(`*p-s*`) [V9(`*s*`)]`
	`DISPLAY SIGN`
	`LEADING SEPARATE.`#9, #12		

SQL data type	COBOL data description	Item coding	Remarks
	L1 *elementary-item-name* PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING.[#11, #12, #13]		
SMALLFLT (REAL)	L1 *elementary-item-name* COMPUTATIONAL-1.	*elementary-item* or *independent-item*	--
FLOAT (DOUBLE PRECISION)	L1 *elementary-item-name* COMPUTATIONAL-2.	*elementary-item* or *independent-item*	--
CHAR [(*n*)] [CHARACTER SET [MASTER.]EBCDIK]	L1 *elementary-item-name* [CHARACTER SET[IS] [MASTER.]EBCDIK] PICTURE X(*n*).[#5]	*elementary-item* or *independent-item*	1 ≤ *n* ≤ 30,000
CHAR[(2n)] CHARACTER SET [MASTER.]UTF16	If the HiRDB server's default character set is not UTF-8:[#15] CHAR type cannot be used. If the HiRDB server's default character set is UTF-8:[#15] L1 *elementary-item-name* PICTURE N(n).	*elementary-item* or *independent-item*	1 ≤ *n* ≤ 15,000
VARCHAR (*n*) [CHARACTER SET [MASTER.]EBCDIK]	L2 *group-item-name* [CHARACTER SET[IS] [MASTER.]EBCDIK]. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).[#5]	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* (number of bytes) *elementary-item-name-2*: *character-string*	1 ≤ *n* ≤ 32,000
VARCHAR(2n) CHARACTER SET [MASTER.]UTF16	If the HiRDB server's default character set is not UTF-8:[#15] VARCHAR type cannot be used. If the HiRDB server's default character set is UTF-8:[#15] L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(n).	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* (number of bytes) *elementary-item-name-2*: *character-string*	1 ≤ *n* ≤ 16,000

SQL data type	COBOL data description	Item coding	Remarks
`NCHAR [(`*n*`)]`	If the HiRDB server's default character set is not UTF-8:#15 `L1` *elementary-item-name* `PICTURE N(`*n*`).` If the HiRDB server's default character set is UTF-8:#15 `NCHAR` type cannot be used.	*elementary-item* or *independent-item*	1 ≤ *n* ≤ 15,000

`NVARCHAR (`*n*`)`	If the HiRDB server's default character set is not UTF-8:#15
	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE N(`*n*`)`
	
	If the HiRDB server's default character set is UTF-8:#15
	`NVARCHAR` type cannot be used.
		A group item composed of two elementary items
		elementary-item-name-1:
		character-string-length
		elementary-item-name-2:
		character-string
			1 ≤ *n* ≤ 16,000

`MCHAR [(`*n*`)]`	`L1` *elementary-item-name* `PICTURE X(`*n*`).`#6 If the HiRDB server's default character set is UTF-8,#15 this can be coded as follows: `L1` *elementary-item-name* `PICTURE N(`n_2`).`#14	*elementary-item or independent-item*	1 ≤ *n* ≤ 30,000

SQL data type	COBOL data description	Item coding	Remarks
MVARCHAR (*n*)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).[#6] If the HiRDB server's default character set is UTF-8,[#15] this can be coded as follows: L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(n_2).[#14]	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* *elementary-item-name-2*: *character-string*	1 ≤ *n* ≤ 32,000
DATE	L1 *elementary-item-name* PICTURE X(10).[#6]	*elementary-item* or *independent-item*	--
TIME	L1 *elementary-item-name* PICTURE X(8).[#6]	*elementary-item* or *independent-item*	--
TIMESTAMP[(*p*)]	L1 *elementary-item-name* PICTURE X(*n*).[#6]	*elementary-item* or *independent-item*	If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If p = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	L1 *elementary-item-name* PICTURE S9(8) COMPUTATIONAL-3.	*elementary-item* or *independent-item*	--
INTERVAL HOUR TO SECOND	L1 *elementary-item-name* PICTURE S9(6) COMPUTATIONAL-3.	*elementary-item* or *independent-item*	--
ROW[#3]	Combination of data items and group items in this table[#1]	A group item composed of elementary items	1 ≤ *total-length* ≤ 30,000

SQL data type		COBOL data description	Item coding	Remarks
BLOB		L2 *group-item-name*[#2] [USAGE [IS]] SQL TYPE IS BLOB(*n*[K |M |G]).[#4, #7]	*elementary-item*	Default: 1 ≤ *n* ≤ 2,147,483,647 In units of K: 1 ≤ *n* ≤ 2,097,152 In units of M: 1 ≤ *n* ≤ 2,048 In units of G: 1 ≤ *n* ≤ 2
BINARY(*n*)		L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).[#5, #7]	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* *elementary-item-name-2*: *character-string* *character-string-length* is the byte count.	1 ≤ *n* ≤ 2,147,483,647
BLOB locator		L1 *elementary-item-name* SQL TYPE IS BLOB AS LOCATOR.[#8]	*elementary-item* or *independent-item*	--
BINARY locator		L1 *elementary-item-name* SQL TYPE IS BINARY AS LOCATOR.[#8]	*elementary-item* or *independent-item*	--
Indicator variable	Other than BLOB, BINARY, BLOB locator, or BINARY locator	L1 *elementary-item-name* PICTURE S9(4) COMPUTATIONAL.	*elementary-item* or *independent-item*	--
	BLOB, BINARY, BLOB locator, or BINARY locator	L1 *elementary-item-name* PICTURE S9(9) COMPUTATIONAL.		

SQL data type	COBOL data description	Item coding	Remarks

SQL statement	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE X(`*n*`)`
		A group item composed of two elementary items
		elementary-item-name-1:
		character-string-length
		elementary-item-name-2:
		character-string
			1 ≤ *n* ≤ 2,000,000

Legend:

--: Not applicable

`L1`: Level number 01-49 or 77

`L2`: Level number 01-48

`L3`: Level number 02-49 (`L2 < L3`)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

#1: The following clauses can be used:

- `REDEFINES`
- `OCCURS`
- `ADDRESSED BY`

If you manipulate a table with the `FIX` attribute in units of rows, you must specify a data item that is to store a row of data as a group item. Each column of the target table must correspond to a data item belonging to the group item. To specify a data item that corresponds to a column, you use the data description that corresponds to the data type of that column. Note that a data description that requires data format conversion cannot be used.

The following table shows the data descriptions that require data format conversion and the substitute data descriptions that can be used:

Column's data type	Data description that is not available because it requires data format conversion	Substitute data description that can be used
DECIMAL	`DISPLAY` (external decimal format)	COMP-3 (internal decimal format)
MCHAR	`PICTURE N` (Japanese data item)	PICTURE X (alphanumeric item)

Column's data type	Data description that is not available because it requires data format conversion	Substitute data description that can be used
DATE	PICTURE X(10) 10: Number of characters in character string representation	PICTURE X(4) 4: Number of bytes in the format X'YYYYMMDD'
TIME	PICTURE X(8) 8: Number of characters in character string representation	PICTURE X(3) 3: Number of bytes in the format X'hhmmss'
TIMESTAMP(p)	PICTURE X(*n*) *n*: Number of characters in character string representation	PICTURE X(n) n=7 + p/2 *n*: Number of bytes in the format X'YYYYMMDDhhmmss[*nn*...*n*]'

Note

You can use the REDEFINES clause to redefine the data items corresponding to the columns of the target table. In such a case, you can specify an item to be redefined regardless of the data type of the corresponding column. However, you can use only those elementary items that correspond to the INTEGER, SMALLINT, DECIMAL, FLOAT SMALLFLT, CHAR, and NCHAR types shown in *Table F-8* and the group items consisting of these elementary items. For these elementary and group items, you can specify the OCCURS clause.

#2:

A group item name should be coded as no more than 21 characters. However, for COBOL2002, a group item name should be 22 characters or less.

#3:

The ROW type is allowed only when the HiRDB server and the HiRDB client use the same endian type.

#4: The coding of a BLOB UAP is expanded internally as follows:

```
L2 group-item-name.
 49 group-item-name_RESERVED PIC S(9) USAGE IS BINARY. 1
 49 group-item-name_LENGTH   PIC S(9) USAGE IS BINARY. 2
 49 group-item-name_DATA     PIC X(m).                 3
```

1. *group-item-name*_RESERVED is not used.
2. *group-item-name*_LENGTH is equal to the BLOB actual length.
3. *group-item-name*_DATA is the BLOB data storage area (where *m* denotes the

actual data length).

#5:

This item can be defined using 9 in place of X. If 9 is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

#6:

Do not use 9 for X during definition, although using 9 does not cause an error during preprocessing.

#7:

The maximum value that can be declared depends on the installed COBOL compiler. For details, see the manual for the COBOL compiler to be used.

#8:

The following internal expansion takes place:

```
L1 elementary-item-name PICTURE S9(9) COMPUTATIONAL.
```

#9:

The HiRDB server's data type is DECIMAL, but it is expressed as an external decimal item whose first byte is an operation sign.

#10:

The value range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

#11

The HiRDB server's data type is DECIMAL, but it is expressed as an external decimal item (zoned format) that has an operation sign in the 4 high-order bits of the rightmost byte.

#12

If a data item format is specified for a numeric item with an operation sign, DISPLAY (external decimal item) is assumed. If the SIGN clause is not specified for an external decimal item with an operation sign, the location and format of the operation sign are treated as being the same as in the DISPLAY SIGN TRAILING format.

#13

The DISPLAY SIGN TRAILING format is supported only in COBOL85 and COBOL2002.

#14

This data description containing a Japanese data item (PICTURE N) of COBOL is permitted as a data description corresponding to the SQL mixed character string data type (MCHAR or MVARCHAR) only if the Unicode functionality of COBOL2002 is used.

For details about UAP execution using the Unicode functionality of COBOL2002, see *8.4.3 UAP execution using the Unicode functionality of COBOL2002*.

You can use this data description in embedded variable declarations and the declarations of data areas for storing ? parameter values.

For embedded variables:

The data is treated as having a character data type (CHAR or VARCHAR) for which UTF-16LE or UTF-16BE is specified as the character set name according to the -XU16 option specification. The following table shows the details:

Data description format	Data description in COBOL	Attribute of embedded variable	
		Data type	Character set name
Fixed length	L1 *elementary-item-name* PICTURE N(n_2) .	CHAR(m) $m = 2 \times n_2$	UTF-16LE or UTF-16BE
Variable length	L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(n_2) .	VARCHAR(m) $m = 2 \times n_2$	UTF-16LE or UTF-16BE

Note: For variable-length *elementary-item-name-1*, specify the length (in bytes) of the Japanese-language character string to be stored in *elementary-item-name-2*. Because the character codes for data to be stored in a Japanese data item are UTF-16 (within the range of UCS-2), the length of the Japanese language character string in bytes is twice the number of characters. This length in bytes is 2 x n_2 or less.

For data areas for storing ? parameter values:

If you use this data description in the declaration of a data area for storing a ? parameter value, you must set the SQL descriptor area and character set descriptor area as shown in the following table:

Data description format	Data description in COBOL	Setting in SQL descriptor area		Setting in character set descriptor area
		Data code	Data length	
Fixed length	`L1` *elementary-item-name* `PICTURE N(`n_2`).`	Data code of `CHAR` type	2 x n_2	`UTF-16LE` or `UTF-16BE`
Variable length	`L2` *group-item-name*. `L3` *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. `L3` *elementary-item-name-2* `PICTURE N(`n_2`).`	Data code of `VARCHAR` type	2 x n_2	`UTF-16LE` or `UTF-16BE`

Note: For variable-length *elementary-item-name-1*, specify the length in bytes of the Japanese-language character string to be stored in *elementary-item-name-2*. Because the character codes for data to be stored in a Japanese data item are UTF-16 (within the range of UCS-2), the length of the Japanese language character string in bytes is twice the number of characters. This length in bytes is 2 x n_2 or less.

For details about the character set names, see Appendix *E.1(3) Character set information that can be set in SQLCSN in the character set descriptor area*.

#15

If you specify `utf-8` as the character code type in the `pdntenv` command (in UNIX edition, `pdsetup` command), the HiRDB server's default character set is set to UTF-8.

(2) SQL data types and COBOL data descriptions when arrays are used

The following table shows the SQL data types and COBOL data descriptions when arrays are used.

Table F-9: SQL data types and COBOL data descriptions when arrays are used

SQL data type	COBOL data description	Item coding	Remarks
SMALLINT	L2 *elementary-item-name* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.	A group item composed of repetitions of data items in which the same data structure is repeated through specification of OCCURS	--
INTEGER	L2 *elementary-item-name* PICTURE S9(9) OCCURS *m* TIMES.		--
DECIMAL [(*p*[,*s*])]	L2 *elementary-item-name* PICTURE S9 (*p*-*s*)[V9(*s*)] COMPUTATIONAL-3 OCCURS *m* TIMES.		1 ≤ *p* ≤ 38[#3], 0 ≤ *s* ≤ *p* If *p* = *s*, SV9(*s*) is used. If *s* = 0, [V9(*s*)] is omitted.
	L2 *elementary-item-name* PICTURE S9(*p*-*s*)[V9(*s*)] DISPLAY SIGN LEADING SEPARATE OCCURS *m* TIMES.		
	L2 *elementary-item-name* PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING OCCURS m TIMES.		--
SMALLFLT (REAL)	L2 *elementary-item-name* COMPUTATIONAL-1 OCCURS *m* TIMES.		--
FLOAT (DOUBLE PRECISION)	L2 *elementary-item-name* COMPUTATIONAL-2 OCCURS *m* TIMES.		--
CHAR [(*n*)] [CHARACTER SET [MASTER.]EBCDIK]	L2 *elementary-item-name* [CHARACTER SET[IS] [MASTER.]EBCDIK] PICTURE X(*n*) OCCURS *m* TIMES.[#1]		1 ≤ *n* ≤ 30,000

SQL data type	COBOL data description	Item coding	Remarks
CHAR[(2n)] CHARACTER SET [MASTER.]UTF16	If the HiRDB server's default character set is not UTF-8:#5 CHAR type cannot be used. If the HiRDB server's default character set is UTF-8:#5 L2 *elementary-item-name* PICTURE N(n) OCCURS m TIMES.		1 ≤ *n* ≤ 15,000
VARCHAR (*n*) [CHARACTER SET [MASTER.]EBCDIK]	L2 *group-item-name* [CHARACTER SET[IS] [MASTER.]EBCDIK] OCCURS *m* TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).#1		1 ≤ *n* ≤ 32,000
VARCHAR(2n) CHARACTER SET [MASTER.]UTF16	If the HiRDB server's default character set is not UTF-8:#5 VARCHAR type cannot be used. If the HiRDB server's default character set is UTF-8:#5 L2 *group-item-name-2* OCCURS m TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(n).		1 ≤ *n* ≤ 16,000
NCHAR [(*n*)]	If the HiRDB server's default character set is not UTF-8:#5 L2 *elementary-item-name* PICTURE N(*n*) OCCURS *m* TIMES. If the HiRDB server's default character set is UTF-8:#5 NCHAR type cannot be used.		1 ≤ *n* ≤ 15,000

SQL data type	COBOL data description	Item coding	Remarks
NVARCHAR (*n*)	If the HiRDB server's default character set is not UTF-8:#5 L2 *group-item-name* OCCURS *m* TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(*n*) If the HiRDB server's default character set is UTF-8:#5 NVARCHAR type cannot be used.		1 ≤ *n* ≤ 16,000
MCHAR [(*n*)]	L2 *elementary-item-name* PICTURE X(*n*) OCCURS *m* TIMES.#2 If the HiRDB server's default character set is UTF-8,#5 this can be coded as follows: L2 *elementary-item-name*L2 *elementary-item-name* PICTURE N(n_2) OCCURS m TIMES.#4		1 ≤ *n* ≤ 30,000
MVARCHAR (*n*)	L2 *group-item-name-2* OCCURS *m* TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).#2 If the HiRDB server's default character set is UTF-8,#5 this can be coded as follows: L2 *group-item-name-2* OCCURS m TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* L3 *elementary-item-name-2* PICTURE N(n_2).#4		1 ≤ *n* ≤ 32,000

SQL data type	COBOL data description	Item coding	Remarks
DATE	L2 *elementary-item-name* PICTURE X(10) OCCURS *m* TIMES.[#2]		--
TIME	L2 *elementary-item-name* PICTURE X(8) OCCURS *m* TIMES.[#2]		--
TIMESTAMP(*p*)	L2 *elementary-item-name* PICTURE X(*n*) OCCURS *m* TIMES.[#2]		If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	L2 *elementary-item-name* PICTURE S9(8) COMPUTATIONAL-3 OCCURS *m* TIMES.		--
INTERVAL HOUR TO SECOND	L2 *elementary-item-name* PICTURE S9(6) COMPUTATIONAL-3 OCCURS *m* TIMES.		--
ROW	L2 *group-item-name-2* OCCURS *m* TIMES. Combination of data items and group items in this table[#6]		--
BLOB	CN	CN	--

SQL data type		COBOL data description	Item coding	Remarks
BINARY		L2 *group-item-name-2* OCCURS m TIMES. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).#1	A group item composed of repetitions of data items in which the same data structure is repeated through specification of OCCURS.	• FETCH that uses an array 4 ≤ *n* ≤ 2,147,483,644 (*n* must be a multiple of 4.) • Other than FETCH that uses an array 4 ≤ *n* ≤ 32,000 (*n* must be a multiple of 4.)
BLOB locator		--	--	--
BINARY locator		L2 *elementary-item-name* SQL TYPE IS BINARY AS LOCATOR OCCURS *m* TIMES.	Group item consisting of iterative data items that repeat the same data structure according to the OCCURS specification	--
Indicator variable	Other than BINARY or BINARY locator	L2 *elementary-item-name* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.		--
	BINARY or BINARY locator	L2 *elementary-item-name* PICTURE S9(9) COMPUTATIONAL OCCURS *m* TIMES.		--
SQL statement		CN	CN	--

Legend:

CN: Cannot be coded.

--: Not applicable.

`L2`: Level number 02-49 (`L2 < L3`). You cannot specify level number 01, 66, 77, or 88 for `L2`. For details, see the syntax rules for the `OCCURS` clause in the COBOL manual.

`L3`: Level number 03-49

m: Number of array elements (1-4,096)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

#1:

This item can be defined using `9` in place of `X`. If `9` is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

#2:

Do not use `9` for `X` during definition, although using `9` does not result in an error during preprocessing.

#3:

The range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

#4

This data description containing a Japanese data item (`PICTURE N`) of COBOL is permitted as a data description corresponding to the SQL mixed character string data type (`MCHAR` or `MVARCHAR`) only if the Unicode functionality of COBOL2002 is used. For details, see the notes on `MCHAR[(`*n*`)]` and `MVARCHAR(`*n*`)` in *Table F-8 SQL data types and COBOL data descriptions*.

#5

If you specify `utf-8` as the character code type in the `pdntenv` command (in UNIX edition, `pdsetup` command), the HiRDB server's default character set is set to UTF-8.

#6

For details about the combinations, see *Table F-8*.

(3) SQL data types and COBOL data descriptions when repetition columns are used

The following table shows the SQL data types and COBOL data descriptions when repetition columns are used.

Table F-10: SQL data types and COBOL data descriptions when repetition columns are used

SQL data type	COBOL data description	Item coding	Remarks
SMALLINT	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.	A group item composed of two elementary items	--
INTEGER	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(9) COMPUTATIONAL OCCURS *m* TIMES.		--
DECIMAL [(*p*[,*s*])]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9 (*p-s*)[V9(*s*)] COMPUTATIONAL-3 OCCURS *m* TIMES.		$1 \leq p \leq 38$[#3], $0 \leq s \leq p$ When *p* = *s*, SV9(*s*) is used. When *s* = 0, [V9(*s*)] is omitted.
	L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(*p-s*)[V9(*s*)] DISPLAY SIGN LEADING SEPARATE OCCURS *m* TIMES.		
	L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(p-s)[V9(s)] DISPLAY SIGN TRAILING OCCURS m TIMES.		

SQL data type	COBOL data description	Item coding	Remarks
SMALLFLT (REAL)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* COMPUTATIONAL-1 OCCURS *m* TIMES.		--
FLOAT (DOUBLE PRECISION)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* COMPUTATIONAL-2 OCCURS *m* TIMES.		--
CHAR [(*n*)]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*) OCCURS *m* TIMES.[#1]		1 ≤ *n* ≤ 30,000
VARCHAR (*n*)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* OCCURS *m* TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE X(*n*).[#1]	A group item composed of two elementary items and a group item composed of one elementary item.	1 ≤ *n* ≤ 32,000
NCHAR [(*n*)]	If the HiRDB server's default character set is not UTF-8:[#5] L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(*n*) OCCURS *m* TIMES. If the HiRDB server's default character set is UTF-8:[#5] NCHAR type cannot be used.	A group item composed of two elementary items.	1 ≤ *n* ≤ 15,000

SQL data type	COBOL data description	Item coding	Remarks
NVARCHAR (*n*)	If the HiRDB server's default character set is not UTF-8:[#5] L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* OCCURS *m* TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE N(*n*). If the HiRDB server's default character set is UTF-8:[#5] NVARCHAR type cannot be used.	A group item composed of two elementary items and a group item composed of one elementary item.	1 ≤ *n* ≤ 16,000
MCHAR [(*n*)]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*) OCCURS *m* TIMES.[#1] If the HiRDB server's default character set is UTF-8,[#5] this can be coded as follows: L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(n2) OCCURS m TIMES.[#4]	A group item composed of two elementary items.	1 ≤ *n* ≤ 30,000

SQL data type	COBOL data description	Item coding	Remarks
MVARCHAR (*n*)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* OCCURS *m* TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE X(*n*).[#1] If the HiRDB server's default character set is UTF-8,[#5] this can be coded as follows: L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *group-item-name-2* OCCURS m TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE N(n_2).[#4]	A group item composed of two elementary items and a group item composed of one elementary item.	$1 \leq n \leq$ 32,000

SQL data type	COBOL data description	Item coding	Remarks
DATE	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(10) OCCURS *m* TIMES.#2	A group item composed of two elementary items.	--
TIME	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(8 OCCURS *m* TIMES.#2		--
TIMESTAMP[(*n*)]	L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*) OCCURS *m* TIMES.#2		If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(8) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(8) COMPUTATIONAL-3 OCCURS *m* TIMES.		--
INTERVAL HOUR TO SECOND	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(6) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(6) COMPUTATIONAL-3 OCCURS *m* TIMES.		--
ROW	CN	CN	--
BLOB	CN	CN	--
BINARY	CN	CN	--
BLOB locator	CN	CN	--
BINARY locator	CN	CN	--

SQL data type	COBOL data description	Item coding	Remarks
Indicator variable (other than BLOB, BINARY, BLOB locator, or BINARY locator)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.	A group item composed of two elementary items.	--
SQL statement	CN	CN	--

Legend:

CN: Cannot be coded.

--: Not applicable.

L2: Level number 02-49

L3 and L4: Level number 03-49

m: Maximum number of repetition column elements (2 to 30,000)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Notes

1. The value of *elementary-item-name-1* must be the current element count.
2. The values of *elementary-item-name-2* and *group-item-name-2* must be specified as the value of each repetition element.
3. *elementary-item-name-1* of the indicator variable must be specified as the indicator of the entire repetition column.
4. *elementary-item-name-2* of the indicator variable must be specified as the indicator of each repetition column element.

#1:

This item can be defined using 9 in place of X. If 9 is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

#2:

Do not use 9 for X during definition, although using 9 does not result in an error during preprocessing.

#3:

The range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

#4

This data description containing a Japanese data item (`PICTURE N`) of COBOL is permitted as a data description corresponding to the SQL mixed character string data type (`MCHAR` or `MVARCHAR`) only if the Unicode functionality of COBOL2002 is used. For details, see the notes on `MCHAR[(`*n*`)]` and `MVARCHAR(`*n*`)` in *Table F-8 SQL data types and COBOL data descriptions*.

#5

If you specify `utf-8` as the character code type in the `pdntenv` command (in UNIX edition, `pdsetup` command), the HiRDB server's default character set is set to UTF-8.

G. Data Dictionary Table Retrieval

This section discusses the retrieval and referencing of data dictionary tables.

HiRDB data dictionary tables can be referenced in the same way as an ordinary HiRDB database by using operation SQL statements. The authorization identifier of a dictionary table is MASTER.

This appendix provides examples of SQL descriptions for dictionary table retrievals and explains the definition information required for referencing.

The following table lists the data dictionary tables that can be referenced.

Table G-1: Data dictionaries

Number	Table name	Description	Row contents
1	`SQL_PHYSICAL_FILES`	HiRDB file information (correspondences between HiRDB file system names and RDAREA names)	One HiRDB file
2	`SQL_RDAREAS`	Information such as the RDAREA names, their definition information, the RDAREA types, the number of stored tables, and number of indexes	One RDAREA
3	`SQL_TABLES`	Owner name and table name of each table (including dictionary tables) in the database	One table
4	`SQL_COLUMNS`	Column definition information, such as the column names and their data types	One column
5	`SQL_INDEXES`	Owner name and index name of each index (including dictionary tables) in the database	One index
6	`SQL_USERS`	Execution privileges and authorization identifiers of users authorized to access the database	One user
7	`SQL_RDAREA_PRIVILEGES`	Grants of RDAREA usage privileges	Use of one RDAREA for one authorization identifier
8	`SQL_TABLE_PRIVILEGES`	Grants of table access privileges	Access to one table for one authorization identifier

Number	Table name	Description	Row contents
9	`SQL_VIEW_TABLE_USAGE`	Names of base tables used for view tables	One view table
10	`SQL_VIEWS`	View definition information	One view table
11	`SQL_DIV_TABLE`	Table partitioning information (partitioning conditions specified in `CREATE TABLE` and names of RDAREAs that store partitioned tables)	One table (described by *n* rows)
12	`SQL_INDEX_COLINF`	Names of columns to which indexes are assigned	One index (described by *n* rows)
13	`SQL_DIV_INDEX`	Index partitioning information (storage RDAREA names)	One index (described by *n* rows)
14	`SQL_DIV_COLUMN`	`BLOB`-type column partitioning information (storage RDAREA names specified when `CREATE TABLE` was executed)	One column (described by *n* rows)
15	`SQL_ROUTINES`	Routine definition information	One routine (described by one row)
16	`SQL_ROUTINE_RESOURCES`	Information about resources used in a routine	One routine (described by *n* rows)
17	`SQL_ROUTINE_PARAMS`	Information about parameter definitions in a routine	One routine (described by *n* rows)
18	`SQL_TABLE_STATISTICS`	Table statistical information	One table
19	`SQL_COLUMN_STATISTICS`	Column statistical information	One column
20	`SQL_INDEX_STATISTICS`	Index statistical information	One index
21	`SQL_DATATYPES`	Information about user-defined types	One user-defined type
22	`SQL_DATATYPE_DESCRIPTORS`	Information about user-defined type configuration attributes	One attribute
23	`SQL_TABLE_RESOURCES`	Information about resources used in a table	One resource
24	`SQL_PLUGINS`	Plug-in information	One plug-in
25	`SQL_PLUGIN_ROUTINES`	Information about routines in a plug-in	One plug-in routine

Number	Table name	Description	Row contents
26	SQL_PLUGIN_ROUTINE_PARAMS	Information about parameters in a plug-in routine	One set of parameter information
27	SQL_INDEX_TYPES	Information about index types	One index type
28	SQL_INDEX_RESOURCES	Information about resources used in an index	One set of resource information
29	SQL_INDEX_DATATYPE	Information about target items in an index	One set of target item information (for one level)
30	SQL_INDEX_FUNCTION	Information about abstract data type functions used in an index	One set of abstract data function information
31	SQL_TYPE_RESOURCES	Information about resources used in a user-defined type	One set of resource information
32	SQL_INDEX_TYPE_FUNCTION	Information about abstract data type functions used in an index that defines index types	One index type (described by *n* rows)
33	SQL_EXCEPT	Information about exclusion key values in an index	Exclusion key groups in one index (described by *n* rows)
34	SQL_IOS_GENERATIONS	For UNIX: Generation information in the HiRDB file system areas when the inner replica facility is used For Windows: Used by the system (table is empty)	For UNIX: One row per HiRDB file system area For Windows: None
35	SQL_TRIGGERS	Information on the trigger that is inside the schema	One trigger in one row
36	SQL_TRIGGER_COLUMNS	UPDATE trigger event column list information	One piece of event column information in one row
37	SQL_TRIGGER_DEF_SOURCE	Trigger definition source information	One piece of trigger definition source information in *n* rows
38	SQL_TRIGGER_USAGE	Resource information referenced inside a trigger action condition	One resource name being referenced inside the trigger action condition in one row

Number	Table name	Description	Row contents
39	`SQL_PARTKEY`	Partitioning key information of a matrix-partitioned table	One piece of partitioning key information in one row
40	`SQL_PARTKEY_DIVISION`	Partitioning condition value information of a matrix-partitioned table	One piece of partitioning condition value information in one row
41	`SQL_AUDITS`	Information on the monitoring target	One object or information on one event for one user in one row
42	`SQL_REFERENTIAL_CONSTRAINTS`	Referential constraint conditions	Information on one constraint in one row
43	`SQL_KEYCOLUMN_USAGE`	Information on the columns that make up the external keys	Information on one column in one row
44	`SQL_TABLE_CONSTRAINTS`	Information on the integrity constraints in a schema	Information on one integrity constraint in one row
45	`SQL_CHECKS`	Check constraint information	Information on one check constraint in one row
46	`SQL_CHECK_COLUMNS`	Information on columns used by a check constraint	Information on one column using one check constraint in one row
47	`SQL_DIV_TYPE`	Partitioning key information for matrix partitioning tables that combine key range partitioning and hash partitioning	Information on one partitioning key in one row
48	`SQL_SYSPARAMS`	Restriction information on the number of consecutive certification failures and the password character string	Information on one setting item in one row, and restriction information on one number of consecutive certification failures or one password character string in *n* rows

Number	Table name	Description	Row contents
49	SQL_INDEX_XMLINF	Information on the component substructure paths of a substructure index	Information on one index in one row
50	SQL_SEQUENCES	Information on the sequence generator	Information on one sequence generator in one row

G.1 Examples of SQL statements for retrieval

Examples of SQL statements that retrieve data dictionary tables are shown as follows: For details about the SQL statements, see the *HiRDB Version 9 SQL Reference* manual.

The types of information that a particular user can retrieve depend on the setting of the data dictionary referencing authorization. For details about how to set data dictionary referencing authorizations, see the *HiRDB Version 9 System Operation Guide*.

After a dictionary table is retrieved, immediately issue a `COMMIT` statement or specify `WITHOUT LOCK NOWAIT` as shown in the retrieval example.

Example 1

Retrieve the server names for the RDAREAs that exist in the HiRDB system, the HiRDB filenames, and the names of the RDAREAs to which the HiRDB files belong:

```
SELECT X_SERVER_NAME, PHYSICAL_FILE_NAME, X.RDAREA_NAME
  FROM MASTER.SQL_PHYSICAL_FILES X, MASTER.SQL_RDAREAS Y
  WHERE X.RDAREA_NAME=Y.RDAREA_NAME
  ORDER BY SERVER_NAME
  WITHOUT LOCK NOWAIT
```

Example 2

From the column definition information for tables owned by a user, retrieve the names of the tables that contain the columns, the column names, the data types, and the column data lengths:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH
  FROM MASTER.SQL_COLUMNS
  WHERE TABLE_SCHEMA=USER*
  ORDER BY TABLE_NAME
  WITHOUT LOCK NOWAIT
```

Example 3

From the index definition information for tables owned by a user, retrieve the names of the tables that contain the index, the index names, and the percentages of unused space per page:

```
SELECT TABLE_NAME, INDEX_NAME, FREE_AREA
```

```
  FROM MASTER.SQL_INDEXES
  WHERE TABLE_SCHEMA=USER*
  ORDER BY TABLE_NAME
  WITHOUT LOCK NOWAIT
```

Example 4

Retrieve the tables that a user can access and the types of access privileges to those tables (SELECT, INSERT, DELETE, and UPDATE privileges):

```
SELECT TABLE_NAME, SELECT_PRIVILEGE, INSERT_PRIVILEGE,
  DELETE_PRIVILEGE, UPDATE_PRIVILEGE
  FROM MASTER.SQL_TABLE_PRIVILEGES
  WHERE GRANTEE=USER* OR GRANTEE='PUBLIC'
  WITHOUT LOCK NOWAIT
```

Example 5

Retrieve the number of RDAREAs that become targets for group specification by a command (RDAREAs beginning with RD1):

```
SELECT COUNT(*) FROM MASTER.SQL_RDAREAS
  WHERE RDAREA_TYPE='U' AND
  RDAREA_NAME LIKE 'RD1%'
  WITHOUT LOCK NOWAIT
```

Example 6

Retrieve the names of RDAREAs that become targets for group specification by a command (RDAREAs beginning with RD1):

```
SELECT RDAREA_NAME FROM MASTER.SQL_RDAREAS
  WHERE RDAREA_TYPE='U' AND
  RDAREA_NAME LIKE 'RD1%' ORDER BY RDAREA_NAME
  WITHOUT LOCK NOWAIT
```

Example 7

Retrieve the name of the RDAREA that stores a non-partitioning table owned by a user (table named T1):

```
SELECT X.RDAREA_NAME
  FROM MASTER.SQL_RDAREAS X, MASTER.SQL_TABLES Y
  WHERE Y.TABLE_SCHEMA=USER*
    AND Y.TABLE_NAME='T1'
    AND X.RDAREA_NAME=Y.RDAREA_NAME
```

* USER refers to a variable that stores a value indicating the executing user's authorization identifier. For details about authorization identifiers, see the *HiRDB Version 9 SQL Reference* manual.

Example 8

Retrieve the name of the RDAREA that stores objects for a stored procedure or stored function, to be used during execution to re-initialize a data dictionary LOB RDAREA.

```
SELECT RDAREA_NAME FROM MASTER.SQL_DIV_COLUMN
  WHERE TABLE_SCHEMA='HiRDB'
    AND TABLE_NAME='SQL_ROUTINES'
    AND COLUMN_NAME='ROUTINE_OBJECT'
  WITHOUT LOCK NOWAIT
```

Note

When a data dictionary LOB RDAREA is reinitialized, all its stored SQL objects must be re-created.

Example 9

Retrieve the name of the stored procedure or stored function that has an invalid SQL object or an invalid index:

```
SELECT ROUTINE_SCHEMA,ROUTINE_NAME
    FROM MASTER.SQL_ROUTINES
    WHERE ROUTINE_VALID='N'
      OR INDEX_VALID='N'
    WITHOUT LOCK NOWAIT
```

Example 10

Retrieve the data types of the arguments that are actually used when embedded variables are used in arguments of the user-defined function `FUNC1`:

```
SELECT PARAMETER_NAME,DATA_TYPE,UDT_OWNER,UDT_NAME,
PARAMETER_NO
      FROM MASTER.SQL_ROUTINE_PARAMS
      WHERE ROUTINE_SCHEMA=USER AND ROUTINE_NAME='FUNC1'
        ORDER BY PARAMETER_NO
      WITHOUT LOCK NOWAIT
```

Example 11

To reorganize all the tables owned by user `USERA`, retrieve the RDAREAs containing any of those tables (the RDAREAs that need to be placed in shutdown status):

Non-partitioned table:

```
SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_TABLES
    WHERE TABLE_SCHEMA=USERA AND RDAREA_NAME IS NOT NULL
     WITHOUT LOCK NOWAIT
```

Partitioned table:

```
SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_DIV_TABLE
     WHERE TABLE_SCHEMA=USERA
     WITHOUT LOCK NOWAIT
```

Eliminate any duplicated RDAREA names from the result, then place all the resulting RDAREAs in shutdown status.

G.2 Data dictionary table details

The definition information required for referencing of each data dictionary table is shown as follows:

Each dictionary table has a column with the VARCHAR or MVARCHAR data type. This is the dictionary datatype operand for the database initialization utility or database structure modification utility, and must be set to either VARCHAR or MVARCHAR.

(1) SQL_PHYSICAL_FILES table

This table manages HiRDB file information (relationships between HiRDB files and RDAREAs). (Each row describes information on one HiRDB file.)

The following table shows the contents of the SQL_PHYSICAL_FILES table.

Table G-2: SQL_PHYSICAL_FILES table contents

Number	Column name	Data type	Contents
1	SERVER_NAME	CHAR(8)	Server name (back-end server name or dictionary server name)
2	PHYSICAL_FILE_NAME	VARCHAR(167)	HiRDB filename
3	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the RDAREA to which HiRDB files are allocated
4	INITIAL_SIZE	INTEGER	Number of HiRDB file segments
5	PHYSICAL_FILE_ID	INTEGER	Physical file ID

(2) SQL_RDAREAS table

This table manages RDAREA definition information. (Each row describes information on one RDAREA.)

The following table shows the contents of the SQL_RDAREAS table.

Table G-3: SQL_RDAREAS table contents

Number	Column name	Data type	Contents
1	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	RDAREA name
2	SERVER_NAME	CHAR(8)	Server name (back-end server name or dictionary server name)

Number	Column name	Data type	Contents
3	RDAREA_TYPE	CHAR(1)	RDAREA type: M: Master directory RDAREA D: Data directory RDAREA S: Data dictionary RDAREA W: Work RDAREA U: User RDAREA P: Data dictionary LOB RDAREA L: User LOB RDAREA R: Registry RDAREA K: Registry LOB RDAREA A: list RDAREA
4	PAGE_SIZE	INTEGER	Page length (in bytes)
5	SEGMENT_SIZE	INTEGER	Segment size (in pages)
6	FILE_COUNT	INTEGER	Number of HiRDB files
7	N_TABLE	INTEGER	Number of tables stored (defined number) (initial value is 0). If tables and sequence generators are both defined, the maximum number of tables and sequence generators combined is 500.
8	N_INDEX	INTEGER	Number of indexes stored (defined number) (initial value is 0)
9	RDAREA_ID	INTEGER	RDAREA ID
10	REBALANCE_TABLE	CHAR(1)	Rebalance table status: Y: A rebalance table is used. Null value: No rebalance table is used.
11	MAX_ENTRIES	INTEGER	Maximum number of entries in the list NULL for any RDAREA other than the list RDAREA or if max entries is not specified
12	EXTENSION	CHAR(1)	Specification of RDAREA expansion: U: Specified. N: Not specified.
13	EXTENSION_SEGMENT_SIZE	INTEGER	Number of extension segments NULL if RDAREA expansion is not specified
14	ORIGINAL_RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	For UNIX: Name of the original RDAREA Null value if the RDAREA is not a replica RDAREA. For Windows: Used by the system (no contents)

Number	Column name	Data type	Contents
15	`ORIGINAL_RDAREA_ID`	`INTEGER`	For UNIX: ID of the original RDAREA Null value if the RDAREA is not a replica RDAREA. For Windows: Used by the system (no contents)
16	`GENERATION_NUMBER`	`SMALLINT`	For UNIX: Generation number Null value if the RDAREA is not an original RDAREA or replica RDAREA. For Windows: Used by the system (no contents)
17	`REPLICA_COUNT`	`SMALLINT`	For UNIX: Replica counter Null value if the RDAREA is not an original RDAREA or if the RDAREA has lost its replica RDAREA. For Windows: Used by the system (no contents)
18	`REPLICA_STATUS`	`CHAR(1)`	For UNIX: Replica status `C`: Current RDAREA `S`: Sub-RDAREA Null value if the RDAREA is not an original RDAREA or replica RDAREA. For Windows: Used by the system (no contents)
19	`SHARED`	`CHAR(1)`	Shared RDAREA `S`: Shared RDAREA Null value: Unshared RDAREA
20	`N_SEQUENCE`	`INT`	Number of stored sequence generators. If the number of sequence generators is 0, the null value is set. If tables and sequence generators are both defined, the maximum number of tables and sequence generators combined is 500.

(3) SQL_TABLES table

This table manages information of the tables found in schemas. (Each row describes information on one table.)

The rows of the `SQL_TABLES` table are created during table definition, and row deletion is performed during table deletion.

The following table shows the contents of the SQL_TABLES table.

Table G-4: SQL_TABLES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner or PUBLIC for a public view table
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	TABLE_TYPE	CHAR(16)	Table type BASE TABLE: Base table VIEW: View table READ ONLY VIEW: Read-only view table
4	TABLE_ID	INTEGER	Table ID Indicates an internal ID that is unique within the system.
5	N_COLS	SMALLINT	Number of structure columns
6	N_INDEX	SMALLINT	Number of defined indexes Total number of the following defined indexes: • B-tree indexes (including primary keys, cluster keys, and substructure indexes) • Plug-in indexes (initial value is 0)
7	DCOLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Partitioned column name (column name of the first partitioning key for multiple column partitioning or matrix partitioning) Null value for a non-partitioned table and for a view table.
8	VDEFLEN	INTEGER	Length of view analysis information Null value for a base table.
9	FREE_AREA	SMALLINT	Percentage of unused space in each page 0 for a view table.
10	FREE_PAGE	SMALLINT	Rate (%) of free pages (unused pages) inside a segment 0 for a view table.
11	TABLE_COMMENT	VARCHAR(255) or MVARCHAR(255)	Comment (initial value is NULL)
12	CREATE_TIME	CHAR(14)	Table creation date and time (*YYYYMMDDHHMMSS*)

Number	Column name	Data type	Contents
13	ENQ_RESOURCE_ SIZE	CHAR(1)	Locked resource unit P: In page units Null value for locking in units of rows and for a view table.
14	DEFAULT_COLUMN	SMALLINT	Number of specified columns with the default value (DEFAULT clause or WITH DEFAULT). Null value for a view table and for a dictionary table.
15	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of storage RDAREA for non-partitioned table Null value for a partitioned table and for a view table.
16	DEFINITION_CACHE_ SIZE	INTEGER	Table definition cache size (in bytes) (Null value for dictionary tables)
17	STATISTICS_CACHE_ SIZE	INTEGER	Statistical information cache size (in bytes) (The initial value is a null value.)
18	N_RDAREA	INTEGER	Number of RDAREAs for storage of table (1 to 1,024) 0 for a view table.
19	FIX_TABLE	CHAR(1)	FIX specification F: Specified N: Not specified
20	VIEW_LEVEL	INTEGER	Number of nesting levels in view definition Null value for a base table.
21	N_BASETABLE	INTEGER	Number of base tables used for a view table Null value for a base table.
22	ROW_LENGTH	INTEGER	Row length of a FIX table Null value for a non-FIX table and for a view table.
23	N_NOTNULL	INTEGER	Number of NOT NULL values Null value for a view table and for a dictionary table.

Number	Column name	Data type	Contents
24	COMPRESS_TYPE	VARCHAR(8)	Data compression information: - Compression type (first byte) S: Data compression (SUPPRESS) - Suppressed data type (byte 2 and beyond): D: DECIMAL Null value for a table without SUPPRESS specification, for a view table, and for a dictionary table.
25	DIV_TYPE	CHAR(1)	Partitioning type P: Boundary value partitioning and matrix partitioning H: Flexible hash partitioning F: FIX hash partitioning M: Hash mixed matrix partitioning Null value for a non-partitioned table, for a key range partitioning table, and for a view table.
26	HASH_NAME	VARCHAR(8) or MVARCHAR(8)	Hash function name "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" "HASHA" "HASHB" "HASHC" "HASHD" "HASHE" "HASHF" Null value for a table without a HASH specification, for a matrix partitioning table, for a view table, and for a data dictionary table.
27	N_LOB_COLUMN	SMALLINT	Number of columns with BLOB-data type Null value for a view table and for a table without BLOB columns.
28	N_LOB_RDAREA	INTEGER	Number of user LOB RDAREAs for a table Null value for a view table, for a table without BLOB columns, and for a table without an abstract data type containing BLOB attributes.
29	CHANGE_TIME	CHAR(14)	Time table definition was changed (*YYYYMMDDHHMMSS*) Null value when a table is created initially.

Number	Column name	Data type	Contents
30	N_DIV_COLUMN	SMALLINT	Number of partitioning key columns (216) Null value for a non-partitioned table, for a table with single column partitioning keys specified, and for a view table.
31	COLUMN_SUP_INF	CHAR(1)	Whether or not data suppression is specified for each column: Y: Specified Null value: No specification Null value for a table for which column-by-column data suppression is not specified and for a view table.
32	N_ADT_COLUMN	SMALLINT	Number of columns with an abstract data type Null value for a table in which an abstract data type is not defined.
33	WITHOUT_ ROLLBACK	CHAR(1)	Whether or not a WITHOUT ROLLBACK is specified 'Y': Specified Null value: No specification Null value for a table for which WITHOUT ROLLBACK is not defined and for a view table.
34	N_EXCEPT_VALUES	INTEGER	Number of exclusion key values in an index Null value for an index without exceptional value specifications and for a view table.
35	EXCEPT_VALUES_LEN	INTEGER	Total length of exclusion key values in an index Null value for an index without exceptional value specifications and for a view table.
36	REBALANCE	CHAR(1)	Whether or not the rebalancing facility is used: Y: Used. Null value for a table that does not use the rebalancing facility and for a view table.
37	INDEXLOCK_OPT	CHAR(1)	Information used by the system
38	N_PK_COLUMNS	SMALLINT	Number of columns for the primary key Null value if no primary key is defined.
39	FOREIGN_SERVER_ NAME	VARCHAR(30) or MVARCHAR(30)	Information used by the system
40	FOREIGN_SERVER_ ID	INTEGER	Information used by the system
41	BASE_FOREIGN_ TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Information used by the system

Number	Column name	Data type	Contents
42	BASE_FOREIGN_ TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Information used by the system
43	N_RDAREA_BEFORE_ REBALANCE	INTEGER	Number of partitioning information items before execution of ALTER TABLE ADD RDAREA (number of rows in SQL_DIV_TABLE table)#1 Null value if rebalancing is started, for a table that is not a rebalancing table, and for a view table.
44	ON_REBALANCE	CHAR(1)	Rebalancing status: Y: Under execution Null value: Execution not ongoing Becomes Y after rebalancing has started, and becomes a null value when rebalancing is normally terminated.
45	SEGMENT_REUSE	CHAR(1)	Whether or not SEGMENT REUSE is specified Y: Specified Null value: Not specified Null value if NO is specified for SEGMENT REUSE (including when its specification is omitted) and for a view table.
46	N_REUSE_SEGMENT	INTEGER	Number of segments that start reusing free areas.#2 Null value if NO is specified for SEGMENT REUSE (including when its specification is omitted) and for a view table.
47	REUSE_SEGMENT_SIZE	CHAR(10)	Specified number of segments that start reusing free areas.#3 Null value if a value other than a segment count is specified for SEGMENT REUSE and for a view table.
48	REUSE_SEGMENT_SIZE_ TYPE	CHAR(1)	Unit for the number of segments that start reusing free areas. K: Specifies K. M: Specifies M. Null value: Not specified. Null value if a value other than a segment count is specified for SEGMENT REUSE and for a view table.

Number	Column name	Data type	Contents
49	INSERT_ONLY	CHAR(1)	Whether or not the falsification prevention facility is specified Y: Specified Null value: Not specified Null value if the falsification prevention facility is not used and for a view table.
50	DELETE_PROHIBIT_TERM_TYPE	CHAR(1)	Type of deletion prevented duration I: Date interval data Y: Labeled duration (YEAR) M: Labeled duration (MONTH) D: Labeled duration (DAY) Null value: Not specified Null value if the falsification prevention facility is not used and for a view table.
51	DELETE_PROHIBIT_TERM	CHAR(10)	Specification value for the deletion prevented duration[#4] Null value if the falsification prevention facility is not used, if no deletion prevented duration is specified, and for a view table.
52	SYSGEN_COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the insert history maintenance column Null value if the falsification prevention facility is not used, if no deletion prevented duration is specified, and for a view table.
53	N_TRIGGER	INTEGER	Number of defined triggers Null value if no trigger is defined, and for a view table or a data dictionary table.
54	N_DIV_DIMENSION	SMALLINT	Number of division dimensions Null value for a table that is not a matrix-partitioned table.
55	AUDIT_TABLE_OPTION	CHAR(1)	Value that specifies whether this table is an audit trail table. Y: Audit trail table V: View table based on an audit trail table Null value for a table that is not an audit trail table or a view table based on an audit trail table.
56	N_PARENTS	SMALLINT	Number of foreign keys Null value for a table without a defined referential constraint and for a view table.

Number	Column name	Data type	Contents
57	N_CHILDREN	SMALLINT	Number of foreign keys that reference the main keys of this table Null value for an unreferenced table that is not a referenced table and for a view table.
58	N_FK_COLUMNS	SMALLINT	Total number of foreign key columns Null value for a table without defined referential constraints and for a view table.
59	CHECK_PEND	CHAR(1)	Type of check pending status for a referential constraint C: Pending status Null value: Non-pending status Null value for a view table.
60	N_CHECK	INTEGER	Number of defined check constraints Null value for a table without defined referential constraints and for a view table.
61	N_CHECK_LIMIT	INTEGER	Check constraint limit[#5] Null value for a table without defined referential constraints and for a view table.
62	CHECK_PEND2	CHAR(1)	Type of check pending status for a check constraint C: Pending status Null value: Non-pending status Null value for a view table.
63	CHK_SOURCE_LEN	INTEGER	Total length of search conditions of a check constraint Null value for a table without defined referential constraints and for a view table.
64	SHARED	CHAR(1)	Shared table specification S: Shared table Null value: Unshared table
65	CHANGE_TIME_INSERT_ONLY	CHAR(14)	Update date and time of a falsification prevention table (*YYYYMMDDHHMMSS*) Null value when a table is defined and for a view table.
66	N_UPDATE_COLUMN	SMALLINT	Number of columns for which an updatable column attribute is specified Null value for a table without a specified updatable column attribute and for a view table.

Number	Column name	Data type	Contents
67	TABLE_CREATOR	VARCHAR(30) or MVARCHAR(30)	Creator of a public view table Null value for a table that is not a public view table.
68	N_CONSTRUCTOR_COLUMN	SMALLINT	Used by the system; always the null value.
69	CONSTRUCTOR_TYPE	CHAR(1)	Used by the system; always the null value.
70	NONE_DFLTCST_CLMCOUNT	SMALLINT	Number of columns for which non-default character sets are specified Null value if no character set is specified.
71	CHARSET_SPECCOUNT	SMALLINT	Number of columns for which character sets are specified Null value if no character set is specified.
72	N_PARTIAL_STRUCTURE_INDEXES	SMALLINT	Number of substructure indexes defined Null value if no substructure index is defined and for a view table.
73	MEMORY_TABLE	CHAR(1)	Information on table expansion to the memory database: A: Waiting to be expanded to the memory database C: Expansion to the memory database has been completed D: Waiting to be released from expansion to the memory database Null value: Not expanded to the memory database Null value for a table that is not expanded to the memory database and for a view table.
74	DBAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of database area for data Null value for a table that is not expanded to the memory database and for a view table.
75	XDS_NAME	CHAR(8)	XDS name of the storage for a table to be expanded to the memory database[#6] Null value for a table that is not expanded to the memory database and for a view table.

#1: If an RDAREA is added to a rebalancing table by using ALTER TABLE ADD RDAREA, the number of partitioning information items (number of rows in the SQL_DIV_TABLE table) existing before the RDAREA was added is stored.

Once the number of partitioning information items has been set, it will not be updated even when an RDAREA is added by using ALTER TABLE ADD RDAREA until the rebalancing operation is completed by the rebalancing utility (pdrbal). When the rebalancing operation is completed, the null value is set.

#2: When a segment count unit is specified, the following values are stored:

When K is specified: Specified value x 1,024

When M is specified: Specified value x $1,024^2$

#3: Values are stored right-justified. Note that the segment count units (K and M) are not included.

#4: The following is stored depending on the type of deletion prevented duration:

When 'I' is specified: +*YYYYMMDD*. character format

When 'Y', 'M', or 'D' is specified: Right-justified character format

#5: The check constraint limit is the sum of the total number of logical operators specified in the search conditions of the check constraints (number of AND and OR specifications, excluding the AND and OR specifications in WHEN search conditions of CASE expressions) and the total number of check constraints.

Example

If a table is defined as follows, the check constraint limit is 4 (the total number of operators (AND and OR) is 2 and the total number of check constraints is 2):

```
CREATE TABLE "STOCK"
  ("GNO" CHAR(5),"GNAME" CHAR(8),"PRICE" INTEGER,
    "QUANTITY" INTEGER,"STOCKING DATE" DATE)
  CHECK("QUANTITY " ≥ 100 AND "QUANTITY" ≤ 1000)
  CONSTRAINT "QUANTITY RULE"
  CHECK("STOCKING DATE"=DATE('1992-08-21')
    OR "STOCKING DATE"=DATE('1992-09-21'))
  CONSTRAINT "STOCKING DATE RULE"
```

#6: If the left-justified length is less than 8 characters, the column is padded with spaces.

(4) SQL_COLUMNS table

This table manages column definition information. (Each row describes information on one column.)

Rows of the SQL_COLUMNS table are created during table definition, and row deletion (including schema deletion) is performed during table deletion.

The following table shows the contents of the SQL_COLUMNS table.

Table G-5: SQL_COLUMNS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner or PUBLIC for a public view table
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the column
3	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name
4	TABLE_ID	INTEGER	Table ID
5	COLUMN_ID	SMALLINT	Column ID (integer beginning with 1; values less than 1 are not allowed)
6	DATA_TYPE	CHAR(24)	Data type[#1]
7	DATA_LENGTH	CHAR(7)	Column data length is stored right justified in character format (spaces are used for leading zeros)
8	IS_NULLABLE	CHAR(3)	Column null information: YES: Null value allowed NO: Null values not allowed
9	DIVIDED_KEY	CHAR(1)	Partitioning key: Y: Partitioning key Empty: Not a partitioning key
10	CLUSTER_KEY	CHAR(1)	Cluster key: Y: Column used for cluster key Empty: Not a column used for cluster key
11	COLUMN_COMMENT	VARCHAR(255) or MVARCHAR(255)	Comment (The initial value is a null value.)
12	BASE_TYPE	CHAR(1)	Base column type[#7]: C: Column F: Function, operation E: Other Null value for a base table.
13	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of base table that contains base column Null value for a base table.
14	BASE_TABLE	VARCHAR(30) or MVARCHAR(30)	Name of base table that contains base column Null value for a base table.

Number	Column name	Data type	Contents
15	BASE_COLUMN	VARCHAR(30) or MVARCHAR(30)	Base column name Null value for a base table.
16	DEFAULT_COLUMN	CHAR(1)	WITH DEFAULT specification Y: Specified N: Not specified Null value for view tables
17	COLUMN_OFFSET	SMALLINT	Column offset Null value for a non-FIX table and for a view table.
18	HASH_KEY	CHAR(1)	Hash key: Y: Hash key Empty: Other than hash key
19	RECOVERY_TYPE	CHAR(1)	RECOVERY specification: A: ALL P: PARTIAL N: NO Null value if the data type is not BLOB.#8
20	LOB_LENGTH	CHAR(20)	Column length specification stored right-justified in character format (spaces are used for leading zeros) Null value if the length is not for BLOB or BINARY.#8
21	LOB_LENGTH_TYPE	CHAR(1)	Column length type (in column lengths): K: K specified M: M specified G: G specified Empty: Default Null value if the data type is not BLOB.#8
22	DATA_TYPE_CODE	SMALLINT	Data type code#2
23	DATA_LENGTH_CODE	SMALLINT	Column data length code#3
24	LOB_LENGTH_CODE	CHAR(8)	BLOB column data length code#4, #5 Null value if the data type is not BLOB or BINARY.

Number	Column name	Data type	Contents
25	DIVCOL_ORDER	SMALLINT	Partitioning key specification order (0-16) Unique values within the applicable table, beginning with 1. Partitioning key specification order +1. 0 is specified for a column that is not a partitioning key. Null value for a non-partitioned table, for a table with single column partitioning keys specified, and for a view table.
26	SUPPRESS_INF	CHAR(1)	Whether or not data suppression is specified: Y: Specified Null value: No specification Null value for a table without the data suppression specification and for a view table.
27	PLUGIN_ DESCRIPTION	VARCHAR(255)	Plug-in option contents Null value if no PLUGIN clause is specified.
28	UDT_OWNER	VARCHAR(30)	Owner of a user-defined type Null value if the type is not user-defined.
29	UDT_NAME	VARCHAR(30)	Name of the user-defined type Null value if the type is not user-defined.
30	UDT_TYPE_ID	INTEGER	User-defined type ID Null value if the type is not user-defined.
31	MAX_ELM	SMALLINT	Maximum number of repetition column elements Null value if the column is not a repetition column.
32	NO_SPLIT	CHAR(1)	Whether or not NO SPLIT is specified: Y: Specified Null value: No specification Null value for a view table and for when ALTER TABLE CHANGE SPLIT is executed.
33	PRIMARY_KEY	CHAR(1)	Primary key type Y: Primary key Empty: Other than the primary key
34	COLLATING_SEQUENCE	CHAR(1)	Information used by the system
35	TRAILING_SPACE	CHAR(1)	Information used by the system

Number	Column name	Data type	Contents
36	SYSTEM_GENERATED	CHAR(1)	Whether or not SYSTEM GENERATED is specified Y: Specified Null value: No specification Null value if SYSTEM GENERATED is not specified and for a view table.
37	DEFAULT_CLAUSE	CHAR(1)	Whether or not the DEFAULT clause is specified Y: Specified Null value: No specification Null value if the DEFAULT clause is not specified and for a view table.
38	DEFAULT_VALUE	VARCHAR(32000) or MVARCHAR(32000)#6	Default value (character format) specified for the DEFAULT clause.#9 Null value if the DEFAULT clause is not specified and for a view table.
39	DEFAULT_VALUE2	VARCHAR(32000) or MVARCHAR(32000)#6	Default value specified for the DEFAULT clause (stores the 32,001st - 64,000th byte values in the character format when a literal is specified).#9 Null value if a literal is not specified, if the DEFAULT clause is not specified, and for a view table.
40	DEFAULT_VALUE3	VARCHAR(3) or MVARCHAR(3)	Default value specified for the DEFAULT clause (stores the 64,000th byte value and beyond in the character format when a literal is specified).#9 Null value if a literal is not specified, if the DEFAULT clause is not specified, and for a view table.
41	CHECK_COLUMN	CHAR(1)	Check constraint specification Y: Specified Null value for a table in which a check constraint is not defined and for a view table.
42	FOREIGN_KEY	CHAR(1)	Foreign key type Y: Foreign key configuration table Null value: Non-foreign key configuration table
43	UPDATABLE	CHAR(1)	Updatable column attribute U: Can be updated (UPDATE) N: Can be updated only once from a null value to a non-null value (UPDATE ONLY FROM NULL) Null value for a table without a specified updatable column attribute and for a view table.
44	CONSTRUCTOR_TYPE	CHAR(1)	Used by the system; always the null value.

Number	Column name	Data type	Contents
45	CHARSET_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Character set owner (always MASTER). Null value if no character set is specified.
46	CHARSET_NAME	VARCHAR(30) or MVARCHAR(30)	Character set name: EBCDIK: EBCDIK is specified for the character set. UTF16: UTF16 is specified for the character set. Null value if no character set is specified.
47	CHARSET_ID	INTEGER	Character set ID. Null value if no character set is specified. For details about the character set IDs, see the *HiRDB Version 9 Installation and Design Guide*.

#1: The stored value depends on the data type, as follows:

Data type	Value to be stored
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT	FLOAT
DOUBLE PRECISION	
SMALLFLT	SMALLFLT
REAL	
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	NCHAR
NVARCHAR	NVARCHAR
MCHAR	MCHAR
MVARCHAR	MVARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
INTERVAL YEAR TO DAY	INTERVAL YEAR TO DAY

Data type	Value to be stored
`INTERVAL HOUR TO SECOND`	`INTERVAL HOUR TO SECOND`
`BINARY`	`BINARY`
`BLOB`	`BLOB`
Abstract data type	`ADT`
`BOOLEAN`	`BOOLEAN`

#2: The stored value depends on the data type, as follows:

Data type	Value to be stored	
	When the null value can be specified	**When the null value cannot be specified**
INTEGER	F1	F0
SMALLINT	F5	F4
DECIMAL	E5	E4
FLOAT	E1	E0
DOUBLE PRECISION		
SMALLFLT	E3	E2
REAL		
CHAR	C5	C4
VARCHAR	C1	C0
NCHAR	B5	B4
NVARCHAR	B1	B0
MCHAR	A5	A4
MVARCHAR	A1	A0
DATE	71	70
TIME	79	78
TIMESTAMP	7D	7C
INTERVAL YEAR TO DAY	65	64

Data type	Value to be stored	
	When the null value can be specified	When the null value cannot be specified
INTERVAL HOUR TO SECOND	6F	6E
BINARY	91	90
BLOB	93	92
Abstract data type	83	82
BOOLEAN	21	20

#3: For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB, BINARY, and abstract data types.

#4: The specified column length is stored in binary format in 8 bytes divided into 4-byte segments.

#5: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

#6: Specifies NO SPLIT.

#7: E (Other) is set when the selection formula is one of the following:

- Scalar operations (four arithmetic operations, data operation, time operation, CASE expression, and scalar functions)
- Literal
- CAST specification
- Function invocation (excluding plug-in functions)
- USER
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP

#8: The value listed below is set for columns of a view table when a function call is specified and its resulting data type is BLOB. For this reason, the resulting format might differ from the format specified when the function was defined.

- NULL is set in a RECOVERY_TYPE column.

- An exact multiple of the largest unit (K, M, or G) is set in a LOB_LENGTH column.
- The largest divisible units are set in a LOB_LENGTH_TYPE column.

#9: The following table shows the values that are stored when the DEFAULT clause is specified.

Table G-6: Values that are stored when the DEFAULT clause is specified

Default value	**Data type#1**	**Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column#2**	
		Data size (in char format)	**Default value (character format)**
Omitted	All	Null value	Null value
NULL	All	4	'NULL'
USER	CHAR and MCHAR	4	'USER'
	VARCHAR and MVARCHAR		
CURRENT DATE	DATE, or CHAR(10)	12	'CURRENTΔDATE'#3
CURRENT_DATE		12	'CURRENT_DATE'
CURRENT TIME	TIME or CHAR(8)	12	'CURRENTΔTIME'#3
CURRENT_TIME		12	'CURRENT_TIME'
CURRENT TIMESTAMP(*p*) (*p*: decimal seconds precision)	TIMESTAMP, CHAR(19), CHAR(22), CHAR(24), or CHAR(26)	20	'CURRENTΔTIMESTAMP(*p*)'#3, #7
CURRENT_TIMESTAMP(*p*) (*p*: decimal seconds precision)		20	'CURRENT_TIMESTAMP(*p*)'#7

Default value			Data type#1	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column#2	
				Data size (in char format)	**Default value (character format)**
Lit	Char string lit	Character string literal Example 1: `'HiRDB'` Example 2: Δ`'2002-10-24 10:50:23.1234'`	`CHAR` or `MCHAR`	*def-val-size* + 2#4	*specified-default-value-size*#4 Example: `''HiRDB''`
			`VARCHAR` or `MVARCHAR`		
			`DATE`, `TIME`, or `TIMESTAMP`	*def-val-size* + 2#4	*specified-default-value-size*#4 Example: `''2002-10-24`Δ`10:50:23.1234''`
		Mixed character string literal Example: `M'100 years'`	`CHAR` or `MCHAR`	*def-val-size* + 3#4	*specified-default-value-size*#4 Example: `'M'100 years''`
			`VARCHAR` or `MVARCHAR`		
		National character string literal Example: `N'software'`	`NCHAR` or `NVARCHAR`	*def-val-size* + 3#4	*specified-default-value-size*#4 Example: `'N'software''`

		Hexadecimal character string literal
		Example 1:
		`X'48692D43'`
		Example 2:
		`X'2002102410502312'`
			`CHAR`, `VARCHAR`, `MCHAR`, `MVARCHAR`, or `BINARY`	*def-val-size* + 3#4	Example: `'X'48692D43''`#4, #6

			`DATE`, `TIME`, or `TIMESTAMP(`*p*`)`		Example: `'X'2002102410502312''`#4, #6

Default value			Data type#1	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column#2	
				Data size (in char format)	Default value (character format)
	Num lit	Integer literal Example: `10`	`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, or `SMALLFLT`	*def-val-size*#5	*specified-default-value*#5 Example: `'10'`
		Floating-point literal Example: `15e + 3`	`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, or `SMALLFLT`	22 or 23	*specified-default-value*#5 Example: `'+1.5000000000000000E+04'` (From the left, 1 byte for a sign, 17 bytes for the virtual number portion (decimal literal), 1 byte for 'E', 1 byte for a sign, 2-3 bytes for the exponential part (power of 10))

		Decimal literal
		Example 1:
		`15.5`
		Example 2:
		`-010101.`
		Example 3:
		`00011399.`
			`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, `SMALLINT`, `INTERVAL YEAR TO DAY`, or `INTERVAL HOUR TO SECOND`	*def-val-size*#5	*specified-default-value*#5
					Example 1: `' 15.5'`
					Example 2: `'-010101.'`
					Example 3:
					`'+00020199.'` for `INTERVAL YEAR TO DAY`
					`' 00011399.'` for `INTEGER`
					(For `INTERVAL YEAR TO DAY` and `INTERVAL HOUR TO SECOND`, the value is corrected and a sign is added to the front (the value is empty in all other cases and for a positive value))

The following abbreviations are used in this table:

Num: Numeric

Lit: Literal

Char: Character

def: default

val: value

Δ : 1-byte space

#1: Excludes `BLOB`, the abstract data type, and `BINARY` of 32,001 bytes or greater.

#2: If the data size is smaller than 32,001 bytes, the DEFAULT_VALUE2 column and DEFAULT_VALUE3 column become null values. If the data size is 32,001-64,000 bytes, the DEFAULT_VALUE3 column becomes a null value.

#3: Spaces between CURRENT and DATE, TIME, or TIMESTAMP are edited into a single space.

#4: The specified default value is stored as a literal expression in the character format. The data size and default value include the literal expressions M, N, X, and apostrophe ('). Therefore, the data size range is 2-32,002 bytes including ' ' for a character string literal, 3-32,003 bytes including M' ' and N' ' for a mixed character string literal and a national character string literal, and 3-64,003 bytes including X' ' for a hexadecimal character string literal.

Bytes 1-32,000 of the specified literal are stored in the DEFAULT_VALUE column; bytes 32,001-64,000 are stored in the DEFAULT_VALUE2 column; and bytes 64,000 and beyond are stored in the DEFAULT_VALUE3 column.

Example:

When 32,000 bytes worth of a default value is specified for the hexadecimal character string literal (a total of 64,003 bytes including X and an apostrophe ('))

```
VARCHAR(32000) DEFAULT X'C1C1C1...C1C1C1'
```

The first 32,000 bytes X'C1C1C1... are stored in the DEFAULT_VALUE column.

The next 32,000 bytes C1C1C1... are stored in the DEFAULT_VALUE2 column.

The remaining 3 bytes C1' are stored in the DEFAULT_VALUE3 column.

#5: The specified default value is stored as a literal expression in the character format. Size in the character format expression is stored for the data size.

Example:

When a default value is specified for the numeric literal

```
INTEGER DEFAULT 100
```

The first 3 bytes 100 are stored in the DEFAULT_VALUE column.

Null values are stored in the DEFAULT_VALUE2 and DEFAULT_VALUE3 columns.

#6: The value is all upper-case letters (upper-case letters are stored even when lower-caser letters are specified for the value).

#7: If the decimal precision (*p*) for the CURRENT_TIMESTAMP value to be specified for the default value is omitted, *p* = 0 is assumed.

(5) SQL_INDEXES table

This table manages information about the following indexes (each row describes information for one index):

- B-tree indexes (including primary keys, cluster keys, and substructure indexes)
- Plug-in indexes

The following table shows the contents of the SQL_INDEXES table.

Table G-7: SQL_INDEXES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains an index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	TABLE_ID	INTEGER	Table ID
6	UNIQUE_TYPE	CHAR(1)	Unique type: U: Unique N: Non-unique
7	COLUMN_COUNT	SMALLINT	Number of columns comprising the index
8	CREATE_TIME	CHAR(14)	Index creation date and time (*YYYYMMDDHHMMSS*)
9	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of storage RDAREA for non-partitioned index Null value for a partitioning key index.
10	CLUSTER_KEY	CHAR(1)	Index type: Y: Cluster index N: Non-cluster index
11	DIV_INDEX	CHAR(1)	Type of first column of the columns that make up the index: Y: Partitioning key or plug-in index (The same order from the first key of partitioning keys specified in CREATE TABLE for multiple-partitioning keys) N: Not a partitioning key
12	FREE_AREA	SMALLINT	Percentage of unused space in each page (%)

Number	Column name	Data type	Contents
13	COLUMN_ID_LIST	VARCHAR(64)	List of IDs of columns constituting the index[#] Ascending and descending orders are indicated with + and -. + is set to specify the descending order of single-column indexes (other than cluster key indexes). + is always set for plug-in indexes.
14	SPLIT_OPT	CHAR(1)	Page split option: U: Unbalanced split Null value for an index for which unbalanced split is not specified.
15	ATTR_COUNT	SMALLINT	Number of abstract data type attributes constituting an index Null value for CREATE INDEX (Format 1).
16	INDEX_TYPE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of an index type Null value for CREATE INDEX (Format 1).
17	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of an index type Null value for CREATE INDEX (Format 1).
18	INDEX_TYPE_ID	INTEGER	Index type ID Null value for CREATE INDEX (Format 1).
19	PLUGIN_DESCRIPTION	VARCHAR(255)	Plug-in option contents Null value if PLUGIN is not specified.
20	N_FUNCTION	INTEGER	Number of applied functions Null value for CREATE INDEX (Format 1).
21	EXCEPT_VALUES	CHAR(1)	Whether or not exclusion key values are specified: Y: Specified N: Not specified
22	N_EXCEPT_VALUES	SMALLINT	Number of exclusion key values in an index Null value for indexes without exception value specifications
23	ARRAY_TYPE	CHAR(1)	Type of the columns that make up the index: M: Includes repetition columns Null value: The columns that make up the index do not include repetition columns.
24	LOCK_OPT	CHAR(1)	Information used by the system
25	PRIMARY_KEY	CHAR(1)	Index type Y: Primary key index Null value: Not a primary key index

Number	Column name	Data type	Contents
26	DIV_IN_SRV	CHAR(1)	Whether or not a non-partitioning key index is partitioned within the server: Y: Partitioned within the server Null value: Not partitioned within the server Null value for a partitioning key index.
27	SHARED	CHAR(1)	Shared index specification S: Shared index Null value: Unshared index
28	N_PARTIAL_STRUCTURE_PATHS	SMALLINT	Number of component substructure paths of a substructure index Null value if no substructure index is defined.
29	USING_UNIQUE_TAG	CHAR(1)	Substructure path uniqueness Y: Substructure paths are unique NULL: Other than the above Null value if no substructure index is defined or when USING UNIQUE TAG is not specified.
30	DBAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Database area name for index Null value if the table is not expanded to the memory database.

#: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(6) SQL_USERS table

This table manages information about the execution and DBA (database administration) privileges of users. (Each row describes information on one user.)

This table can be referenced only by owners with the DBA privilege and auditors.

The following table shows the contents of the SQL_USERS table.

Table G-8: SQL_USERS table contents

Number	Column name	Data type	Contents
1	USER_ID	VARCHAR(30) or MVARCHAR(30)	Name of the user with privileges
2	DBA_PRIVILEGE	CHAR(1)	DBA privilege: Y: Has the DBA privilege N: Does not have the DBA privilege

Number	Column name	Data type	Contents
3	SCHEMA_PRIVILEGE	CHAR(1)	Schema definition privilege: Y: Has the schema definition privilege S: Owns a schema N: Does not have the schema definition privilege The initial value is N.
4	CREATE_TIME	CHAR(14)	Schema creation date and time (*YYYYMMDDHHMMSS*) The initial value is a null value; also a null value when DROP SCHEMA is executed.
5	AUDIT_PRIVILEGE	CHAR(1)	Audit privilege status: Y: Granted Null value: Not granted Null value for any user who is not the auditor.
6	AUTH_ERR_COUNT	SMALLINT	Number of consecutive certification failures Null value if the number of consecutive certification failures is not specified, the number of consecutive user certification failures is 0, or the number of continuous certification failures has been cleared.
7	CON_LOCK_TIME	TIMESTAMP(0)	Consecutive certification failure account lock date and time Null value if the number of consecutive certification failures is not specified or if the consecutive certification failure account lock state has not occurred.#
8	PWD_LOCK_TIME	TIMESTAMP(0)	Password-invalid account lock date and time Null value if a password character string limit is not specified or if the password-invalid account lock state has not occurred.
9	PASSWORD_TEST	CHAR(1)	Password limit violation type code L: Minimum number of allowed bytes U: Specification of authentication indicator prohibited S: Specification of single-character type prohibited Null value if the user for whom the password-invalid account lock state occurs has not been prechecked or if there is no violation after the precheck.

#: If the consecutive certification failure account lock is set and no connection is established after the specified account lock period has elapsed, a null value is not set even if the consecutive certification failure account lock state has not occurred.

(7) SQL_RDAREA_PRIVILEGES table

This table manages the assignment of RDAREA usage privileges. (Each row describes information on one user of one RDAREA.)

The following table shows the contents of the SQL_RDAREA_PRIVILEGES table.

Table G-9: SQL_RDAREA_PRIVILEGES table contents

Number	Column name	Data type	Contents
1	GRANTEE	VARCHAR(30) or MVARCHAR(30)	Name of the user with the RDAREA usage privilege or PUBLIC
2	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the RDAREA
3	GRANT_TIME	CHAR(14)	Date and time at which the relevant privilege was granted (*YYYYMMDDHHMMSS*)

(8) SQL_TABLE_PRIVILEGES table

This table manages the granting of table access privileges. (Each row describes information on one user.)

Rows of the SQL_TABLE_PRIVILEGES table are created when users are granted table access privileges by GRANT. Rows are deleted when all of a user's privileges are revoked by REVOKE.

The following table shows the contents of the SQL_TABLE_PRIVILEGES table.

Table G-10: SQL_TABLE_PRIVILEGES table contents

Number	Column name	Data type	Contents
1	GRANTOR	VARCHAR(30) or MVARCHAR(30)	Name of the user granting the table access privileges or the definer of the public view table
2	GRANTEE	VARCHAR(30) or MVARCHAR(30)	Name of the user who receives table access privilege, or PUBLIC
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which access privilege is to be granted. PUBLIC for a public view table.
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which access privileges are to be granted

Number	Column name	Data type	Contents
5	SELECT_PRIVILEGE	CHAR(1)	SELECT privilege status: G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
6	INSERT_PRIVILEGE	CHAR(1)	INSERT privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
7	DELETE_PRIVILEGE	CHAR(1)	DELETE privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
8	UPDATE_PRIVILEGE	CHAR(1)	UPDATE privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
9	GRANT_TIME	CHAR(14)	Date and time at which the relevant privilege was granted (*YYYYMMDDHHMMSS*)
10	GRANTEE_TYPE	CHAR(1)	Null value (fixed)

(9) SQL_VIEW_TABLE_USAGE table

This table manages information of the base tables that serve as the basis for view tables. (Each row describes information on one view table.)

The following table shows the contents of the SQL_VIEW_TABLE_USAGE table.

Table G-11: SQL_VIEW_TABLE_USAGE table contents

Number	Column name	Data type	Contents
1	VIEW_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of a view table or PUBLIC for a public view table
2	VIEW_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a view table
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the base table or the resource to be used, or PUBLIC for a public view table or public routine

Number	Column name	Data type	Contents
4	BASE_TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the base table or the resource to be used
5	BASE_TYPE	CHAR(1)	Type of the base table or the resource to be used R: Real table V: View table P: User-defined function (excluding plug-in functions)
6	BASE_ROUTINE_CREATOR	VARCHAR(30) or MVARCHAR(30)	User who defined a public function if the resource to be used is the public function Null value if the resource to be used is not a public function

(10) SQL_VIEWS table

This table manages view table definition information. (Each row describes information on one view table.)

The following table shows the contents of the SQL_VIEWS table.

Table G-12: SQL_VIEWS table contents

Number	Column name	Data type	Contents
1	VIEW_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of a view table or PUBLIC for a public view table
2	VIEW_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a view table
3	SOURCE_ORDER	INTEGER	Order if source is divided and stored in multiple rows (1-*n*)
4	IS_UPDATABLE	CHAR(3)	Update possibility: YES: Possible NO: Not possible
5	VIEW_DEFINITION	VARCHAR(32000) or MVARCHAR(32000)	View definition source statements
6	VIEW_ID	INTEGER	View ID

(11) SQL_DIV_TABLE table

This table manages table partitioning information in the database. (Each row describes information on one table.)

The following table shows the contents of the SQL_DIV_TABLE table.

Table G-13: SQL_DIV_TABLE table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of a view table
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a view table
3	DIV_NO	INTEGER	Partitioning condition specification order (unique value beginning with 1 for the corresponding table, which is obtained by adding 1 to the partitioning condition specification order)
4	TABLE_ID	INTEGER	Table ID
5	DCOND	CHAR(2)	Partitioning condition code The partitioning storage condition value is stored in character format; the storable values are =, ^=, <, <=, >, and >=; if <> or != is specified, it is stored as ^=. For a matrix-partitioned table, <= is stored. Nothing is stored if no partitioning storage condition is specified or if hash partitioning is specified.
6	DCVALUES	VARCHAR(256) or MVARCHAR(256)	Partitioning condition value The value to be stored is not converted even if a character set is specified for a partitioning key. Null value if no partitioning storage condition is specified or if hash partitioning is specified.
7	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of storage RDAREA
8	DCVALUES2	VARCHAR(255) or MVARCHAR(255)	Second dimension key partitioning condition value (The storage format is the same as that for DCVALUES.) Null value for a table that is not a matrix-partitioned table and for a matrix-partitioned table for which no boundary value is specified.

(12) SQL_INDEX_COLINF table

This table manages index column information. (Each row describes information on one index.)

The following table shows the contents of the SQL_INDEX_COLINF table.

Table G-14: SQL_INDEX_COLINF table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains an index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	INDEX_ORDER	INTEGER	Order of columns comprising the index (integer beginning with 1, which identifies the name order of columns comprising the index)
6	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (name of columns comprising the index)
7	ASC_DESC	CHAR(1)	Ascending or descending order: A: Ascending order D: Descending order Empty: (for plug-in indexes) (If descending order is specified for a single-column index, it is stored as ascending order.)

(13) SQL_DIV_INDEX table

This table manages index partitioning information (partitioning conditions and names of storage RDAREAs specified by CREATE TABLE). (Each row describes information on one index.)

The following table shows the contents of the SQL_DIV_INDEX table.

Table G-15: SQL_DIV_INDEX table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains an index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name

Number	Column name	Data type	Contents
4	DIV_NO	INTEGER	RDAREA definition order (unique value beginning with 1 for the corresponding index which is obtained by adding 1 to the RDAREA definition order)[#]
5	INDEX_ID	INTEGER	Index ID
6	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of partitioned storage RDAREA comprising the index)

#: This value is not related to DIV_NO of SQL_DIV_TABLE.

(14) SQL_DIV_COLUMN table

This table manages BLOB-type column partitioning information (name of storage RDAREA specified by CREATE TABLE). (Each row describes information on one column.)

The following table shows the contents of the SQL_DIV_COLUMN table.

Table G-16: SQL_DIV_COLUMN table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name
4	DIV_NO	INTEGER	Storage order
5	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user LOB RDAREA
6	STORE_NO	INTEGER	Always 1
7	MASTER_RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of user RDAREA for the corresponding table
8	N_LEVEL	SMALLINT	Number of levels Null value for BLOB type columns.
9	COMPONENT_NAME	VARCHAR(30) or MVARCHAR(30)	Component name Null value for BLOB type columns.
10	LOB_NO	SMALLINT	LOB attribute number Null value for BLOB type columns.

(15) SQL_ROUTINES table

This table manages routine definition information. (Each row describes information on one routine.)

The following table shows the contents of the `SQL_ROUTINES` table.

Table G-17: SQL_ROUTINES table contents

Number	Column name	Data type	Contents
1	`ROUTINE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Routine owner `PUBLIC` for public routines
2	`ROUTINE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Routine name[#9]
3	`OBJECT_ID`	`INTEGER`	Object ID
4	`SPECIFIC_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Specific name[#2]
5	`ROUTINE_TYPE`	`CHAR(1)`	Routine type: `P`: Procedure `F`: Function
6	`ROUTINE_VALID`	`CHAR(1)`	Validity flag: `Y`: Validity routine `N`: Invalidity routine
7	`INDEX_VALID`	`CHAR(1)`	Index status change flag: `Y`: Index status valid[#1] `N`: Index status invalid[#1]
8	`CREATE_TIME`	`CHAR(14)`	Routine creation date and time (*YYYYMMDDHHMMSS*) SQL analysis time for SQL procedure statements or definition creation time for external routines
9	`ALTER_TIME`	`CHAR(14)`	Routine re-creation date and time (*YYYYMMDDHHMMSS*) (The initial value is a null value.)
10	`OBJECT_SIZE`	`INTEGER`	Object size (in bytes) `0` for external routines
11	`SOURCE_SIZE`	`INTEGER`	Definition source size (bytes) `0` for external routines and registry operation procedures
12	`ISOLATION_LEVEL`	`SMALLINT`	Data guarantee level (0-2) Valid for procedures

Number	Column name	Data type	Contents
13	OPTIMIZE_LEVEL	INTEGER	SQL optimization option (converted to decimal format) Specifies the value of OPTIMIZE LEVEL for CREATE PROCEDURE, ALTER PROCEDURE, CREATE TYPE, or ALTER ROUTINE.
14	SQL_LEVEL	SMALLINT	SQL level (0-2) Valid for procedures
15	N_PARAM	INTEGER	Number of parameters
16	N_RESOURCE	INTEGER	Number of resources used in an object
17	PARAM_LOCATION	INTEGER	Start position of a procedure statement in a definition source statement.#7

18	ROUTINE_
	COMMENT
		VARCHAR(255)
		or
		MVARCHAR(255)
			Comment
			(The initial value is a null value.)

19	DEF_SOURCE	BLOB	Definition source statement (not including compiler options) Null value for a system definition function, for a registry operation procedure, and for a trigger action procedure.
20	ROUTINE_ADT_OWNER	VARCHAR(30)	Owner of the abstract data type that defined routines Null value for a routine that is not defined in an abstract data type.
21	ROUTINE_ADT_NAME	VARCHAR(30)	Name of the abstract data type that defined routines Null value for a routine that is not defined in an abstract data type.
22	ROUTINE_BODY	CHAR(1)	Function routine type: S: SQL procedure E: External routine T: Trigger action procedure Null value for a procedure (excluding a trigger action procedures) that is not a foreign routine.
23	FUNCTION_TYPE	CHAR(1)	Function type: C: System-defined function constructor Empty: User-defined function Null value for a procedure.

Number	Column name	Data type	Contents
24	EXTERNAL_NAME	VARCHAR(255)	External routine name (*library-name* ! *operation-name*) or a Java method name if defined in Java Null value if this is not for an external stored routine.
25	EXTERNAL_LANGUAGE	CHAR(20)	External descriptive language type: C: C language Java: Java language Null value if this is not for an external stored routine.
26	PARAMETER_STYLE	VARCHAR(20)	Parameter style (external stored routine type) PLUGIN: Plug-in RDSQL: RDSQL Java: Java Null value if this is not for an external stored routine.
27	ENCAPSULATION_ LEVEL	VARCHAR(10)	Encapsulation level (PUBLIC, PRIVATE, or PROTECTED) Null value for a routine that is not defined in an abstract data type.
28	RETURN_UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of a return value data type Null value if the return value is not a user-defined function.
29	RETURN_UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a return value data type Null value if the return value is not a user-defined function.
30	RETURN_UDT_TYPE_ ID	INTEGER	ID of a return value data type Null value if the return value is not a user-defined function.
31	RETURN_DATA_TYPE	CHAR(24)	Return value data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table. Null value if the return value data type is not a function.
32	RETURN_DATA_TYPE _CODE	SMALLINT	Code for a return value data type For details about the storage format, see the DATA_TYPE_CODE column in the SQL_COLUMNS table. Null value if the return value data type is not a function.

Number	Column name	Data type	Contents
33	RETURN_DATA_ LENGTH_CODE	SMALLINT	Code for a return value data length[#3] Null value for a procedure.
34	RETURN_DATA_ LENGTH	CHAR(7)	Return value data length stored right-justified in character format (spaces are used for leading zeros) Null value for a procedure.
35	RETURN_LOB_ LENGTH_CODE	CHAR(8)	Code for a return value BLOB data length[#4, #8] Null value for a procedure or if the return value is not a BLOB or BINARY function.
36	RETURN_LOB_ LENGTH	CHAR(20)	Specification value of a return value BLOB data length Right-justified in character format (spaces are used for leading zeros) Null value for a procedure or if the return value is not a BLOB or BINARY function.
37	RETURN_LOB_ LENGTH_TYPE	CHAR(1)	Type of a return value BLOB data length: K: K specified M: M specified G: G specified Empty: Default Null value for a procedure or if the return value is not a BLOB or BINARY function.
38	ADDITIONAL_ OPTIMIZE_LEVEL	INTEGER	Extended SQL optimization option (converted to decimal format) Specifies the value of ADD OPTIMIZE LEVEL for CREATE PROCEDURE, ALTER PROCEDURE, CREATE TYPE, or ALTER ROUTINE. Null value if the routine was created by HiRDB of version 06-00 or earlier.
39	CLASS_NAME	VARCHAR(255)	*package-name*.*class-name*[#5] Null value if the foreign routine is not coded in Java.
40	JAR_NAME	VARCHAR(255)	Java archive file name Null value if the foreign routine is not coded in Java.
41	DYNAMIC_RESULT_ SETS	SMALLINT	Maximum number of result sets to be returned Null value if no maximum number is specified for the result sets.
42	SQL_ SPECIFICATION	CHAR(1)	Information used by the system

Number	Column name	Data type	Contents
43	RETURNS_JAVA_ DATA_TYPE	VARCHAR(255)	Java return value's data type corresponding to return value's data type#6 Null value if the foreign routine is not coded in Java.
44	RETURNS_JAVA_ DATA_TYPE_CODE	INTEGER	Java return value's data type code corresponding to return value's data type#6 Null value if the foreign routine is not coded in Java.
45	RETURN_DATA_ MAX_ELM	SMALLINT	Maximum number of elements for return value's data type Null value if ARRAY is not specified for the return value data type.
46	N_JAVA_RESULT_ SETS	INTEGER	Number of java.sql.ResultSet[]s specified Null value if java.sql.ResultSet[] is not specified.
47	FOR_UPDATE_EXCLUSIV E_LOCK	CHAR(1)	Whether ISOLATION LEVEL is a value other than 2 and FOR UPDATE EXCLUSIVE is specified Y: Yes Null value: No Null value for a routine created with a HiRDB version earlier than 07-01, if FOR UPDATE EXCLUSIVE has not been specified, or if the ISOLATION LEVEL value is 2.
48	SUBSTR_LENGTH	SMALLINT	Specification value of SUBSTR LENGTH of the SQL compile option Null value for a routine created with a HiRDB version earlier than 08-00 or when the character code type is not Unicode (UTF-8).
49	RETCSET_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Character set owner for the data type of return value (always MASTER) Null value if no character set is specified for the data type of the return value.
50	RETCSET_NAME	VARCHAR(30) or MVARCHAR(30)	Character set name for the data type of the return value: EBCDIK: EBCDIK is specified for the character set. UTF16: UTF16 is specified for the character set. Null value if no character set is specified for the data type of the return value.

Number	Column name	Data type	Contents
51	RETCSET_ID	INTEGER	Character set ID for the data type of the return value Null value if no character set is specified for the data type of return value. For details about character set IDs, see the *HiRDB Version 9 Installation and Design Guide*.
52	ROUTINE_CREATOR	VARCHAR(30) or MVARCHAR(30)	User who defined the public routine Null value if the routine is not a public routine

#1: Index information in the routine is invalid (the routine cannot be executed). In this case, SQL objects must be re-created by ALTER ROUTINE or ALTER PROCEDURE.

#2: For procedures, this name is the same as the routine name; for functions, the system internally generates a name from the routine name and object ID as follows:

F routine name (up to 19 bytes) object ID (10 bytes)

#3: For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

#4: The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

#5: The following shows the storage format for *package-name*.*class-name*:

- Package name specified

  package-name.*class-name*

- Package name not specified

  class-name

#6: The following Java data types are stored as a character string in RETURN_JAVA_DATA_TYPE. The Java data types expressed in hexadecimal numbers are stored in RETURN_JAVA_DATA_TYPE_CODE.

Java data type	Value in hexadecimal
byte[]	1000
byte[][]	100A
short	1002
short[]	1003

Java data type	Value in hexadecimal
int	1004
int[]	1005
float	1006
float[]	1007
double	1008
double[]	1009
java.match.BigDecimal	2000
java.match.BigDecimal[]	2001
java.lang.String	2002
java.lang.String[]	2003
java.sql.Date	2004
java.sql.Date[]	2005
java.sql.Time	2006
java.sql.Time[]	2007
java.lang.Double	2008
java.lang.Double[]	2009
java.lang.Float	200A
java.lang.Float[]	200B
java.lang.Integer	200C
java.lang.Integer[]	200D
java.lang.Short	200E
java.lang.Short[]	200F
java.sql.Timestamp	2010
java.sql.Timestamp[]	2011
void	0000

#7: The location at which the procedure statement starts is counted from the top of the SQL statement, beginning at 1. For an external routine (Java stored routine), the location at which the external routine specification (EXTERNAL NAME clause) starts is counted from the beginning of the SQL statement. A value of 0 is set for the following:

- External routine (excluding Java stored routines)
- Registry manipulation procedure
- Trigger action procedure

#8: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

#9: For a trigger action procedure, the following routine name (22 bytes long) is stored:

'(TRIG*yyyymmddhhmmssth*)'

yyyymmddhhmmssth: Time stamp at the time of trigger definition (units: 1/100 seconds)

(16) SQL_ROUTINE_RESOURCES table

This table manages resource information used in routines. (*n* rows describe information on one routine.)

The following table shows the contents of the SQL_ROUTINE_RESOURCES table.

Table G-18: SQL_ROUTINE_RESOURCES table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner PUBLIC for public routines
2	ROUTINE_NAME	VARCHAR(30) or MVARCHAR(30)	Routine name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name#1
4	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Resource owner or PUBLIC for a public view table or a public routine
5	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	Resource identifier
6	BASE_TYPE	CHAR(1)	Resource type: R: Base table V: View table that can be updated U: Read-only view table I: Index D: Data type P: Routine T: Trigger Q: Sequence generator

Number	Column name	Data type	Contents
7	ROUTINE_TYPE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of abstract data type for routine defined in abstract data type Null value for routines that are not defined in an abstract data type.
8	ROUTINE_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of abstract data type for routine defined in abstract data type Null value for routines that are not defined in an abstract data type.
9	SELECT_OPERATION#2	CHAR(1)	Retrieval target specification status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3
10	INSERT_OPERATION#2	CHAR(1)	Data insertion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3
11	UPDATE_OPERATION#2	CHAR(1)	Data update target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3
12	DELETE_OPERATION#2	CHAR(1)	Data deletion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3
13	LOCK_OPERATION#2	CHAR(1)	Data insertion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3
14	PURGE_OPERATION#2	CHAR(1)	Whether or not a data deletion target is specified in a PURGE TABLE statement: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.#3

Number	Column name	Data type	Contents
15	BASE_ROUTINE_CREATOR	VARCHAR(30) or MVARCHAR(30)	User who defined a public routine if the resource to be used is the public routine Null value if the resource to be used is not a public routine
16	ROUTINE_CREATOR	VARCHAR(30) or MVARCHAR(30)	User who defined the public routine. Null value if this is not for a public routine.

#1: For procedures, this name is the same as the routine name; for functions, the system internally generates a name from the routine name and object ID as follows:

'F' *routine name* (up to 19 bytes) *object ID* (10 bytes)

#2: If a view table is used as an SQL object, information that merges the operation types of all view tables being used is set in the base table (the highest order base table if the base table is a view table) that is the base for the view table being used as the SQL object.

#3: If the type of resource being used is a view table (V), a null value is set for a view table that is not actually contained in the SQL object.

(17) SQL_ROUTINE_PARAMS table

This table manages parameter information in routines. (*n* rows describe information on one routine.)

The following table shows the contents of the SQL_ROUTINE_PARAMS table.

Table G-19: SQL_ROUTINE_PARAMS table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner PUBLIC for public routines
2	ROUTINE_NAME	VARCHAR(30) or MVARCHAR(30)	Routine name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name
4	PARAMETER_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter name#4
5	PARAMETER_NO	INTEGER	Parameter specification sequence (a unique number within the routine beginning with 1)
6	DATA_TYPE	CHAR(24)	Data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table.

Number	Column name	Data type	Contents
7	DATA_LENGTH	CHAR(7)	Data length stored right-justified in character format (spaces are used for leading zeros) Null value if the data type is BLOB, BINARY, or a user-defined type.
8	LOB_LENGTH	CHAR(20)	Column length specification value right-justified in character format (spaces are used for leading zeros) Null value if the data type is not BLOB or BINARY.
9	LOB_LENGTH_TYPE	CHAR(1)	Column length type: K: K specified M: M specified G: G specified Empty: Default Null value if the data type is not BLOB.
10	PARAMETER_MODE	CHAR(5)	Parameter I/O mode: IN: Input mode NOUT: Output mode INOUT: Input/output mode NONE: Other than above
11	DATA_TYPE_CODE	SMALLINT	Data type code For details about the storage format, see the DATA_TYPE_CODE column of the SQL_COLUMNS table.
12	DATA_LENGTH_CODE	SMALLINT	Data length code#1 Null value if the data type is BLOB, BINARY, or a user-defined type.
13	LOB_LENGTH_CODE	CHAR(8)	Column length specification value#2, #3 Null value if the data type is not BLOB or BINARY.
14	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of a data type parameter Null value if the parameter is a system-defined type.
15	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a data type parameter Null value if the parameter is a system-defined type.
16	UDT_TYPE_ID	INTEGER	ID of a data type parameter Null value if the parameter is a system-defined type.

Number	Column name	Data type	Contents
17	JAVA_DATA_TYPE	VARCHAR(255)	Data type of the corresponding Java parameter For the storage format, see the RETURNS_JAVA_DATA_TYPE column in the SQL_ROUTINES table. Null value if the foreign routine is not coded in Java.
18	JAVA_DATA_TYPE_CODE	INTEGER	Data type code of the corresponding Java parameter For the storage format, see the RETURNS_JAVA_DATA_TYPE_ CODE column in the SQL_ROUTINES table. Null value if the foreign routine is not coded in Java.
19	MAX_ELM	SMALLINT	Maximum number of parameter elements Null value if the number of parameter elements is not specified.
20	TRIGGER_COLUMN	CHAR(1)	Parameter information for the column specified by an old or new values correlation name of the trigger action procedure O: Column referenced by an old values correlation name N: Column referenced by a new values correlation name Null value: Neither of the above Null value if the parameter is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.
21	TRIGGER_TABLE_ID	INTEGER	Table ID that defines the column before it is replaced with a parameter Null value if the ID is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.
22	TRIGGER_COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name before being replaced with a parameter Null value if the name is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.

Number	Column name	Data type	Contents
23	PARMCSET_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Character set owner for the data type (always MASTER) Null value if no character set is specified for the data type.
24	PARMCSET_NAME	VARCHAR(30) or MVARCHAR(30)	Character set name for the data type: EBCDIK: EBCDIK is specified for the character set. UTF16: UTF16 is specified for the character set. Null value if no character set is specified for the data type.
25	PARMCSET_ID	INTEGER	Character set ID for the data type Null value if no character set is specified for the data type. For details about character set IDs, see the *HiRDB Version 9 Installation and Design Guide*.

#1: For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

#2: The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

#3: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

#4: For a trigger action procedure, the following parameter name (27 bytes long) is used:

'(T#*tbl_id*#*col_id*#*nnnnn*)'

tbl_id

Table ID (hexadecimal, 8 digits (If the number of digits is less than 8, the front portion is zero filled.))

col_id

Column ID (hexadecimal, 8 digits (If the number of digits is less than 8, the front portion is zero filled.))

nnnnn

00001: Parameter that corresponds to a column modified by an old values correlation name

00002: Parameter that corresponds to a column modified by a new values correlation name

(18) SQL_TABLE_STATISTICS table

This table manages table statistical information. (Each row describes information on one table.)

If there is no statistical information (for example, immediately following CREATE TABLE), the contents of this table are empty.

The following table shows the contents of the SQL_TABLE_STATISTICS table.

Table G-20: SQL_TABLE_STATISTICS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	N_PAGE	FLOAT	Number of pages stored (statistical information) Null value if lvl1 is specified for the -c option of pdgetcst
4	N_ROW	FLOAT	Total number of rows (statistical information)
5	UPDATE_TIME	CHAR(14)	Update date and time (*YYYYMMDDHHMMSS*)

(19) SQL_COLUMN_STATISTICS table

This table manages column statistical information. (Each row describes information on one column.)

If there is no statistical information (for example, immediately after CREATE TABLE), the contents of this table are empty.

The following table shows the contents of the SQL_COLUMN_STATISTICS table.

Table G-21: SQL_COLUMN_STATISTICS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains a column
3	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name
4	N_UNIQUE	FLOAT	Number of unique values (statistical information)

Number	Column name	Data type	Contents
5	`N_MAX_DUP_KEY`	`FLOAT`	Maximum number of duplicate key values (statistical information)
6	`N_MIN_DUP_KEY`	`FLOAT`	Minimum number of duplicate key values (statistical information)
7	`N_NULL`	`FLOAT`	Number of null values
8	`UPDATE_TIME`	`CHAR(14)`	Update date and time (*YYYYMMDDHHMMSS*)
9	`RANGE_VALUES`	`VARCHAR(2872)`	Column value frequency distribution information (statistical information)#

#: The maximum and minimum column values set in the `pdgetcst` parameter file are stored in the `RANGE_VALUES` column after being converted into an internal format. To reference these maximum and minimum values, the SQL described as follows must be executed. The retrieval results are displayed in hexadecimal.

- SQL for retrieving the maximum column value

```
SELECT HEX(SUBSTR("RANGE_VALUE",33,a))
    FROM "MASTER".SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

For *a*, specify the data length of the column in bytes. If the data is of the character string type, it is truncated to 16 bytes, so a value equal to or less than 16 must be specified.

- SQL for retrieving the minimum column value

- When the column's data type is `DECIMAL` or `NUMERIC` and the precision is 32 digits or more

```
SELECT HEX(SUBSTR("RANGE_VALUE",33,a))
     FROM "MASTER".SQL_COLUMN_STATISTICS
     WITHOUT LOCK NOWAIT
```

- When the column's data type is neither `DECIMAL` nor `NUMERIC`

```
SELECT HEX(SUBSTR("RANGE_VALUES"),49,a)
     FROM "MASTER".SQL_COLUMN_STATISTICS
     WITHOUT LOCK NOWAIT
```

For *a*, specify the data length of the column in bytes. If the data is of the character string type, it is truncated to 16 bytes, so a value equal to or less than 16 must be specified.

Example

Referencing the maximum column value of an `INT`-type column

```
SELECT HEX(SUBSTR("RANGE_VALUE"),33,4)
    FROM "MASTER".SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

Output result (when maximum column value is 10)

```
'0000000A'
```

(20) SQL_INDEX_STATISTICS table

This table manages index statistical information. (Each row describes information on one index.)

If there is no statistical information (for example, immediately following `CREATE TABLE`), the contents of this table are empty.

The following table shows the contents of the `SQL_INDEX_STATISTICS` table.

Table G-22: SQL_INDEX_STATISTICS table contents

Number	Column name	Data type	Contents
1	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Table owner
2	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the table that contains the index
3	`INDEX_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Index name
4	`N_ENTRY`	`FLOAT`	Number of key entries (statistical information)
5	`N_IXPG`	`FLOAT`	Number of leaf pages (statistical information)
6	`N_LEVEL`	`SMALLINT`	Number of levels (statistical information)
7	`SEQ_RATIO`	`INTEGER`	Sequential level (statistical information)
8	`UPDATE_TIME`	`CHAR(14)`	Update date and time (*YYYYMMDDHHMMSS*)

(21) SQL_DATATYPES table

This table manages user-defined type information (each row defines information on one user-defined type).

The following table shows the contents of the `SQL_DATATYPES` table.

Table G-23: SQL_DATATYPES table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the user-defined type
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user-defined type
3	META_TYPE	CHAR(1)	Type of the user-defined type: A: Abstract data type
4	TYPE_ID	INTEGER	ID of the user-defined type
5	N_ATTR	SMALLINT	Number of attributes
6	CREATE_TIME	CHAR(14)	Creation date and time (*YYYYMMDDHHMMSS*)
7	N_SUBTYPE	INTEGER	Number of subtypes
8	SOURCE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the supertype abstract data type Null value if there is no supertype abstract data type.
9	SOURCE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the supertype abstract data type Null value if there is no supertype abstract data type.
10	SOURCE_TYPE_ID	INTEGER	ID of the supertype abstract data type Null value if there is no supertype abstract data type.
11	ROOT_TYPE_ID	INTEGER	ID of the highest order abstract data type if the supertype abstract data type also has a supertype
12	LEVEL_NO	SMALLINT	Number of generations from highest order supertype abstract data type if the supertype abstract data type also has a supertype
13	TYPE_COMMENT	VARCHAR(255)	Comment The initial value is a null value. Null value if there is no comment.
14	N_LOB_ATTR	SMALLINT	Number of BLOB-type attributes
15	N_ADT_ATTR	SMALLINT	Number of abstract-data-type attributes
16	N_LARGE_BINARY_ATTR	SMALLINT	Number of attributes for BINARY-type data of 32,001 bytes or more

(22) SQL_DATATYPE_DESCRIPTORS table

This table manages user-defined type attribute information. (Each row describes information on one attribute.)

The following table shows the contents of the SQL_DATATYPE_DESCRIPTORS table.

Table G-24: SQL_DATATYPE_DESCRIPTORS table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the user-defined type
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user-defined type
3	OBJECT_NAME	VARCHAR(30) or MVARCHAR(30)	Attribute name
4	TYPE_ID	INTEGER	ID of the user-defined type
5	META_TYPE	CHAR(1)	Type of the user-defined type: S: System-defined type A: Abstract data type
6	ORDINAL_POSITION	SMALLINT	Order position
7	ENCAPSULATION_LEVEL	VARCHAR(10)	Encapsulation level (PUBLIC, PRIVATE, or PROTECTED)
8	IS_NULLABLE	CHAR(3)	Column null value information YES: Null value allowed NO: Null values not allowed
9	DATA_TYPE	CHAR(24)	Data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table.
10	DATA_TYPE_CODE	SMALLINT	Data type code For details of the storage format, see the DATA_TYPE_CODE column of the SQL_COLUMNS table.
11	DATA_LENGTH_CODE	SMALLINT	Data length code#1
12	DATA_LENGTH	CHAR(7)	Data length stored right-justified in character format (spaces are used for leading zeros)
13	LOB_LENGTH_CODE	CHAR(8)	BLOB attribute length code#2, #3 Null value if the code is not for BLOB or BINARY.

Number	Column name	Data type	Contents
14	LOB_LENGTH	CHAR(20)	BLOB attribute length specification value stored right-justified in character format (spaces are used for leading zeros) Null value if the value is not for BLOB or BINARY.
15	LOB_LENGTH_TYPE	CHAR(1)	BLOB attribute length type (unit): K: K specified M: M specified G: G specified Empty: Default Null value if the type is not BLOB.
16	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type for an abstract data type attribute that has another abstract data type Null value if the owner is for the system definition type.
17	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type for an abstract data type attribute that has another abstract data type Null value if the name is for the system definition type.
18	DATA_COMMENT	VARCHAR(255)	Comment (The initial value is a null value; null value is also used if there is no comment.)
19	NO_SPLIT	CHAR(1)	Whether or not NO SPLIT is specified: Y: Specified Null value: No specification

#1: For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

#2: The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

#3: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(23) SQL_TABLE_RESOURCES table

This table manages resource information used in tables. (Each row describes information on one resource.)

The following table shows the contents of the SQL_TABLE_RESOURCES table.

Table G-25: SQL_TABLE_RESOURCES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	Type of the resource used: A: Abstract data type

(24) SQL_PLUGINS table

This table manages plug-in information. (Each row describes information on one plug-in.)

The following table shows the contents of the SQL_PLUGINS table.

Table G-26: SQL_PLUGINS table contents

Number	Column name	Data type	Contents
1	PLUGIN_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Plug-in owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	PLUGIN_TYPE	CHAR(1)	Plug-in type: D: Data type plug-in I: Index type plug-in
4	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type or index type
5	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type or index type
6	CREATE_TIME	CHAR(14)	Plug-in creation time
7	PLUGIN_LIB_NAME	VARCHAR(255)	Library path name
8	PLUGIN_COMMENT	VARCHAR(255)	Comment The initial value is a null value. Null value if there is no comment.

Number	Column name	Data type	Contents
9	PLUGIN_VERSION	VARCHAR(10)	Plug-in version Null value if the plug-in is the initial version.
10	PLUGIN_EXT_FUNC	VARCHAR(255)	Plug-in extended function code (information used in the system)

(25) SQL_PLUGIN_ROUTINES table

This table manages plug-in routine information. (Each row describes information on one plug-in routine.)

The following table shows the contents of the SQL_PLUGIN_ROUTINES table.

Table G-27: SQL_PLUGIN_ROUTINES table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	OPERATION_NAME	VARCHAR(255)	Operation name
4	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name#
5	N_PARAM	INTEGER	Number of parameters
6	TIMING_DESCRIPTOR	VARCHAR(30)	Timing descriptor
7	OPERATION_ DESCRIPTOR	VARCHAR(255)	Operation modification information

#: A plug-in routine is named in the following format:

'P' *function-name registration-date-and-time*

P

Code that indicates a function provided by a plug-in

function-name

The leading characters (maximum 15 characters) are truncated so that the specific name is within 30 characters.

registration-date-and-time

Indicates the year, month, hour, minute, and second with 14 characters.

(26) SQL_PLUGIN_ROUTINE_PARAMS table

This table manages plug-in routine parameter information. (Each row describes information on one parameter.)

The following table shows the contents of the SQL_PLUGIN_ROUTINE_PARAMS table.

Table G-28: SQL_PLUGIN_ROUTINE_PARAMS table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name
4	PARAMETER_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter name
5	PARAMETER_MODE	CHAR(7)	Parameter I/O attribute: IN: Input mode OUT: Output mode INOUT: Input/output mode RETURNS: Return value attribute PICKUP: ROWID output attribute
6	PARAMETER_DESCRIPTOR	VARCHAR(255)	Parameter modification information Parameter modification information specified with the plug-in IDL is held as a character string without changes. Null value if no parameter modification information is specified.
7	SPECIFIC_BIND_OPERATION_NAME	VARCHAR(30) or MVARCHAR(30)	Specific bind operation name Null value if bind operation is not specified.

Number	Column name	Data type	Contents
8	PARAMETER_TYPE	CHAR(1)	Parameter mode: Empty: Normal (data type that can be handled by SQL) I: Indicator N: New data C: Current data D: dbifb K: Index key inf P: Pointer R: rowid U: utlifb T: Pointer These are plug-in specific parameter modes, except normal.
9	PARAMETER_NO	INTEGER	Parameter specification order position for abstract data type functions
10	DATA_TYPE	CHAR(24)	Parameter data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table. Null value if the parameter mode is D, K, P, R, U, or T.
11	DATA_TYPE_CODE	SMALLINT	Parameter data code Null value if the parameter mode is D, K, P, R, U, or T. For details about the storage format, see the DATA_TYPE_CODE column of the SQL_COLUMNS table.
12	DATA_LENGTH_CODE	SMALLINT	Parameter data type definition length code#1 Null value if the parameter mode is D, K, P, R, U, or T.
13	DATA_LENGTH	CHAR(7)	Parameter data definition length stored right-justified in character format (spaces are used for leading zeros) Null value if the parameter mode is D, K, P, R, U, or T.
14	LOB_LENGTH_CODE	CHAR(8)	LOB column length code or BINARY column length code#2, #3 Null value if the parameter mode is normal and the data type is not BLOB or BINARY.

Number	Column name	Data type	Contents
15	LOB_LENGTH	CHAR(20)	LOB column length specification value or BINARY column length specification value Stored in the character format, right-justified (higher-order 0s are left as spaces). Null value if the parameter mode is normal and the data type is not BLOB or BINARY.
16	LOB_LENGTH_TYPE	CHAR(1)	LOB column length type (unit): K: K specified M: M specified G: G specified Empty: Default Null value if the parameter mode is normal and the data type is not BLOB or BINARY.
17	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Parameter data type owner Null value if the data type is not a user-defined type.
18	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter data type name Null value if the data type is not a user-defined type.
19	UDT_TYPE_ID	INTEGER	Parameter data type ID Null value if the data type is not a user-defined type.

#1: For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

#2: The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

#3: SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(27) SQL_INDEX_TYPES table

This table manages index type information. (Each row describes information on one index type.)

The following table shows the contents of the SQL_INDEX_TYPES table.

Table G-29: SQL_INDEX_TYPES table contents

Number	Column name	Data type	Contents
1	INDEX_TYPE_ SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index type owner
2	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Index type name
3	INDEX_TYPE_ID	INTEGER	Index type ID
4	CREATE_TIME	CHAR(14)	Creation time
5	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Abstract data type owner
6	ADT_NAME	VARCHAR(30) or MVARCHAR(30)	Abstract data type name
7	N_FUNCTION	INTEGER	Number of abstract data type functions that can be used in an index-type-defined index

(28) SQL_INDEX_RESOURCES table

This table manages resource information used in indexes. (Each row describes information on one resource.)

The following table shows the contents of the SQL_INDEX_RESOURCES table.

Table G-30: SQL_INDEX_RESOURCES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the index definition table
2	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	Type of the resource used: I: Index type

(29) SQL_INDEX_DATATYPE table

This table manages target item information in indexes. (Each row describes information on one target item (one level).)

The following table shows the contents of the SQL_INDEX_DATATYPE table.

Table G-31: SQL_INDEX_DATATYPE table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (index column name)
6	N_LEVEL	SMALLINT	Number of levels (number used to identify the name order of attributes constituting an abstract data type)
7	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type
8	ADT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type
9	ADT_ATTR_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type attribute
10	ADT_ATTR_ID	SMALLINT	Attribute position

(30) SQL_INDEX_FUNCTION table

This table manages abstract data type function information used in indexes. (Each row describes information on one abstract data type function.)

The following table shows the contents of the SQL_INDEX_FUNCTION table.

Table G-32: SQL_INDEX_FUNCTION table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name

Number	Column name	Data type	Contents
4	INDEX_ID	INTEGER	Index ID
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (index column name)
6	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner name of the abstract data type function
7	ADT_FUNCTION_ NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type function (routine name)
8	ADT_FUNCTION_ OBJECT_ID	INTEGER	Object ID of the abstract data type function

(31) SQL_TYPE_RESOURCES table

This table manages resource information used in user-defined types. (Each row describes information on one resource.)

The following table shows the contents of the SQL_TYPE_RESOURCES table.

Table G-33: SQL_TYPE_RESOURCES table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	User-defined type owner
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	User-defined type name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	ID of the resource used A: Abstract data type

(32) SQL_INDEX_TYPE_FUNCTION table

This table manages abstract data type function information that can be used in an index that defines index types. (Each row describes information on one index type.)

The following table shows the contents of the SQL_INDEX_TYPE_FUNCTION table.

Table G-34: SQL_INDEX_TYPE_FUNCTION table contents

Number	Column name	Data type	Contents
1	INDEX_TYPE_ SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index type owner
2	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
3	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type function
4	ADT_FUNCTION_ NAME	VARCHAR(30) or MVARCHAR(30)	ID of the abstract data type function#
5	ADT_FUNCTION_ OBJECT_ID	INTEGER	Object ID of the abstract data type function

#: This is not a specific name.

(33) SQL_EXCEPT table

This table manages index exclusion key value information. (Each row describes information on the exclusion key group for one index.) This table manages one exclusion key value (exclusion value group for multicolumn indexes) in each row.

The following table shows the contents of the SQL_EXCEPT table.

Table G-35: SQL_EXCEPT table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	TABLE_ID	INTEGER	Table ID

6	EXCEPT_VALUE	VARCHAR(573)
		or
		MVARCHAR(573)
			Contents of the exclusion key value
			The specified values for each column are delimited with a comma in a character format.
			The initial value is a null value.

(34) SQL_IOS_GENERATIONS table contents

This table manages the generation information of HiRDB file system areas when the inner replica facility is used. (Each row describes information on one HiRDB file system area.)

If the HiRDB Staticizer Option is not installed, this table is empty. However, if a database is created with HiRDB Staticizer Option installed, and then HiRDB Staticizer Option is removed, any data set in the table remains.

The following table shows the contents of the `SQL_IOS_GENERATIONS` table.

Table G-36: SQL_IOS_GENERATIONS table contents

Number	Column name	Data type	Contents
1	`FILE_SYSTEM_NAME`	`VARCHAR(165)`	HiRDB file system area name (absolute path name)
2	`GENERATION_NUMBER`	`SMALLINT`	Generation number
3	`SERVER_NAME`	`CHAR(8)`	Server name (`BES` or `SDS`)#
4	`ORIGINAL_FILE_SYSTEM_NAME`	`VARCHAR(165)`	Original HiRDB file system area name (absolute path name)

#: Even when a dictionary table of a HiRDB/Parallel Server is used in a HiRDB/Single Server without any modification, the server name is not changed.

If the name is less than 8 characters when left justified, the remaining spaces are filled with spaces.

(35) SQL_TRIGGERS table contents

This table manages the information of the triggers that are inside a schema. (Each row describes information on one trigger.)

The following table shows the contents of the `SQL_TRIGGERS` table.

Table G-37: SQL_TRIGGERS table contents

Number	Column name	Data type	Contents

1	`TRIGGER_SCHEMA`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Trigger owner

2	`TRIGGER_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Trigger name

3	`OBJECT_ID`	`INTEGER`	Object ID

Number	Column name	Data type	Contents

4	TABLE_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Owner of the table for which the trigger is defined.

5	TABLE_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Name of the table for which the trigger is defined.

6	TRIGGER_VALID	CHAR(1)	Trigger-enabling flag Y: Enabled N: Disabled Same value as the ROUTINE_VALID column of the SQL_ROUTINES table for the trigger action procedure
7	INDEX_VALID	CHAR(1)	Index-enabling flag Y: Enabled N: Disabled Same value as the INDEX_VALID column of the SQL_ROUTINES table for the trigger action procedure
8	ACTION_TIME	CHAR(1)	Trigger action timing A: AFTER B: BEFORE
9	EVENT	CHAR(1)	Trigger event type I: INSERT D: DELETE U: UPDATE
10	ACTION_TYPE	CHAR(1)	Trigger action unit R: ROW S: STATEMENT

11	OLD_ROW_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Old values correlation name (correlation name specified in OLD ROW)
			Null value if OLD ROW is not specified.

12	NEW_ROW_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			New values correlation name (correlation name specified in NEW ROW)
			Null value if NEW ROW is not specified.

13	CREATE_TIME	VARCHAR(16)	Trigger definition creation time

Number	Column name	Data type	Contents
14	ALTER_TIME	CHAR(14)	Trigger SQL object re-creation time Same value as the ALTER_TIME column of the SQL_ROUTINES table for the trigger action procedure Null value if a trigger SQL object is not re-created.
15	DEF_SOURCE_LEN	INTEGER	Trigger definition source length

16	SPECIFIC_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Specific name of the trigger action procedure

17	N_UPDATE_COLUMNS	SMALLINT	Number of trigger event columns Null value for an UPDATE trigger for which no INSERT trigger, DELETE trigger, or trigger event column is specified.
18	REFERENCING_TABLE_ID	INTEGER	Table ID of the referencing table Null value for a trigger that is not created by a referential constraint action.
19	REFERENCE_ACTION	CHAR(2)	Referential constraint operation type DC: ON DELETE CASCADE UC: ON UPDATE CASCADE Null value for a trigger that is not created by a referential constraint action.

20	CONSTRAINT_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Constraint name of referential trigger
			Null value for a trigger that is not created by a referential constraint action.

(36) SQL_TRIGGER_COLUMNS table contents

This table manages the list information of UPDATE trigger event columns. (Each row describes information on one trigger column.)

The following table shows the contents of the SQL_TRIGGER_COLUMNS table.

Table G-38: SQL_TRIGGER_COLUMNS table contents

Number	Column name	Data type	Contents

1	TRIGGER_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Trigger owner

2	TRIGGER_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Trigger name

Number	Column name	Data type	Contents

3	`TABLE_SCHEMA`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Owner of the table for which the trigger is defined.

4	`TABLE_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Name of the table for which the trigger is defined.

5	`COLUMN_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Column name specified for the column list

6	`TABLE_ID`	`INTEGER`	ID of the table for which the trigger is defined.

(37) SQL_TRIGGER_DEF_SOURCE table contents

This table manages the source information of trigger definitions. (Each row describes information on one trigger definition source.)

The following table shows the contents of the `SQL_TRIGGER_DEF_SOURCE` table.

Table G-39: SQL_TRIGGER_DEF_SOURCE table contents

Number	Column name	Data type	Contents

1	`TRIGGER_SCHEMA`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Trigger owner

2	`TRIGGER_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Trigger name

3	`TABLE_SCHEMA`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Owner of the table for which the trigger is defined.

4	`TABLE_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Name of the table for which the trigger is defined.

5	`SOURCE_NO`	`INTEGER`	Definition source serial number
6	`DEF_SOURCE`	`VARCHAR(32000)` or `MVARCHAR(32000)`	Definition source (excluding SQL compile options and `WITH PROGRAM`)

(38) SQL_TRIGGER_USAGE table contents

This table manages the resource information being referenced inside trigger action conditions. (Each row describes information on one resource name being referenced in a trigger action condition.)

The following table shows the contents of the SQL_TRIGGER_USAGE table.

Table G-40: SQL_TRIGGER_USAGE table contents

Number	Column name	Data type	Contents

1	TRIGGER_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Trigger owner

2	TRIGGER_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Trigger name

3	TABLE_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Owner of the table for which the trigger is defined.

4	TABLE_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Name of the table for which the trigger is defined.

5	BASE_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Owner of the resource being used
			PUBLIC for public routines

6	BASE_TABLE	VARCHAR(30)
		or
		MVARCHAR(30)
			Table name of the resource being used
			Null value if the type of the resource being used is F (function).

7	BASE_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Name of the resource being used (specific name or column name)

8	BASE_TYPE	CHAR(1)	Type of resource being used F: Function C: Column name
9	TABLE_ID	INTEGER	Table ID Null value if the type of the resource being used is F (function).
10	BASE_ID	INTEGER	ID of the resource being used (object ID or column ID)

Number	Column name	Data type	Contents

11	BASE_ROUTINE_CREATOR	VARCHAR(30)
		or
		MVARCHAR(30)
			User who defined a public function if the resource being used is a public function
			Null value if the resource being used is not a public function.

(39) SQL_PARTKEY table contents

This table manages the partitioning key information of matrix-partitioned tables. (Each row describes information on one partitioning key.)

If HiRDB Advanced High Availability is not installed, this table is empty. However, if HiRDB Advanced High Availability is installed and a database is created, and then HiRDB Advanced High Availability is removed afterwards, the data in the table remains.

The following table shows the contents of the SQL_PARTKEY table.

Table G-41: SQL_PARTKEY table contents

Number	Column name	Data type	Contents

1	TABLE_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Table owner

2	TABLE_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Table name

3	KEY_NO	SMALLINT	Partitioning key number (dimension number 1 or 2)

4	KEY_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Partitioning key column name

5	COLUMN_ID	SMALLINT	Partitioning key column ID
6	N_DIVISION	SMALLINT	Number of divisions inside the key
7	HASH_KEY_NO	SMALLINT	Sequence number in hash key column Null value for dimensions of boundary value partitioning.

(40) SQL_PARTKEY_DIVISION table contents

This table manages the information on the partitioning condition values for a matrix-partitioned table. (Each row describes information on one partitioning condition value.)

If HiRDB Advanced High Availability is not installed, this table is empty. However, if HiRDB Advanced High Availability is installed and a database is created, and then HiRDB Advanced High Availability is removed afterwards, the data in the table remains.

The following table shows the contents of the `SQL_PARTKEY_DIVISION` table.

Table G-42: SQL_PARTKEY_DIVISION table contents

Number	Column name	Data type	Contents

1	`TABLE_SCHEMA`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Table owner

2	`TABLE_NAME`	`VARCHAR(30)`
		or
		`MVARCHAR(30)`
			Table name

3	`KEY_NO`	`SMALLINT`	Partitioning key number (dimension number 1 or 2)
4	`IN_DIM_NO`	`SMALLINT`	Serial number inside a partitioning key

5	`DCVALUES`	`VARCHAR(255)`
		or
		`MVARCHAR(255`
		`)`
			Partitioning condition value (the specified partitioning condition value is stored in the character format).
			The value to be stored is not converted even if a character set is specified for a partitioning key.
			Null value for the last boundary value within a partitioning key and for dimensions of hash partitioning.

(41) SQL_AUDITS table contents

This table manages audit target information. (Each row describes information on one event for one object or user.)

The following table shows the contents of the `SQL_AUDITS` table.

Table G-43: SQL_AUDITS table contents

Number	Column name	Data type	Contents
1	`EVENT_TYPE`	`VARCHAR(30)`	Name of the event type#1 specified by the `CREATE AUDIT FOR` operation type or `'ANY'`.
2	`EVENT_SUBTYPE`	`VARCHAR(30)`	Event sub-type name#2 or `'ANY'` Null value if `CREATE AUDIT FOR ANY` is specified.

Number	Column name	Data type	Contents
3	OBJECT_TYPE	VARCHAR(30)	Type of object specified by the CREATE AUDIT selection option.[#3] Null value if no object is specified or if the HiRDB version is earlier than 07-03.

4	OBJECT_SCHEMA	VARCHAR(30)
		or
		MVARCHAR(30)
			Owner of object specified by the CREATE AUDIT selection option.
			Null value if no object is specified or if the HiRDB version is earlier than 07-03.

5	OBJECT_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Name of object specified by the CREATE AUDIT selection option.
			Null value if no object is specified or if the HiRDB version is earlier than 07-03.

6	USER_NAME	VARCHAR(30)
		or
		MVARCHAR(30)
			Authorization identifier of event executor (null value).

7	ANY_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER ANY is specified: Y: Specified N: Not specified
8	SUCCESSFUL_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER SUCCESSFUL is specified: Y: Specified N: Not specified
9	UNSUCCESSFUL_ANY_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER UNSUCCESSFUL is specified: Y: Specified N: Not specified
10	AUDIT_TYPE	CHAR(1)	Acquisition information type: E: CREATE AUDIT AUDITTYPE EVENT is specified A: CREATE AUDIT AUDITTYPE ANY is specified Null value: CREATE AUDIT AUDITTYPE PRIVILEGE is specified or AUDITTYPE is omitted.

#1: The following event types are available:

SESSION, PRIVILEGE, DEFINITION, ACCESS, and UTILITY

#2: The following event sub-types are available:

CONNECT, AUTHORIZATION, DISCONNECT, GRANT, REVOKE, CREATE, DROP, ALTER, SELECT, INSERT, UPDATE, DELETE, PURGE, CALL, OPEN, LOCK, PDLOAD, PDRORG, and PDEXP

#3: The following object types are available:

FUNCTION, INDEX, PROCEDURE, SCHEMA, SERVER, TABLE, TRIGGER, DATA TYPE, VIEW, LIST, COMMENT, and SEQUENCE

(42) SQL_REFERENTIAL_CONSTRAINTS table contents

This table manages the corresponding conditions of referential constraints. (Each row describes information on one constraint.)

The following table shows the contents of the SQL_REFERENTIAL_CONSTRAINTS table.

Table G-44: SQL_REFERENTIAL_CONSTRAINTS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
2	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Constraint owner
3	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint is defined
4	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint is defined
5	COLUMN_COUNT	SMALLINT	Number of columns in the foreign key
6	COLUMN_NAME	VARCHAR(527) or MVARCHAR(527)	Column names of the table containing the foreign key Enclose each column in quotation marks and link the columns with commas.
7	COLUMN_NO	VARCHAR(32)	Column IDs (16 IDs) of the table containing the foreign key#
8	R_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the table to be referenced
9	R_TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table to be referenced
10	DELETE_RULE	CHAR(11)	Deletion rule (RESTRICT or CASCADE)

Number	Column name	Data type	Contents
11	UPDATE_RULE	CHAR(11)	Update rule (RESTRICT or CASCADE)
12	CONSTRAINT_TIME	CHAR(14)	Date and time when the constraint was defined (*YYYYMMDDHHMMSS*)
13	CHECK_PEND	CHAR(1)	Type of check pending status C: Pending Null value: Non-pending
14	DELETE_TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the trigger created by the action of the ON DELETE referential constraint (DRA*YYYYMMDDHHMMSS*th) Null value if no trigger is created by the action of the ON DELETE referential constraint.
15	UPDATE_TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the trigger created by the action of the ON UPDATE referential constraint (DRA*YYYYMMDDHHMMSS*th) Null value if no trigger is created by the action of the ON UPDATE referential constraint.
16	R_COLUMN_NAME	VARCHAR(527) or MVARCHAR(527)	Column names of the columns that make up the main key Enclose each column in quotation marks and link the columns with commas.
17	R_COLUMN_NO	VARCHAR(32)	Column IDs (16 IDs) of the columns that make up the main key#

#: Endian conversion is not performed on the SQL results even if the connection modes have different endians. Therefore, when an application accesses the SQL results, the SQL must consider the endian and convert the endian if necessary.

(43) SQL_KEYCOLUMN_USAGE table contents

This table manages information on the columns that make up foreign keys. (Each row describes information on one column.)

The following table shows the contents of the SQL_KEYCOLUMN_USAGE table.

Table G-45: SQL_KEYCOLUMN_USAGE table contents

Number	Column name	Data type	Contents
1	`CONSTRAINT_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Constraint owner
2	`CONSTRAINT_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Constraint name
3	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of the table for which the constraint was defined
4	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the table for which the constraint was defined
5	`COLUMN_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the column for which the constraint was defined
6	`COLUMN_ORDER`	`SMALLINT`	Position of the column for which the constraint was defined

(44) SQL_TABLE_CONSTRAINTS table contents

This table manages information on integrity constraints found in a schemas. (Each row describes information on one integrity constraint.)

The following table shows the contents of the `SQL_TABLE_CONSTRAINTS` table.

Table G-46: SQL_TABLE_CONSTRAINTS table contents

Number	Column name	Data type	Contents
1	`CONSTRAINT_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Constraint owner
2	`CONSTRAINT_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Constraint name
3	`CONSTRAINT_TYPE`	`VARCHAR(30)`	Constraint type `FOREIGN KEY`: Foreign key `CHECK`: Check constraint
4	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of the table for which the constraint was defined
5	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the table for which the constraint was defined

(45) SQL_CHECKS table contents

This table manages information on check constraints. (Each row describes information on one check constraint.)

The following table shows the contents of the SQL_CHECKS table.

Table G-47: SQL_CHECKS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Check constraint owner
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined
5	CHK_SOURCE_LEN	INTEGER	Length of the check constraint search conditions
6	CHK_SOURCE	BINARY(2000000)	Check constraint search conditions
7	CREATE_TIME	CHAR(14)	Date and time when the search constraint was defined (*YYYYMMDDHHMMSS*)
8	CHECK_PEND2	CHAR(1)	Check pending status type C: Pending Null value: Non-pending
9	N_CHK_COLUMN	INTEGER	Number of constraint columns specified in the check constraint definition (number of duplicate exclusion columns)

(46) SQL_CHECK_COLUMNS table contents

This table manages information on the columns used by check constraints. (Each row describes information on one column used by one check constraint.)

The following table shows the contents of the SQL_CHECK_COLUMNS table.

Table G-48: SQL_CHECK_COLUMNS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Check constraint owner
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name

Number	Column name	Data type	Contents
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the column used by the constraint

(47) SQL_DIV_TYPE table contents

This table manages information on partitioning keys in matrix partitioning tables that combine key range partitioning and hash partitioning. (Each row describes information on one partitioning key.)

The following table shows the contents of the SQL_DIV_TYPE table.

Table G-49: SQL_DIV_TYPE table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	KEY_NO	SMALLINT	Partitioning key number (dimension number)
4	DIV_TYPE	CHAR(1)	Partitioning type in the dimension P: Boundary value partitioning F: FIX hash partitioning H: Flexible hash partitioning
5	HASH_NAME	VARCHAR(30) or MVARCHAR(30)	Hash function name "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" Null value for dimensions without hash partitioning
6	N_DIV_COLUMN	SMALLINT	Number of partitioning columns in the dimension

(48) SQL_SYSPARAMS table contents

This table manages information about limits on the number of consecutive certification failures and password character strings. (Each row describes information on one setting item. *n* rows describes information on one limit on the number of consecutive certification failures or one password character string limit.) The SQL_SYSPARAMS table can be referenced only by owners with the DBA privilege and auditors.

The following table shows the contents of the SQL_SYSPARAMS table.

Table G-50: SQL_SYSPARAMS table contents

Number	Column name	Data type	Contents
1	PARAM_KIND	VARCHAR(20)	Parameter type (CONNECTION_SECURITY)
2	FUNCTION_KEY	VARCHAR(20)	Function name CONNECT: Limit on the number of consecutive certification failures PASSWORD: Password character sting limit
3	PARAM_KEY	VARCHAR(20)	Specification item When the function name is CONNECT, the specification item is one of the following: PERMISSION_COUNT: Permitted number of consecutive certification failures LOCK_MINUTE: Account lock period (minutes) LOCK_MINUTE_CODE: Account lock period code When the function name is PASSWORD, the specification item is one of the following: MIN_LENGTH: Minimum number of allowed bytes USER_IDENTIFIER: Specification of authorization identifier prohibited SIMILAR: Specification of single character type prohibited
4	INT_VALUE	INTEGER	INT-type data value#
5	CHAR_VALUE	VARCHAR(30)	CHAR-type data value#

#: The table below shows the values that are stored for the INT-type and CHAR-type data values.

PARAM_KEY setting value	Value specified in SQL	INT_VALUE	CHAR_VALUE
PERMISSION_COUNT	Constant	Constant	Constant
	No specification	2	2
LOCK_MINUTE	Constant	Constant	Constant
	UNLIMITED	Null value	UNLIMITED
	No specification	1440	1440
LOCK_MINUTE_CODE	Constant	Constant	Constant
	UNLIMITED	Null value	UNLIMITED
	No specification	1000000	1000000
MIN_LENGTH	Constant	Constant	Constant
	No specification	8	8
USER_IDENTIFIER	RESTRICT	Null value	RESTRICT
	UNRESTRICT	Null value	UNRESTRICT
	No specification	Null value	RESTRICT
SIMILAR	RESTRICT	Null value	RESTRICT
	UNRESTRICT	Null value	UNRESTRICT
	No specification	Null value	RESTRICT

(49) SQL_INDEX_XMLINF table contents

This table manages information about the component substructure paths of substructure indexes (each row describes information on one index).

The following table shows the contents of the SQL_INDEX_XMLINF table.

Table G-51: SQL_INDEX_XMLINF table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name

Number	Column name	Data type	Contents
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	PARTIAL_STRUCTURE_PATH_ORDER	SMALLINT	Always 1
5	PARTIAL_STRUCTURE_PATH	VARCHAR(32000) or MVARCHAR(32000)	Substructure path
6	ASC_DESC	CHAR(1)	Sort order: A: Ascending order D: Descending order
7	DATA_TYPE	CHAR(24)	Data type of the substructure path For details about the storage format, see the DATA_TYPE column of the SQL_COLUMNS table.
8	DATA_TYPE_CODE	SMALLINT	Data type code of the substructure path For details about the storage format, see the DATA_TYPE_CODE column of the SQL_COLUMNS table.
9	DATA_LENGTH	CHAR(7)	Data length of the substructure path (character format) For details about the storage format, see the DATA_LENGTH column of the SQL_COLUMNS table.
10	DATA_LENGTH_CODE	SMALLINT	Data length of the substructure path. For details about the storage format, see the DATA_LENGTH_CODE column of the SQL_COLUMNS table.

(50) SQL_SEQUENCES table contents

This table manages information about sequence generators. (Each row describes information on one sequence generator.)

The following table shows the contents of the SQL_SEQUENCES table.

Table G-52: SQL_SEQUENCES table contents

Number	Column name	Data type	Contents
1	SEQUENCE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the sequence generator

Numb er	Column name	Data type	Contents
2	SEQUENCE_NAME	VARCHAR(30) or MVARCHAR(30)	Sequence generator identifier
3	SEQUENCE_ID	INT	Sequence generator ID
4	SEQUENCE_TYPE	CHAR(1)	Information used by the system Always E.
5	PUBLIC_USAGE	CHAR(1)	Whether PUBLIC USAGE is specified: Y: Specified Null value: PUBLIC USAGE is not specified.
6	CREATE_TIME	CHAR(14)	Sequence generator creation time
7	ALTER_TIME	CHAR(14)	Always the null value
8	DATA_TYPE	CHAR(24)	Data type of the sequence generator: INTEGER: Default value For details about the storage format, see the DATA_TYPE column of the SQL_COLUMNS table.
9	DATA_TYPE_CODE	SMALLINT	Data type code of the sequence generator For details about the storage format, see the DATA_TYPE_CODE column of the SQL_COLUMNS table.
10	DATA_LENGTH	CHAR(7)	Data length of the sequence generator (character format) For details about the storage format, see the DATA_LENGTH column of the SQL_COLUMNS table.
11	DATA_LENGTH_CODE	SMALLINT	Data length of the sequence generator For details about the storage format, see the DATA_LENGTH_CODE column of the SQL_COLUMNS table.
12	START_VALUE	VARCHAR(255)	Start value Null value if this information is not specified.

Number	Column name	Data type	Contents
13	MAXIMUM_VALUE	VARCHAR(255)	Maximum value: NO MAXVALUE: NO MAXVALUE is specified Null value if this information is not specified.
14	MINIMUM_VALUE	VARCHAR(255)	Minimum value: NO MINVALUE: NO MINVALUE is specified Null value if this information is not specified.
15	INCREMENT	VARCHAR(255)	Increment Null value if this information is not specified.
16	CYCLE_OPTION	CHAR(1)	Cycle option: Y: CYCLE N: NO CYCLE is specified or the specification is omitted.
17	LOGINTERVAL	INT	Sequence generator log output interval If this information is not specified, 1 is set. The initial value is 1.
18	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the sequence generator storage RDAREA
19	RDAREA_ID	INT	ID of the sequence generator storage RDAREA

H. Functions provided by HiRDB

This appendix explains the following functions provided by HiRDB:

- Hash function for table partitioning
- Space conversion facility
- Function for conversion to a DECIMAL signed normalized number
- Function that sets the character code classification

Note that the Linux for AP8000 edition client cannot use the functions provided by HiRDB.

H.1 Hash function for table partitioning

The hash function for table partitioning uses the partitioning key values to obtain the order of partitioning conditions that are specified for partitioning a table. If a UAP is executed using the hash function for table partitioning, the storage RDAREAs can be identified before data is stored in a table, even if the table is a hash-partitioned table. Because this function can identify each storage RDAREA, you can use the function for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when determining the hash function and partitioning key for hash partitioning
- To create an input data file for each RDAREA when loading data to a hash-partitioned table in units of RDAREAs concurrently using the database load utility

(1) Prerequisites for using the hash function for table partitioning

The following describes the prerequisites for using the hash function for table partitioning.

(a) Program language

When the hash function for table partitioning is used to create a UAP, the UAP can be written in either C or C++.

(b) Execution environment

The hash function for table partitioning can be executed on a server machine in which a HiRDB server or HiRDB client has been installed.

However, certain combinations of a HiRDB server operating system and a HiRDB client operating system can produce incorrect results when the function is executed with a HiRDB client.

The following table shows the UAP execution conditions in the HiRDB client.

Table H-1: Execution conditions in the HiRDB client

HiRDB server operating system	HiRDB client operating system	
	HP-UX, Solaris, and AIX	Linux and Windows
HP-UX, Solaris, and AIX	E	--
Linux and Windows	--	E

E: Can be executed.

--: Errors occur in the partitioning condition specification order or the partitioning key sequence numbers because the operating systems use a different byte order.

(2) Creating and executing UAPs that use the hash function for table partitioning

Create and execute a UAP according to the following procedure:

1. Create a source program.
2. Compile and link the source program.
3. Execute the load module.

(a) Creating a source program

Specify function calling of the hash function for table partitioning in the source program written in C or C++. Because the hash function for table partitioning is presented in a shared library format, link the source program to use the function.

When the hash function for table partitioning is used, the distributed header files must be included when the source program is created. Include all header files required by the hash function for table partitioning. For details about the header files required by the hash function for table partitioning, see *(3) Function details*.

(b) Compiling and linking the source program

Compile and link the source program in a server machine that has either the HiRDB server or HiRDB client installed.

If SQL statements are embedded in the source program, preprocessing must be executed before compiling and linking.

For details about compiling, linking, and preprocessing, see *8. Preparation for UAP Execution*.

Compiling and linking in a UNIX environment (HiRDB server)

Specification examples for compiling and linking a source program in the HiRDB server are shown as follows:

Example (C)

When the source filename is `sample.c` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
cc -I $PDDIR/include sample.c -L$PDDIR/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
cc +DD64 -I $PDDIR/include sample.c -L$PDDIR/client/lib
-l sqlauxf64
```

Example (C++)

When the source filename is `sample.C` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
CC -I $PDDIR/include sample.C -L$PDDIR/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
CC +DD64 -I $PDDIR/include sample.C -L$PDDIR/client/lib
-l sqlauxf64
```

Compiling and linking in a UNIX environment (HiRDB client)

Shown below are specification examples for compiling and linking a source program in the HiRDB client.

Example (C)

When the source filename is `sample.c` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
cc -I /HiRDB/include sample.c -L/HiRDB/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
cc +DD64 -I /HiRDB/include sample.c -L/HiRDB/client/lib
-l sqlauxf64
```

Note

The underline indicates the HiRDB client's installation directory.

Example (C++)

When the source filename is `sample.C` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
CC -I /HiRDB/include sample.C -L/HiRDB/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
CC +DD64 -I /HiRDB/include sample.C -L/HiRDB/client/lib
-l sqlauxf64
```

Note

The underline indicates the HiRDB client's installation directory.

Compiling and linking in a Windows environment (HiRDB server)

For a source program written in C, use an ANSI-C-compliant compiler to compile the program. For a source program written in C++, use a C++-compliant compiler to compile the program.

If you are using Microsoft Visual C++ Version 1.0 to compile and link the source program, select **Set Project** from the **Options** menu for the option settings.

If you are using Microsoft Visual C++ Version 2.0 to compile and link the source program, select **Set** from the **Project** menu for the option settings.

The following table shows the items to be set in the HiRDB server with **Set Project** or **Set**.

Table H-2: Items to be set in the HiRDB server with Set Project or Set

Item	Category	Category setting	Setting value
Compiler	Code generation	Structure member alignment	8 bytes
		Run time library to be used	Multi-thread
	Processor	Include file path	`\`*HiRDB*`\client\include`
Linker	Input	Library	`\`*HiRDB*`\client\lib\pdsqlauxf.lib`

Note

The underline indicates the HiRDB client's installation directory.

Compiling and linking in a Windows environment (HiRDB client)

For a source program written in C, use an ANSI-C-compliant compiler to compile the program. For a source program written in C++, use a C++-compliant compiler to compile the program.

If you are using Microsoft Visual C++ Version 1.0 to compile and link the source program, select **Set Project** from the **Options** menu for the option settings.

If you are using Microsoft Visual C++ Version 2.0 to compile and link the source program, select **Set** from the **Project** menu for the option settings.

The following table shows the items in the HiRDB client to be set with **Set Project** or **Set**.

Table H-3: Items to be set in the HiRDB client with Set Project or Set

Item	Category	Category setting	Setting value
Compiler	Code generation	Structure member alignment	8 bytes
		Run time library to be used	Multi-thread
	Processor	Include file path	\HiRDB`\include`
Linker	Input	Library	\HiRDB`\lib\pdsqlauxf.lib`

Note

The underline indicates the HiRDB client's installation directory.

(3) Function details

(a) Required input information

To call the hash function for table partitioning, obtain the information for items 1 through 8, described as follows, and set the information to arguments.

1. Hash function name specified for partitioning
2. Number of columns specified in partitioning keys
3. Specification order of partitioning keys, data type codes, and data length codes
4. Number of table partitions
5. Data values stored in partitioning keys
6. Double-byte space character for each national character code type used in the HiRDB server.
7. Space conversion level
8. Whether or not to use the facility for conversion to a DECIMAL signed normalized number

Items 1 through 4 correspond to the following sections in the `CREATE TABLE` statement:

```
CREATE TABLE TABLE1 ( C1  CHAR(10) NOT NULL,
                                  3
                      C2  NVARCHAR(4) NOT NULL,
                                  3
                      C3  DEC(5,2) NOT NULL,
                                  3
                     C4  INT,

                      C5  SMALLINT NOT NULL )
                                  3
        FIX HASH  HASH6  BY  C1,C3,C5,C2
                    1          2,3
            in ( RU01,RU02,RU03,RU04,RU05,RU06 )
                              4
```

If the table is already defined, information for items 1 through 4 can be obtained by retrieving the dictionary table. For examples of dictionary table retrieval, see *(6) Retrieval from dictionary tables (for hash partitioning)*.

For details about the space conversion level (space conversion facility) and the facility for conversion to a DECIMAL signed normalized number, see the *HiRDB Version 9 System Operation Guide*.

(b) Specification configuration

Details about the hash function for table partitioning are explained as follows:

Description

Provides an overview of the function.

Header files

Explains the headers that are necessary for using the hash function for table partitioning.

Format

Explains the actual specification format.

Arguments

Explains the arguments specified in the format.

Return value

Explains the return value types (specified as data types) of the hash function for table partitioning.

(c) p_rdb_dbhash hash function for table partitioning

Description

This function obtains the partitioning condition specification order (1 to number of table partitions) in which the partitioning keys are stored, or the partitioning key sequence numbers. If the function does not terminate normally, an incorrect value is obtained for the partitioning condition specification order.

If the partitioning condition specification order is obtained from multiple rows, the partitioning key data must be changed for each of those rows before the hash function for table partitioning is called. In this case, only those arguments that contain data values for partitioning keys must be changed; all other arguments do not need to be changed.

For details about how to determine the partitioning condition specification order from the partitioning key sequence numbers, see *(7) Retrieval from dictionary tables (for matrix partitioning)*.

Header files

`#include<pddbhash.h>`

This header file must be specified when the hash function for table partitioning is used.

`#include<pdbsqlda.h>`

This header file should be specified when a macro that begins with `PDSQL_` is used to set data type codes for the partitioning keys. This header file can be omitted when the data type codes to be set are retrieved from the dictionary table.

Format 1 (when no character is set specified for partitioning keys)

```
int p_rdb_dbhash(short           hashcode,
                 short           ncol,
                 p_rdb_collst_t  *collst,
                 p_rdb_dadlst_t  *dadlst,
                 unsigned int    ndiv,
                 unsigned char   ncspace[2],
                 int             flags,
                 int             *rdno);
```

Format 2 (When character sets are specified for partitioning keys)

```
int p_rdb_dbhash_cs(short           hashcode,
                    short           ncol,
                    p_rdb_collst_t  *collst,
                    p_rdb_csidlst_t *csidlst,
                    p_rdb_dadlst_t  *dadlst,
                    unsigned int    ndiv,
                    unsigned char   ncspace[2],
                    int             flags,
                    int               *rdno);
```

Arguments

`hashcode` (input)

Specifies a hash function code that corresponds to a hash function name. For details on hash function codes, see *(4)(a) Hash function codes*.

`ncol` (input)

Specifies the number of columns that were specified as partitioning keys when the table was defined.

`collst` (input)

Specifies a pointer to a partitioning key list. A partitioning key list is a structure that consists of the data type code and data size code of a partitioning key, and is an area in which partitioning keys are continuously listed. For details on the partitioning key list, see *(4)(b) Partitioning key list*.

You can obtain the data type code and data size code of a partitioning key by retrieving them from a dictionary table. For examples of dictionary table retrieval, see *(6) Retrieval from dictionary tables (for hash partitioning)*.

`csidlst` (input)

Specifies a pointer to the character set ID list for partitioning keys. This argument is specified only in format 2 (when character sets are specified for partitioning keys).

The character set ID list is an area containing as many consecutive structures composed of character set IDs as there are partitioning keys. For details about the character set ID list, see *(4)(c) Character set ID list*.

You can obtain the character set IDs of the partitioning keys by searching data dictionary tables. For details about searching data dictionary tables, see *(6) Retrieval from dictionary tables (for hash partitioning)*.

`dadlst` (input)

Specifies the pointer to the data address list. The data address list is a structure composed of the addresses to the data storage areas for partitioning keys and is allocated as a contiguous area for all partitioning keys. For details, see *(4)(d) Data address list*.

`ndiv` (input)

Specifies the number of partitions in hash partitioning.

`ncspace` (input)

Specifies the double-byte space character for each national character code type used in the HiRDB server. The character is specified in a two-byte area. When the data type of a partitioning key is NVARCHAR, this argument is used to remove spaces that follow character strings before hashing is executed. This argument is also used for space conversion for the partitioning key value (NCHAR, NVARCHAR, MCHAR, or MVARCHAR) when space conversion level 1 or 3 is specified in the flags argument.

An error results if the area specified in the ncspace argument contains no space character in the following cases:

- Partitioning key is NVARCHAR.
- Space conversion level 1 or 3 is specified in flags and the partitioning key is NCHAR, NVARCHAR, MCHAR, or MVARCHAR.

The following table lists the double-byte space characters that can be specified in ncspace.

Table H-4: Double-byte space characters specified in ncspace

Character code type specified in pdsetup#2	**ncspace**	
	ncspace[0]	**ncspace[1]**
sjis (shift JIS kanji code)	0x81	0x40
Chinese (EUC Chinese kanji code)	0xA1	0xA1
ujis (EUC Japanese kanji code)	0xA1	0xA1
lang-c (single-byte character code)#1	0x00	0x00
Unicode (UTF-8)#3	0x00	0x00
chinese-gb18030 (Chinese kanji code GB18030)#3	0xA1	0xA1
Default value (sjis for HP-UX)	0x81	0x40
Default value (sjis for AIX)	0x81	0x40
Default value (ujis for Solaris)	0xA1	0xA1
Default value (ujis for Linux)	0xA1	0xA1
Default value (sjis for Windows)	0x81	0x40

#1: If the character code type is lang-c, NCHAR, NVARCHAR, MCHAR, or MVARCHAR cannot be used for the column data type.

#2: For a Windows environment, specify the space character code of the character code type that was specified in the pdntenv command.

#3: NCHAR or NVARCHAR cannot be used for the column data type.

flags (input)

Specifies the flag according to the space conversion level and the facility for conversion to a DECIMAL signed normalized number (this argument is required even if these facilities are not used). For details about the space conversion level (space conversion facility) and the facility for conversion to a DECIMAL signed normalized number, see the *HiRDB Version 9 System Operation Guide*.

The following table shows the values for flags:

HiRDB operating environment		Value of flags
Space conversion level#	Omitted	p_rdb_FLG_SPLVL_0
	0	
	1	p_rdb_FLG_SPLVL_1
	3	p_rdb_FLG_SPLVL_3
Facility for conversion to a DECIMAL signed normalized number	Omitted	p_rdb_FLG_DECNRM_N
	N	
	Y	p_rdb_FLG_DECNRM_Y

#: If the character code for the HiRDB server is Unicode (UTF-8), spaces must be converted before this function is executed. Therefore, specify only p_rdb_FLG_SPLVL_0 for the flags value.

rdno (output)

Sets the partitioning condition specification order (1 to number of table partitions) or the partitioning key sequence numbers.

Return values

data type: int

p_rdb_RC_RTRN(0)

Normal termination.

p_rdb_RC_ERRHASH(-1)

Invalid hash function code (p_rdb_HASH1 to p_rdb_HASH6, p_rdb_HASH0, p_rdb_HASHA to p_rdb_HASHF).

p_rdb_RC_ERRNCOL(-2)

Partitioning key count error (1 to p_rdb_MXDCL).

`p_rdb_RC_ERRCLST(-3)`

Area error for partitioning key data type or data length.

`p_rdb_RC_ERRCTYP(-31)`

Invalid data type for partitioning key.

`p_rdb_RC_ERRCLEN(-32)`

Invalid data type for partitioning key.

`p_rdb_RC_ERRCSID(-33)`

Invalid character set ID for partitioning key

`p_rdb_RC_ERRDLST(-4)`

Area error for data address.

`p_rdb_RC_ERRDADR(-41)`

Data address not set.

`p_rdb_RC_ERRDLEN(-42)`

Actual data length is shorter than character length limit for hash function.

`p_rdb_RC_ERRNDIV(-5)`

Table partition count error (`1` to `p_rdb_MNCND`)

`p_rdb_RC_ERRRADR(-6)`

Storage area for partitioning condition specification order or partitioning key sequence numbers is not set.

`p_rdb_RC_ERRNCSC(-7)`

Area for double-byte space character is not set.

Notes

1. If the partitioning key is `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`, the value of `rdno` may be invalid unless an appropriate value corresponding to the space conversion level is specified in `flags`.
2. If the partitioning key is `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`, and `1` or `3` is specified for the space conversion level, perform one of the following:

   - Use the `setlocale` function to set an appropriate locale for the `LC_CTYPE` category or the `LC_ALL` category.
   - Call the `p_rdb_set_lang` function.

   Operation is not guaranteed if the character code type of the key value and the locale specified by the `setlocale` function or `p_rdb_set_lang` function contradict each other. If this function is called from a Windows UAP, Linux UAP with SJIS character codes type, or UAP with CHINESE character codes type, the p_rdb_set_lang function is used instead of the setlocale function. For details about the p_rdb_set_lang function, see *H.4 Character code type specification function*.

3. If the partitioning key value is DECIMAL, INTERVAL YEAR TO DAY, or INTERVAL HOUR TO SECOND, the value of rdno may be invalid unless an appropriate value is specified in flags which corresponds to the facility for conversion to a DECIMAL signed normalized number.
4. When using the hash function for table partitioning with a HiRDB client version earlier than 05-05, you cannot use the space conversion level in flags or the facility for conversion to a DECIMAL signed normalized number. In this case, the function ignores flags and assumes that the facility for conversion to a DECIMAL signed normalized number is omitted. To use the flags specification, either execute the function at the HiRDB server or at a HiRDB client with version 05-05 or later.
5. If the character code for the HiRDB server is Unicode (UTF-8), this function does not convert spaces. The partitioning key value to be specified for dadlst must be converted beforehand using the space conversion function p_rdb_conv_space_utf8.
6. If UTF16 is specified as the character set, data for the partitioning key must be in big endian format. If the data is not in big endian format, the rdno value might become invalid.

(4) Data types and macros

(a) Hash function codes

The following table shows the hash function codes that correspond to the hash functions specified in CREATE TABLE or ALTER TABLE.

Table H-5: Hash function codes for hash functions

Hash function name	Hash function code (value)
HASH1(when hash function name is omitted)	p_rdb_HASH1(1)
HASH2	p_rdb_HASH2(2)
HASH3	p_rdb_HASH3(3)
HASH4	p_rdb_HASH4(4)
HASH5	p_rdb_HASH5(5)
HASH6	p_rdb_HASH6(6)

Hash function name	Hash function code (value)
`HASH0`	`p_rdb_HASH0(100)`
`HASHA`	`p_rdb_HASHA(101)`
`HASHB`	`p_rdb_HASHB(102)`
`HASHC`	`p_rdb_HASHC(103)`
`HASHD`	`p_rdb_HASHD(104)`
`HASHE`	`p_rdb_HASHE(105)`
`HASHF`	`p_rdb_HASHF(106)`

(b) Partitioning key list

The partitioning key list is a structure composed of data type codes and data length codes for partitioning keys, and is allocated a contiguous area for all partitioning keys. *Table H-6* shows the area for setting partitioning keys. If there are multiple partitioning keys, the area must be specified as an array consisting of all columns specified as partitioning keys.

Table H-7 lists the data type codes and the data length codes.

Table H-6: Area for setting partitioning keys

Data type	Data type details	Explanation

`p_rdb_collst_t`	`struct p_rdb_TG_collst {`
	`    unsigned short datatype ;`
	`    short datalen ;`
	`}  p_rdb_collst_t ;`
		
		Data type code
		Data length code

Table H-7: Data type codes and data length codes

Data type	Data type code	Data length code
`INTERVAL YEAR TO DAY`	`PDSQL_YEARTODAY`	8 x 256
`INTERVAL HOUR TO SECOND`	`PDSQL_HOURTOSEC`	6 x 256
`DATE`	`PDSQL_DATE`	4
`TIME`	`PDSQL_TIME`	3
`TIMESTAMP[(`*p*`)]`	`PDSQL_TIMESTAMP`	7 + ↑*p*/2↑ (`0` is assumed if *p* is omitted.)
`MVARCHAR(`*n*`)`	`PDSQL_MVARCHAR`	*n*

Data type	Data type code	Data length code
`MCHAR[(`*n*`)]`	`PDSQL_MCHAR`	*n* (default value is 1)
`NVARCHAR(`*n*`)`	`PDSQL_NVARCHAR`	*n*
`NCHAR[(`*n*`)]`	`PDSQL_NCHAR`	*n* (default value is 1)
`VARCHAR(`*n*`)`	`PDSQL_VARCHAR`	*n*
`CHAR[(`*n*`)]`	`PDSQL_CHAR`	*n* (default value is 1)
`FLOAT`	`PDSQL_FLOAT`	8
`SMALLFLT`	`PDSQL_SMALLFLT`	4
`DECIMAL[(`*p*`[,`*q*`])]`	`PDSQL_DECIMAL`	*p* x 256 + *q* (default values are 15 for *p* and 0 for *q*)
`INTEGER`	`PDSQL_INTEGER`	4
`SMALLINT`	`PDSQL_SMALLINT`	2

(c) Character set ID list

A character set ID list is specified only when a character set is specified for a partitioning key. *Table H-8* shows the area in which a character set ID is set. If there are multiple partitioning keys, the character set ID list must be an array containing as many elements as there are columns specified as partitioning keys. *Table H-9* lists the character set ID codes.

Table H-8: Area in which a character set ID is set

Data type	Details of data type	Description

p_rdb_csidlst_t	`struct p_rdb_TG_csidlst {`
	`    int` **`charset_id`** `;`
	`} p_rdb_csidlst_t ;`
		Character set ID code

Table H-9: Character set ID codes

Character set	Character set ID code (value)
No character set specified	p_rdb_CSDEF (0x00000000)
EBCDIK	p_rdb_CSEBK (0x01000004)
UTF16	p_rdb_CSU16 (0x11030004)

(d) Data address list

The data address list is a structure composed of the addresses to the data storage areas for partitioning keys, and is allocated as a contiguous area for all partitioning keys. The table below shows the area for setting the data address of a partitioning key. If there are multiple partitioning keys, the area must be specified as an array consisting of all columns specified as partitioning keys.

Specify the area in binary format. For details about the binary format, see the *HiRDB Version 9 Command Reference* manual.

Table H-10: Area for setting the data address of a partitioning key

Data type	Data type details	Explanation

`p_rdb_dadlst_t`	`struct p_rdb_TG_dadlst {`
	`    unsigned char * dataaddr ;`
	`}  p_rdb_dadlst_t ;`
		Address to data area

Notes common to all data types

- Convert the real data in the data address list to the data type format defined in the column.
- The boundaries of the real data area for the data address list do not have to be adjusted.

Notes about data types DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND

- For positive values, use `C` or `F` in the sign section of the real data in the data address list. If `Y` is specified for the facility for conversion to a DECIMAL signed normalized number, `A` and `E` are also available.
- For negative values, use `D` in the sign section of the real data in the data address list. If `Y` is specified for the facility for conversion to a DECIMAL signed normalized number, `B` is also available.

Notes about data types CHAR, NCHAR, and MCHAR

- For `CHAR` and `MCHAR`, pad the data area of the data address list with single-byte space characters up to the defined length.
- For `NCHAR`, pad the data area of the data address list with double-byte space characters up to the defined length. The double-byte space characters must be of the character code that was specified at HiRDB server setup.
- The data area of the data address list must be specified with the character codes used by the HiRDB server.

Notes about data types VARCHAR, NVARCHAR, and MVARCHAR

- For the real length section in the data area of the data address list, use bytes instead of character string length to indicate the data length.
- If the real length of the data area for the data address list is less than the defined length of the partitioning key list, do not pad the character string that follows.
- Specify character codes used by the HiRDB server in the data area of the data address list.

(e) Macros for maximum values

The following table lists the macros for maximum values.

Table H-11: Macros for maximum values

Macro name	Description (value)
p_rdb_MXDCL	Maximum number of partitioning key columns (16)
p_rdb_MNCND	Maximum number of table partitions (1,024)

(5) Coding examples

A partial coding example that uses C to describe hash partitioning is shown below. Use this coding example by customizing it to suit the user needs. However, because this example does not include error handling during SQL statement execution, code error handling as needed. For details about error handling, see *3.6 SQL error identification and corrective measures*.

(a) Declaration section

```
/
**************************************************************
*/
/* ALL RIGHTS RESERVED. COPYRIGHT (C) 1999,2000, HITACH, LTD.  */
/* LICENSED MATERIAL OF HITACHI,LTD.                           */
/* Sample Program that Uses the Hash Function for Table
Partitioning                                                   */
/
**************************************************************
*/
#include <stdio.h>
#include <string.h>
#include <pdbsqlda.h>
#include <pddbhash.h>

union data_area {                     /* Data storage area */
    short data_smallint ;
    int data_int ;
    unsigned char data_dec[15] ;
    float data_smallflt ;
    double data_float ;
    unsigned char data_char[255] ;
    struct {
        short length ;
        unsigned char data[255] ;
    } data_varchar ;
    unsigned char data_date[4] ;
    unsigned char data_time[3] ;
    unsigned char data_timestamp[10] ;
    unsigned char data_iytd[5] ;
    unsigned char data_ihts[4] ;
} ;

void print_data(short , p_rdb_collst_t * , union data_area *) ;


/
***************************************************************
*/
/* Main Function                                              */
/
***************************************************************
*/
int main(int argc , char *argv[])
{

  short           hashcode ;        /* Hash function code */
  short          ncol ;            /* Number of partitioning key columns */
  p_rdb_collst_t  collst[p_rdb_MXDCL] ;/* Partitioning key list         */
  p_rdb_csidlst_t csidlst[p_rdb_MXDCL] ;  /* Character set ID list#   */
  p_rdb_dadlst_t  dadlst[p_rdb_MXDCL] ;/* Data address list             */
  union data_area data[p_rdb_MXDCL] ;  /* Data storage area             */
  unsigned int    ndiv ;               /* Number of storage RDAREAs  */
  unsigned char   ncspace[2] ;     /* Space code for each national character
code
                                       type */
  int             flags ;          /* Enhancement flag */
  int           rdno ;          /* Partitioning condition specification order */
  int             rc ;             /* Return value */
  short           i, j, k ;        /* Counter variables */

  struct rdarea {                  /* RDAREA list */
  int             rdareaid ;
  char            rdareaname[31] ;
  } rdarealst [p_rdb_MNCND] ;
```

```
EXEC SQL BEGIN DECLARE SECTION ;
  struct {                /* Embedded variable for hash function name */
    short length ;
    char data[9] ;
  } xhashname ;
 short xncol ;      /* Embedded variable for number of partitioning key columns */
  short xndiv ;      /* Embedded variable for number of table partitions */
  short xdatatype ; /* Embedded variable for data type code */
  short xdatalen ; /* Embedded variable for data length code */
  int   xcharset_id ; /* Embedded variable for character set ID code# */
  struct {          /* Embedded variable for storage RDAREA name */
    short length ;
    char data[31] ;
  } xrdname ;
EXEC SQL END DECLARE SECTION ;

EXEC SQL CONNECT ;
```

#: Specified only when character sets are specified for partitioning keys.

(b) Settings for the data storage area and space code for national character codes

```
for  (k = 0 ; k < p_rdb_MXDCL ; k ++) {
dadlst[k].dataaddr = (unsigned char *)&data[k] ;
}
ncspace[0] = 0x81 ;                    /* Space code */
ncspace[1] = 0x40 ;                    /* Example of shift JIS kanji code */
flags = 0 ;
```

(c) Settings for flags

```
/
*****************************************************************
**/
/* (a) Specifying explicitly                                    */
/*   1 specified for space conversion level and Y for facility */
/* for conversion to a DECIMAL signed normalized number         */
/
*****************************************************************
**/
flags=p_rdb_FLG_SPLVL_1+p_rdb_FLG_DECNRM_Y;
```

(d) Settings for the hash function name, number of partitioning key columns, and number of storage RDAREAs

```
/************************************************************/
/*(a)Setting values with codes                              */
/************************************************************/
```

```
hashcode = p_rdb_HASH6 ;           /* When HASH6 is specified */
ncol = 4 ;                        /* For partitioning with 4 columns */
ndiv = 6 ;                        /* For 6 partitions */

/***********************************************************/
/* (b) Retrieving values from the dictionary table        */
/***********************************************************/
  EXEC SQL
      select  HASH_NAME,

            value(N_DIV_COLUMN,1) ,
            N_RDAREA
          into  :xhashname , :xncol, :xndiv
          from MASTER.SQL_TABLES
          where TABLE_SCHEMA=USER
            and TABLE_NAME='TABLE1' ;

  xhashname.data[xhashname.length] = '\0' ;
  if (strcmp(xhashname.data,"HASH1") == 0) {
      hashcode=p_rdb_HASH1 ;                      /* HASH1 setting */
  } else if (strcmp(xhashname.data,"HASH2") == 0) {
      hashcode=p_rdb_HASH2 ;                      /* HASH2 setting */
  } else if (strcmp(xhashname.data,"HASH3") == 0) {
      hashcode=p_rdb_HASH3 ;                      /* HASH3 setting */
  } else if (strcmp(xhashname.data,"HASH4") == 0) {
      hashcode=p_rdb_HASH4 ;                      /* HASH4 setting */
  } else if (strcmp(xhashname.data,"HASH5") == 0) {
      hashcode=p_rdb_HASH5 ;                      /* HASH5 setting */
  } else if (strcmp(xhashname.data,"HASH6") == 0) {
      hashcode=p_rdb_HASH6 ;                      /* HASH6 setting */
  } else if (strcmp(xhashname.data,"HASHA") == 0) {
      hashcode=p_rdb_HASH0 ;                      /* HASH0 setting */
  } else if (strcmp(xhashname.data,"HASHA") == 0) {
      hashcode=p_rdb_HASHA ;                      /* HASHA setting */
  } else if (strcmp(xhashname.data,"HASHB") == 0) {
      hashcode=p_rdb_HASHB ;                      /* HASHB setting */
  } else if (strcmp(xhashname.data,"HASHC") == 0) {
      hashcode=p_rdb_HASHC ;                     /* HASHC setting  */
  } else if (strcmp(xhashname.data,"HASHD") == 0) {
      hashcode=p_rdb_HASHD ;                     /* HASHD setting  */
  } else if (strcmp(xhashname.data,"HASHE") == 0) {
      hashcode=p_rdb_HASHE ;                     /* HASHE setting  */
  } else if (strcmp(xhashname.data,"HASHF") == 0) {
      hashcode=p_rdb_HASHF ;                     /* HASHF setting  */
  } else {
      /* Add when a hash function is added in the future. */
  }
```

```
ncol = xncol ;
ndiv = xndiv ;

/********************************************************/
/* Displaying table definition information             */
/********************************************************/
printf("Hash function code:%d\n",hashcode);
printf("Number of partitioning key columns:%d\n",ncol);
printf("Number of table partitions:%d\n",ndiv);
printf("\n") ;
```

(e) Settings for the partitioning key specification order, data type code, and data length code (when no character set is specified for partitioning keys)

```
/**********************************************************/
/* (a) Setting values with codes                          */
/**********************************************************/
collst[0].datatype=PDSQL_CHAR ;          /* CHAR(10)*/
collst[0].datalen=10 ;
collst[1].datatype=PDSQL_DECIMAL ;       /* DEC(5,2) */
collst[1].datalen=5*256+2 ;
collst[2].datatype=PDSQL_SMALLINT ;      /* SMALLINT */
collst[2].datalen=2 ;
collst[3].datatype=PDSQL_NVARCHAR ;      /* NVARCHAR(4) */
collst[3].datalen=4 ;

/**********************************************************/
/* (b) Retrieving values from the dictionary table        */
/**********************************************************/

    EXEC SQL
    declare cr1 cursor for
        select  value(DIVCOL_ORDER,1) ,
                DATA_TYPE_CODE,
                DATA_LENGTH_CODE
        from MASTER.SQL_COLUMNS
        where TABLE_SCHEMA=USER
          and TABLE_NAME='TABLE1'
          and DIVIDED_KEY='Y'
        order by 1 asc ;

EXEC SQL open cr1 ;
EXEC SQL whenever not found goto fetch_end1 ;

for  (i = 0 ; ; i++) {
   EXEC SQL fetch cr1 into :xncol , : xdatatype , : xdatalen ;
    collst[i].datatype = xdatatype ;
    collst[i].datalen = xdatalen ;
```

```
  }

  fetch_end1 :
  EXEC SQL close cr1 ;
```

(f) Settings for the partitioning key specification order, data type code, data length code, and character set ID (when a character set is specified for partitioning keys)

```
  /
**************************************************************
*****/
  /* (a) Setting codes                              */
  /
**************************************************************
*****/
  collst[0].datatype=PDSQL_CHAR ;      /* CHAR(10)          */
  collst[0].datalen=10 ;
  csidlst[0].charset_id=p_rdb_CSEBK ;  /* CHARACTER SET EBCDIK
*/
  collst[1].datatype=PDSQL_DECIMAL ;    /* DEC(5,2)
*/
  collst[1].datalen=5*256+2 ;
  csidlst[1].charset_id=p_rdb_CSDEF ;
  collst[2].datatype=PDSQL_SMALLINT ;   /* SMALLINT
*/
  collst[2].datalen=2 ;
  csidlst[2].charset_id=p_rdb_CSDEF ;
  collst[3].datatype=PDSQL_NVARCHAR ;   /* NVARCHAR(4)
*/
  collst[3].datalen=4 ;
  csidlst[3].charset_id=p_rdb_CSDEF ;

  /
**************************************************************
*****/
  /* (b) Retrieving from data dictionary tables      */
  /
**************************************************************
*****/
  EXEC SQL
      declare cr1_cs cursor for
          select  value(DIVCOL_ORDER,1),
                  DATA_TYPE_CODE,
                  DATA_LENGTH_CODE,
                  value(CHARSET_ID,0)
             from MASTER.SQL_COLUMNS
            where TABLE_SCHEMA=USER
              and TABLE_NAME='TABLE1'
```

```
                and DIVIDED_KEY='Y'
              order by 1 asc ;

  EXEC SQL open cr1_cs ;
  EXEC SQL whenever not found goto fetch_end1_cs ;

  for  (i = 0 ; ; i++) {
       EXEC SQL fetch cr1_cs into :xncol , : xdatatype , :
xdatalen, :xcharset_id ;
       collst[i].datatype = xdatatype ;
       collst[i].datalen = xdatalen ;
       csidlst[i].charset_id = xcharset_id ;
  }

  fetch_end1_cs :
  EXEC SQL close cr1_cs ;
```

(g) Settings for storage RDAREA name

```
/************************************************************/
/* Retrieving values from the dictionary table              */
/************************************************************/
EXEC SQL
    declare cr2 cursor for
        select RDAREA_NAME
            from MASTER.SQL_DIV_TABLE
            where TABLE_SCHEMA=USER
              and TABLE_NAME='TABLE1'
            order by DIV_NO asc ;

EXEC SQL open cr2 ;
EXEC SQL whenever not found goto fetch_end2 ;

for (j = 0 ; ; j++) {
    EXEC SQL fetch cr2 into :xrdname ;
    strncpy(rdarealst[j].rdareaname,
            xrdname.data,
            xrdname.length) ;
    rdarealst[j].rdareaname[xrdname.length] = '\0' ;
}

fetch_end2 :
EXEC SQL close cr2 ;

EXEC SQL DISCONNECT ;


/************************************************************/
/* Displaying RDAREA information                            */
```

```
  /*********************************************************/
  printf("RDAREA-name[")  ;
  for (j = 0 ; j<ndiv ; j++) {
      printf("%s",rdarealst[j].rdareaname) ;
      if (j != ndiv-1) {
               printf(",") ;
      } else ;
  }
  printf("]\n") ;
  printf("\n") ;
```

(h) Data setting to be stored in partitioning keys

```
/
**************************************************************
*/
/* Assigning data in binary format.                          */
/* Setting data and call hash function for each line.        */
/
**************************************************************
*/
memcpy((char *)data[0].data_char,"abcdefg  ",10) ;/*"abcdefg "
*/

data[1].data_dec[0] = 0x04 ;
data[1].data_dec[1] = 0x32 ;
data[1].data_dec[2] = 0x1D ;               /* -43.21 */

data[2].data_smallint = 12345 ;            /* 12345 */

/* NCHAR and NVARCHAR specify character codes used in the HiRDB server.*/
data[3].data_varchar.length = 6 ;
data[3].data_varchar.data[0] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[1] = 0xa0 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[2] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[3] = 0xa2 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[4] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[5] = 0xa4 ; /* Example of shift JIS kanji code */

/
**************************************************************
*/
/*Displaying data type code, data length code, and data area   */
/
**************************************************************
*/
print_data(ncol , collst , data) ;

/
```

```
*************************************************************
*/
/* Hash function call                                        */
/
*************************************************************
*/
rc =
p_rdb_dbhash(hashcode,ncol,collst,dadlst,ndiv,ncspace,flags,&r
dno);

switch (rc) {
case p_rdb_RC_RTRN :
  /
************************************************************/
  /* Normal processing                                       */
  /
************************************************************/
  printf("Partitioning condition specification order : %d ->
%s\n",

     rdno,rdarealst[rdno-1].rdareaname) ;
  break ;
default :
/
*************************************************************
*/
/* Adding error processing                                   */
/
*************************************************************
*/
  printf("RETURN CODE=%d\n",rc) ;
  break ;
}

return ;
}

/
*************************************************************
*/
/* Display function for data type code, data length code,     */
/*and data area                                              */
/
*************************************************************
*/
void print_data( short           ncol ,
                 p_rdb_collst_t  *pcollst ,
                 union data_area *pdata )
```

```
{
  int i , j ;                           /* Counter variables */
  int len;
  p_rdb_collst_t   *ccollst ;
  union data_area  *cdata ;

  printf("Partitioning key specification order Data type code
Data length code Binary-format data value\n") ;
  for  (i = 0 , ccollst = pcollst , cdata = pdata ;
        i<ncol ;
        i++ , ccollst++ , cdata++) {
      printf("             %2d         %#.4x         %#.4x  ",
              i+1,ccollst->datatype, ccollst->datalen) ;

    switch (ccollst->datatype) {
      case PDSQL_CHAR :
      case PDSQL_MCHAR :
      case PDSQL_INTEGER :
      case PDSQL_SMALLFLT :
      case PDSQL_FLOAT :
      case PDSQL_SMALLINT :
      case PDSQL_DATE :
      case PDSQL_TIME :
      case PDSQL_TIMESTAMP :
          len=ccollst->datalen ;
          break ;
      case PDSQL_VARCHAR :
      case PDSQL_MVARCHAR :
      case PDSQL_NVARCHAR :
          len=cdata->data_varchar.length+2 ;
          break ;
      case PDSQL_NCHAR :
          len=ccollst->datalen*2 ;
          break ;
      case PDSQL_DECIMAL :
      case PDSQL_YEARTODAY :
      case PDSQL_HOURTOSEC :
          len=ccollst->datalen/256/2+1 ;
          break ;
      default :
          break ;
      }
      for(j=0 ; j<len ;j++){
          printf("%.2X",cdata->data_char[j]) ;
      }
      printf("\n") ;
  }
  printf("\n") ;
return;
}
```

(i) Execution results for HP-UX and shift JIS kanji codes

```
Hash function code: 6
Number of partitioning key columns: 4
Number of table partitions: 6
RDAREA names: [RU01, RU02, RU03, RU04, RU05, RU06]
```

Partitioning key specification order	Data type code	Data length code	Binary format data value
1	0x00c4	0x000a	61626364656667202020
2	0x00e4	0x0502	04321D
3	0x00f4	0x0002	3039
4	0x00b0	0x0004	000682A082A282A4

```
Partitioning condition specification order: 1 > RU01
```

(6) Retrieval from dictionary tables (for hash partitioning)

Examples of retrieval from dictionary tables for hash partitioning are shown below.

(a) Obtaining the hash function name, number of partitioning key columns, and number of table partitions for a hash-partitioned table

```
SELECT HASH_NAME,                   /* Hash function name */
     VALUE(N_DIV_COLUMN,1),         /* Number of partitioning key columns */
     N_RDAREA                       /* Number of table partitioned */
  FROM MASTER.SQL_TABLES
    WHERE TABLE_SCHEMA = authorization-identifier
                               /* Item with matching authorization identifier */
      AND TABLE_NAME = table-identifier
                               /* Item with matching table identifier */
```

(b) Obtaining the partitioning key specification order, data type code, and data length code (when no character set is specified for partitioning keys)

```
SELECT VALUE(DIVCOL_ORDER,1),  /* Partitioning key specification order */
     DATA_TYPE_CODE,           /* Data type code */
     DATA_LENGTH_CODE          /* Data length code */
  FROM MASTER.SQL_COLUMNS
    WHERE TABLE_SCHEMA = authorization-identifier
                              /* Item with matching authorization identifier */
      AND TABLE_NAME= table-identifier
                              /* Item with matching table identifier */
```

```
        AND DIVIDED_KEY='Y'     /* Item that is a partitioning key*/
    ORDER BY 1 ASC
```

(c) Obtaining the partitioning key specification order, data type code, data length code, and character set ID code (when a character set is specified for partitioning keys)

```
SELECT VALUE(DIVCOL_ORDER,1),         /* Partitioning key specification order */
     DATA_TYPE_CODE,                          /* Data type code     */
     DATA_LENGTH_CODE,                        /* Data length code     */
     VALUE(CHARSET_ID,0)                      /* Character set ID code  */
   FROM MASTER.SQL_COLUMNS
     WHERE TABLE_SCHEMA=authorization-identifier    /* Item with matching
authorization identifier */
       AND TABLE_NAME=table-identifier  /* Item with matching table identifier */
       AND DIVIDED_KEY='Y'              /* Item that is a partitioning key */
   ORDER BY 1 ASC
```

(d) Obtaining the storage RDAREA name

```
SELECT  DIV_NO,               /* Partitioning condition specification order */
        RDAREA_NAME           /* Storage RDAREA name */
  FROM MASTER.SQL_DIV_TABLE
    WHERE TABLE_SCHEMA = authorization-identifier
                             /* Item with matching authorization identifier */
      AND TABLE_NAME = table-identifier
                             /* Item with matching table identifier */
  ORDER BY 1 ASC
```

(7) Retrieval from dictionary tables (for matrix partitioning)

Examples of retrieval from dictionary tables for matrix partitioning are shown below.

(a) Obtaining the hash function name, number of partitioning key columns, partitioning key numbers, and number of table partitions for a hash-partitioned table

- Obtaining the hash function name and the number of partitioning key columns

```
select HASH_NAME,                           /* Hash function name */
     value(N_DIV_COLUMN,1),             /* Number of partitioning key columns */
      KEY_NO                                /* Partitioning key number */
 from MASTER.SQL_DIV_TYPE
     where TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
     and TABLE_NAME=table-identifier          /* Item with matching table identifier */
```

- Obtaining the number of partitions in the key

```
select distinct N_DIVISION           /* Number of partitions in key */
from MASTER.SQL_PARTKEY
     where TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
      and TABLE_NAME=table-identifier          /* Item with matching table identifier */
       and KEY_NO=partitioning-key-number          /* Set partitioning key numbers */
                                              /* for hash partitioning */
```

(b) Obtaining the partitioning key specification order, data type code, and data length code (when no character set is specified for partitioning keys)

```
select DIVCOL_ORDER,                              /* Number of partitions in key */
       DATA_TYPE_CODE,                            /* Data type code */
       DATA_LENGTH_CODE                           /* Data length code */
 from MASTER.SQL_COLUMNS X,
      MASTER.SQL_PARTKEY Y
     where X.TABLE_SCHEMA=Y.TABLE_SCHEMA
       and X.TABLE_NAME=Y.TABLE_NAME
       and X.COLUMN_ID=Y.COLUMN_ID
      and Y.TABLE_SCHEMA=authorization-identifier          /* Item with matching
authorization identifier */
       and Y.TABLE_NAME=table-identifier             /* Item with matching table
identifier */
       and Y.KEY_NO=partitioning-key-number               /* Set partitioning key
number */
                                                  /* for hash partitioning */
  order by DIVCOL_ORDER asc
```

(c) Obtaining the partitioning key specification order, data type code, data length code, and character set ID (when a character set is specified for partitioning keys)

```
select DIVCOL_ORDER,                 /* Number of partitions in key */
      DATA_TYPE_CODE,                /* Data type code       */
      DATA_LENGTH_CODE,              /* Data length code      */
      value(CHARSET_ID,0)            /* Character set ID code     */
 from MASTER.SQL_COLUMNS X,
      MASTER.SQL_PARTKEY Y
     where X.TABLE_SCHEMA=Y.TABLE_SCHEMA
       and X.TABLE_NAME=Y.TABLE_NAME
       and X.COLUMN_ID=Y.COLUMN_ID
       and Y.TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
       and Y.TABLE_NAME=table-identifier              /* Item with matching table
identifier   */
```

```
      and Y.KEY_NO=partitioning-key-number     /* Set partitioning key numbers */
                                                /* for hash partitioning */
  order by DIVCOL_ORDER asc
```

(d) Obtaining the storage RDAREA name

```
select DIV_NO,                       /* Partitioning condition specification order */
       RDAREA_NAME                   /* Storage RDAREA name */
 from MASTER.SQL_DIV_TABLE
      where TABLE_SCHEMA=authorization-identifier    /* Item with matching
authorization identifier */
      and TABLE_NAME=table-identifier     /* Item with matching table identifier */
        order by 1 asc
```

Note

The partitioning condition specification order is determined from the partitioning key sequence numbers. The expression follows:

```
N x m - (N - n)
 N: Number of partitions in second dimension
 m: Partitioning key sequence number of first partitioning
key
 n: Partitioning key sequence number of second partitioning
key
```

H.2 Space conversion function

The space conversion function converts single-byte spaces in a character string to double-byte spaces, and vice versa. Because this function lets you know the conversion result without having to store character string data in a database, you can use the function for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when determining the partitioning key for partitioning a table by the key range
- To create an input data file for each RDAREA when loading data to a key-range-partitioned table in units of RDAREAs concurrently using the database load utility

Prerequisites for using the space conversion function

The prerequisites for using the space conversion function are the same as for the hash function for table partitioning. For details, see *H.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the space conversion function

The prerequisites for creating and executing a UAP using the space conversion function are the same as for the hash function for table partitioning. For details, see *H.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the space conversion function

(a) Specification configuration

For details about the specification configuration, see *H.1(3)(b) Specification configuration*.

(b) Space conversion function (p_rdb_conv_space)

Function

The function converts spaces according to the specified conversion type as follows:

Single-byte space → double-byte space:

Converts two consecutive single-byte spaces in a character string to one double-byte space.

Double-byte space → single-byte space:

Converts each double-byte space in a character string to two single-byte spaces.

The function converts spaces in the character string indicated by `srcp` and stores the conversion result in `destp`. The following table shows the combination of `stype` and `flags` arguments and the conversion type:

stype (data type)	flags (conversion type)	
	Single-byte space → double-byte space	Double-byte space → single-byte space
`NCHAR`	Checks two bytes at a time from the top and converts any two consecutive single-byte spaces to a double-byte space.[#] The function does not convert any isolated single-byte space.	Checks two bytes at a time from the top and converts any double-byte space[#] to two single-byte spaces.
`NVARCHAR`		
`MCHAR`	Results in an error.	Checks each character code from the top and converts any double-byte space[#] to two single-byte spaces.
`MVARCHAR`		

#: The function treats the value specified in the `ncspace` argument as the character code for double-byte space.

Header files

`#include<pdauxcnv.h>`

This header file is required to use the space conversion function.

`#include<pdbsqlda.h>`

This header file lets you use macros (with a name beginning with `PDSQL_`) to specify a data type code. If the data type code is to be retrieved from a data dictionary table, this header file is not necessary.

Format

```
int p_rdb_conv_space(char            *srcp,
                     unsigned char   stype,
                     unsigned int    srcl,
                     char            *destp,
                     unsigned char   ncspace[2],
                     int             flags);
```

Arguments

`srcp (input)`

Specifies the start address of the character string storage area.

`stype (input)`

Specifies the data type before conversion. Specifiable data types are as follows:

Macro name	Data type
PDSQL_NCHAR	NCHAR
PDSQL_NVARCHAR	NVARCHAR
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

`srcl (input)`

Specifies the length of the character string specified in srcp. For a variable-length character string, specify the length of the actual character string (in bytes) that is stored in the area indicated by srcp.

`destp (output)`

Sets the start address of the character string storage area after conversion. Allocate this area indicated by destp on the side that calls the space conversion function.

`ncspace (input)`

Specifies a two-byte area that contains the double-byte space character for the national character code used in the HiRDB server. For the double-byte space characters that can be specified in ncspace, see *Table H-4 Double-byte space characters specified in ncspace*.

`flags (input)`

Specifies the conversion type. Available conversion types are as follows:

Macro name	Conversion type
`p_rdb_HALF_TO_FULL_SPACE`	Single-byte space → double-byte space
`p_rdb_FULL_TO_HALF_SPACE`	Double-byte space → single-byte space

Return values

data type: `int`

`p_rdb_RC_RTRN(0)`

Normal termination.

`p_rdb_RC_ERRINVF(-8)`

Invalid flags argument

`p_rdb_RC_ERRTYPC(-9)`

Invalid data type

Notes

1. The National Language Support (NLS) facility provided by the OS is used to convert double-byte spaces into single-byte spaces. Therefore, before invoking the space conversion function, you must use the `setlocale` function to set an appropriate locale to the `LC_CTYPE` or `LC_ALL` category. Additionally, for a Windows UAP, a Linux UAP with a character code type of `SJIS`, or a UAP with a character code type of `CHINESE`, you must invoke the `p_rdb_set_lang` function before invoking the space conversion function. For details about the `p_rdb_set_lang` function, see *H.4 Character code type specification function*.

   If the character code type of the character string indicated by the `srcp` argument contradicts the locale specified by the `setlocale` or `p_rdb_set_lang` function, the operation cannot be guaranteed.

2. Operation is guaranteed if the data input area is the same as the data output

area, or if the output area is located before the input area and the latter half of the output area overlaps the first half of the input area.

3. Be sure to specify an appropriate value in srcl because the function does not check the length of a character string for any error.
4. The function uses 0x20 as the character code for a single-byte space and the character code specified in the ncspace argument as the character code for a double-byte space.
5. The data types that can be specified for input are NCHAR, NVARCHAR, MCHAR, and MVARCHAR.
6. For a variable-length character string, the function references srcl to determine the length of character string to be converted. Specify the length without the real length section in the srcl argument.
7. The real length section of a variable-length character string remains unchanged after space conversion.
8. If the character code type is Unicode, the operation of this function cannot be guaranteed. If the character code type is Unicode, use the p_rdb_conv_space_utf8 function. For details on this function, see *(c) Space conversion function (p_rdb_conv_space_utf8)*.

(c) Space conversion function (p_rdb_conv_space_utf8)

Function

This function converts double-byte spaces into single-byte spaces when the character code is Unicode (UTF-8). It converts each double-byte space inside a character string into two single-byte spaces.

This function applies space conversion to the space characters inside the character string indicated by srcp. The conversion result is stored in destp and the character string length following the conversion is stored in destl.

The following table shows the combinations of the stype and flags arguments, along with conversion details.

stype (data type)	flags (conversion type)	
	Single-byte spaces -> Double-byte spaces	Double-byte spaces -> Single-byte spaces
MCHAR	An error occurs.	Character codes are checked from the beginning, and any double-byte spaces# found are converted into single-byte spaces.
MVARCHAR		

#: 0xE38080 is treated as a double-byte space character code.

Header files

`#include <pdauxcnv.h>`

This header file is required to use the space conversion function.

`#include <pdbsqlda.h>`

This header file lets you use macros (with a name beginning with PDSQL_) to specify a data type code. If the data type code is to be retrieved from a data dictionary table, this header file is not necessary.

Format

```
int p_rdb_conv_space_utf8(char          *srcp,
                  unsigned char         stype,
                  unsigned int          srcl,
                  char                  *destp,
                  unsigned int          *destl,
                  int                   flags) ;
```

Arguments

`srcp` (input)

Specifies the start address of the character string storage area.

`stype` (input)

Specifies the data type before conversion. The following table shows the data types that can be specified:

Macro name	Data type
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

`srcl` (input)

Specifies the length of the character string specified by srcp. For a variable-length character string, this argument specifies the length (units: bytes) of the character string actually stored in the area indicated by srcp.

`destp` (output)

Specifies the start address of the character string storage area after conversion. The area indicated by destp must be allocated in the system that invokes the space conversion function.

`destl` (output)

Specifies the length of the character string specified by destp. For a variable-length character string, this argument specifies the length (units:

bytes) of the character string actually stored in the area indicated by destp.

flags (input)

Specifies a conversion type. The following table shows the conversion types:

Macro name	Conversion type
p_rdb_FULL_TO_HALF_SPACE	Double-byte spaces -> Single-byte spaces

Return values

Data type: int

p_rdb_RC_RTRN(0)

Normal termination

p_rdb_RC_ERRINVF(-8)

Invalid flags argument

p_rdb_RC_ERRTYPC(-9)

Invalid data type

Notes

1. This space conversion function is used only for Unicode (UTF-8).
2. Before invoking this function, you must set UTF8 for the lang argument and invoke the p_rdb_set_lang function. For details about the p_rdb_set_lang function, see *H.4 Character code type specification function*.
3. If the data input area is the same as the data output area, or if the output area is located before the input area and the second half of the output area overlaps with the first half of the input area, the correct operation of the function is guaranteed.
4. Because errors related to character string length are not checked, you must enter an appropriate value in srcl.
5. The single- and double-byte space codes use 0x20 and 0xE38080, respectively.
6. The data types that can be set for the input are MCHAR and MVARCHAR.
7. When a character string has a variable length, srcl is referenced for the length of the character string to be converted. Specify for srcl a length that excludes the effective-length portion.
8. Because space conversion converts each double-byte space (3 bytes) into two single-byte spaces (2 bytes), the length of the character string following

conversion is shorter than that before the conversion.

9. When a character string has a variable length, the effective-length portion of the area for storing character strings following conversion stores the data length following the conversion.
10. The data inside the area for storing character strings following conversion is guaranteed only for the length specified in `destl`.
11. If this function is invoked by a character code other than Unicode (`UTF-8`), the operation of this function cannot be guaranteed.

H.3 Function for conversion to a DECIMAL signed normalized number

The function for conversion to a DECIMAL signed normalized number sets the sign for `DECIMAL` data to either `X'C'` or `X'D'` (for a value of `0`, the sign is `X'C'`). Because this function lets you obtain the normalized sign without having to store `DECIMAL` data in a database, you can use it for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when `pd_dec_sign_normalize=Y` is specified in the system definition and the key for partitioning a table is determined by the key range
- To create an input data file for each RDAREA when `pd_dec_sign_normalize=Y` is specified in the system definition and data is loaded to a key-range-partitioned table in units of RDAREAs concurrently using the database load utility

Prerequisites for using the function for conversion to a DECIMAL signed normalized number

The prerequisites are the same as those for the hash function for table partitioning. For details, see *H.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the function for conversion to a DECIMAL signed normalized number

The prerequisites are the same as those for the hash function for table partitioning. For details, see *H.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the function for conversion to a DECIMAL signed normalized number

(a) Specification configuration

For details about the specification configuration, see *H.1(3)(b) Specification configuration*.

(b) Function for conversion to a DECIMAL signed normalized number (p_rdb_dec_sign_norm)

Function

The function normalizes the sign of DECIMAL data indicated by srcp as follows:

Before normalization	After normalization
X'A'	X'C'
X'B'	X'D'#
X'C'	X'C'
X'D'	X'D'#
X'E'	X'C'
X'F'	X'C'
X'0' to X'9'	Error

#: If the absolute value of data is 0, the sign part is set to X'C'.

Header file

#include<pdauxcnv.h>

This header file is required to use the function for conversion to a DECIMAL signed normalized number.

Format

```
int p_rdb_dec_sign_norm(unsigned char   *srcp,
                        short           srcl,
                        unsigned char   *destp);
```

Arguments

srcp (input)

Specifies the start address of the DECIMAL data to be normalized.

srcl (input)

Specifies the length code of the DECIMAL data indicated by the srcp argument. Specifiable data length codes are as follows:

Data type	Data length code
INTERVAL YEAR TO DAY	8 x 256
INTERVAL HOUR TO SECOND	6 x 256

Data type	Data length code
DECIMAL[(*p*[,*q*])]	*p* x 256 + *q* (If *p* is omitted, 15 is assumed; if *q* is omitted, 0 is assumed.)

destp (output)

Sets the normalized DECIMAL data. Allocate this area indicated by destp on the side that calls the function for conversion to a DECIMAL signed normalized number.

Return values

data type: int

p_rdb_RC_RTRN(0)

Normal termination.

p_rdb_RC_ERRDFRM(-12)

Invalid sign part for data.

Notes

1. The function does not check anything for error other than the sign part of DECIMAL data. Operation is not guaranteed if DECIMAL data is invalid or the data length code specified by the srcl argument contradicts the DECIMAL data.
2. Operation is guaranteed if the data input area is the same as the data output area, or if the output area is located before the input area and the latter half of the output area overlaps the first half of the input area.

H.4 Character code type specification function

The character code type specification function is used to pass the type of the character code from a UAP to a hash function for table partitioning or a space conversion function.

By using this function to specify the type of the character code, you can execute processing that depends on the type of character code, such as the hash function for table partitioning and the space conversion function.

Prerequisites for using the character code type specification function

The prerequisites are the same as those for the hash function for table partitioning. For details, see *H.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the character code type specification function

The prerequisites are the same as those for the hash function for table partitioning. For details, see *H.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the character code type specification function

(a) Specification configuration

For details about the specification configuration, see *H.1(3)(b) Specification configuration*.

(b) Character code type specification function (p_rdb_set_lang)

Function

The character code type specification function specifies the type of character code to be handled by the hash function for table partitioning and the space conversion function.

Header file

```
#include<pdauxcnv.h>
```

This header file is required to use the character code type specification function.

Format

```
int p_rdb_set_lang(char *lang);
```

Arguments

`lang (input)`

Specifies the type of character encoding to be handled by the hash function for table partitioning and the space conversion function. Specifiable character encodings are as follows:

Type of character codes	Value of lang argument
Shift JIS kanji codes[#1]	"SJIS"
EUC Chinese kanji codes	"CHINESE"
Single-byte character codes[#2]	"C"
Unicodes (UTF-8)	"UTF8"
Chinese kanji codes (GB18030)	"GB18030"

#1: Can be specified for Linux and Windows.

#2: Can be specified for Windows.

If an empty character string (for example, `p_rdb_set_lang ("")`) is specified, the operation is as follows:

UNIX environment

The `setlocale` function executed before this function sets the character code type corresponding to the locale that was set to the `LC_ALL` category. If the `setlocale` function has not been executed, the character code type corresponding to the default locale of the `LC_ALL` category is set.

Windows environment

The default character code type of the OS is set. However, if the default character code type is set to a type that is not listed in the above table, the operation cannot be guaranteed.

Return values

data type: `int`

`p_rdb_RC_RTRN(0)`

Normal termination.

`p_rdb_RC_ERRIVLG(-10)`

Invalid character encoding type.

Notes

1. You must execute `p_rdb_set_lang` if any of the following conditions is applicable:

   - When setting a character code type from a UAP in a Windows environment
   - When invoking `p_rdb_conv_space_utf8` from a UAP in a UNIX environment#
   - When setting the character code type to `SJIS` from a UAP in a Linux environment
   - When setting the character code type to `CHINESE` from a UAP in a UNIX environment

   #: Before invoking `p_rdb_conv_space_utf8`, execute `p_rdb_set_lang`. When invoking the space conversion function `p_rdb_conv_space`, use the `setlocale` function provided by the OS instead of `p_rdb_set_lang`.

2. In an UNIX environment, after setting a character code type using this function, to use a character code type that cannot be used to use another function, issue `p_rdb_set_lang("")` first and then invoke the

setlocale function to reset the character code type to an appropriate one.

I. Scalar Functions That Can Be Specified in the Escape Clause

The following table lists the scalar functions that can be specified in the escape clause.

Table I-1: Scalar functions that can be specified in the escape clause

Scalar function	Standard format of scalar function	Format conversion[#1]		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
Mathematical functions	ABS(number)	None	None	None
	ACOS(float)	None	MASTER.ACOS(float)	None
	ASIN(float)	None	MASTER.ASIN(float)	None
	ATAN(float)	None	MASTER.ATAN(float)	None
	ATAN2(float1, float2)	None	MASTER.ATAN2(float1, float2)	None
	CEILING(number)[#2]	CEIL(number) 08-02	MASTER.CEIL(number)	None
	COS(float)	None	MASTER.COS(float)	None
	COT(float)[#3, #4]	None	None	None
	DEGREES(number)	None	MASTER.DEGREES(number)	None
	EXP(float)	None	MASTER.EXP(float)	None
	FLOOR(number)	None	MASTER.FLOOR(number)	None
	LOG(float)[#2]	LN(float) 08-02	MASTER.LN(float)	`LN(float)`
	LOG10(float)	None	MASTER.LOG10(float)	None
	MOD(integer1, integer2)	None	None	None
	PI()	None	MASTER.PI()	None
	POWER(number, power)	None	MASTER.POWER(number, power)	None

Scalar function	Standard format of scalar function	Format conversion#1		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
	RADIANS(number)	None	MASTER.RADIANS(number)	None
	RAND(integer)#3, #4	None	None	None
	ROUND(number, places)	None	MASTER.ROUND(number, places)	None
	SIGN(number)	None	MASTER.SIGN(number)	None
	SIN(float)	None	MASTER.SIN(float)	None
	SQRT(float)	None	MASTER.SQRT(float)	None
	TAN(float)	None	MASTER.TAN(float)	None
	TRUNCATE(number, places)#2	TRUNC(number, places) 08-02	MASTER.TRUNC(number, places)	None
String functions	ASCII(string)	None	MASTER.ASCII(string)	None
	BIT_LENGTH(string)#3	None	None	None
	CHAR(code)#2	CHR(code) 08-02	MASTER.CHR(code)	None
	CHAR[ACTER]_LENGTH(string)#3	None	None	None
	CONCAT(string1, string2)#3	None	None	None
	DIFFERENCE(string1, string2)#3	None	None	None
	INSERT(string1, start, length, string2)#2	INSERTSTR(string1, start, length, string2) 08-02	MASTER.INSERTSTR(string1, start, length, string2)	None

Scalar function	Standard format of scalar function	Format conversion[#1]		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
	LCASE(string)[#2]	LOWER(string) 08-02	Same as at left	Same as at left
	LEFT(string, count)[#2]	LEFTSTR(string, count) 08-02	MASTER.LEFTSTR(string, count)	None
	LENGTH(string)	None	None	None
	LOCATE(string1, string2[, start])[#2]	POSITION (string1 IN string2 [FROM start]) 08-02	Same as at left	Same as at left
	LTRIM(string)	None	MASTER.LTRIM(string)	TRIM(LEADING FROM string)
	OCTET_LENGTH(string)[#3]	None	None	None
	POSITION(character IN character)	None	None	None
	REPEAT(string, count)[#3]	None	None	None
	REPLACE(string1, string2, string3)	None	MASTER.REPLACE(string1, string2, string3)	None
	RIGHT(string, count)[#2]	RIGHTSTR(string, count) 08-02	MASTER.RIGHTSTR(string, count)	None
	RTRIM(string)	None	MASTER.RTRIM(string)	TRIM(TRAILING FROM string)
	SOUNDEX(string)[#3]	None	None	None
	SPACE(count)[#3]	None	None	None

Scalar function	Standard format of scalar function	Format conversion[#1]		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
	SUBSTRING(string, start, length)[#2]	SUBSTR(string, start, length)	Same as at left	Same as at left
	UCASE(string)[#2]	UPPER(string)	Same as at left	Same as at left
Time and date functions	CURDATE()[#2]	CURRENT DATE	Same as at left	Same as at left
	CURRENT_DATE()[#2]	CURRENT DATE	Same as at left	Same as at left
	CURTIME()[#2]	CURRENT TIME	Same as at left	Same as at left
	CURRENT_TIME	None	None	None
	CURRENT_TIME(time-precision)[#2, #5] The *time-precision* argument specifies the precision of the fraction of a second in the return value.	CURRENT TIME 08-02	Same as at left	None
	CURRENT_TIMESTAMP[(timestamp-precision)] The *timestamp-precision* argument specifies the precision of the fraction of a second in the returned timestamp.	None	None	None
	DAYNAME(date)	None	MASTER.DAYNAME(date)	None
	DAYOFMONTH(date)[#3]	None	None	None
	DAYOFWEEK(date)	None	MASTER.DAYOFWEEK(date)	None

Scalar function	Standard format of scalar function	Format conversion[#1]		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
	DAYOFYEAR(date)	None	MASTER.DAYOFYEAR(date)	None
	EXTRACT(extract-field FROM extract-source)[#3]	None	None	None
	HOUR(time)	None	None	None
	MINUTE(time)	None	None	None
	MONTH(time)	None	None	None
	MONTHNAME(date)	None	MASTER.MONTHNAME(date)	None
	NOW()[#2]	CURRENT TIMESTAMP(6) 08-02	Same as at left	Same as at left
	QUARTER(date)	None	MASTER.QUARTER(date)	None
	SECOND(time)	None	None	None
	TIMESTAMPADD(interval, count, timestamp)[#3]	None	None	None
	TIMESTAMPDIFF(interval, timestamp1, timestamp2)[#3]	None	None	None
	WEEK(date)	None	MASTER.WEEK(date)	None
	YEAR(date)	None	None	None
System functions	DATABASE()[#3]	None	None	None
	IFNULL(expression, value)[#3]	None	None	None
	USER()[#4]	USER 08-02	Same as at left	Same as at left

Scalar function	Standard format of scalar function	Format conversion#1		
		Type2	Type4	Type4 (XDM/RD E2 is connected)
Data type conversion function	CONVERT(value, SQLtype)#2, #4	None	None	None

#1

Indicates the format of the converted scalar function in the analysis of the escape syntax in the `Statement` object. *None* means that the function is not converted. *xx-xx* indicates an added version.

#2

The standard format differs from the HiRDB format or XDM/RD E2 format.

#3

There is no corresponding scalar function in HiRDB or XDM/RD E2.

#4

Because the function is not supported by HiRDB or XDM/RD E2, specifying the scale function (standard format) in the escape syntax will result in a HiRDB server or XDM/RD E2 error. *xx-xx* indicates an added version of Type2 JDBC driver.

#5

A Type4 JDBC driver treats the precision of the fraction of a second as being 0 because it converts the function to `CURRENT TIME`. The precision of seconds specified in the argument is ignored.

J. Character Code Conversion Rules When Character Sets Are Used

This section describes the character code conversion rules when character sets are used.

J.1 Converting shift JIS kanji codes to EBCDIK

This section describes conversion from shift JIS kanji codes to EBCDIK.

Low	High order															
order	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	[NUL]	[DLE]	Space	0	@	P	`	p				ｰ	ﾀ	ﾐ		
	0	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
	0	10	40	F0	7C	D7	79	76	20	30	57	58	91	A5	B2	DC
1	[SOH	[DC1]	!	1	A	Q	a	q			｡	ｱ	ﾁ	ﾑ		
	1	11	21	31	41	51	61	71	81	91	A1	B1	C1	D1	E1	F1
	1	11	4F	F1	C1	D8	59	77	21	31	41	81	92	A6	B3	DD
2	[STX]	[DC2]	"	2	B	R	b	r			｢	ｲ	ﾂ	ﾒ		
	2	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2
	2	12	7F	F2	C2	D9	62	78	22	1A	42	82	93	A7	B4	DE
3	[ETX]	[DC3]	#	3	C	S	c	s			｣	ｳ	ﾃ	ﾓ		
	3	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3
	3	13	7B	F3	C3	E2	63	80	23	33	43	83	94	A8	B5	DF
4	[EOT]	[DC4]	$	4	D	T	d	t			､	ｴ	ﾄ	ﾔ		
	4	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4
	37	3C	E0	F4	C4	E3	64	8B	24	34	44	84	95	A9	B6	EA
5	[ENQ]	[NAK	%	5	E	U	e	u			･	ｵ	ﾅ	ﾕ		
	5	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5
	2D	3D	6C	F5	C5	E4	65	9B	25	35	45	85	96	AA	B7	EB
6	[ACK	[SYN	&	6	F	V	f	v			ｦ	ｶ	ﾆ	ﾖ		
	6	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6
	2E	32	50	F6	C6	E5	66	9C	6	36	46	86	97	AC	B8	EC
7	[BEL]	[ETB]	'	7	G	W	g	w			ｧ	ｷ	ﾇ	ﾗ		
	7	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7
	2F	26	7D	F7	C7	E6	67	A0	17	8	47	87	98	AD	B9	ED
8	[BS]	[CAN	(	8	H	X	h	x			ｨ	ｸ	ﾈ	ﾘ		
	8	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8
	16	18	4D	F8	C8	E7	68	AB	28	38	48	88	99	AE	CA	EE
9	[HT]	[EM]	)	9	I	Y	i	y			ｩ	ｹ	ﾉ	ﾙ		
	9	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9
	5	19	5D	F9	C9	E8	69	B0	29	39	49	89	9A	AF	CB	EF
A	[NL]	[SUB]	*	:	J	Z	j	z			ｪ	ｺ	ﾊ	ﾚ		
	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA
	15	3F	5C	7A	D1	E9	70	B1	2A	3A	51	8A	9D	BA	CC	FA
B	[VT]	[ESC]	+	;	K	[	k	{			ｫ	ｻ	ﾋ	ﾛ		
	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB
	0B	27	4E	5E	D2	4A	71	C0	2B	3B	52	8C	9E	BB	CD	FB
C	[FF]	[FS]	,	<	L	\	l	|			ｬ	ｼ	ﾌ	ﾜ		
	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC
	0C	1C	6B	4C	D3	5B	72	6A	2C	4	53	8D	9F	BC	CE	FC
D	[CR]	[GS]	-	=	M	]	m	}			ｭ	ｽ	ﾍ	ﾝ		
	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD
	0D	1D	60	7E	D4	5A	73	D0	9	14	54	8E	A2	BD	CF	FD
E	[SO]	[RS]	.	>	N	^	n	~			ｮ	ｾ	ﾎ	ﾞ		
	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE
	0E	1E	4B	6E	D5	5F	74	A1	0A	3E	55	8F	A3	BE	DA	FE
F	[SI]	[US]	/	?	O	_	o	[DEL]			ｯ	ｿ	ﾏ	ﾟ		E0
	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF
	0F	1F	61	6F	D6	6D	75	7	1B	E1	56	90	A4	BF	DB	FF

How to interpret the table:

Row 1: Character to be converted

Row 2: Code before conversion

Row 3: Code after conversion

A double-byte character (whose first byte is `0x81` to `0x9F`, `0xE0` to `0xEF`, or `0xF0` to `0xFC`) is treated as two single-byte characters.

J.2 Converting EBCDIK to shift JIS kanji codes

This section describes conversion from EBCDIK to shift JIS kanji codes.

Low	High order															
order	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	[NUL] 0 0	[DLE] 10 10	20 80	30 90	Space 40 20	& 50 26	- 60 2D	j 70 6A	s 80 73	ｿ 90 BF	w A0 77	y B0 79	{ C0 7B	} D0 7D	$ E0 24	0 F0 30
1	[SOH] 1 1	[DC1] 11 11	21 81	31 91	｡ 41 A1	ｪ 51 AA	/ 61 2F	k 71 6B	ｱ 81 B1	ﾀ 91 C0	~ A1 7E	z B1 7A	A C1 41	J D1 4A	0 E1 9F	1 F1 31
2	[STX] 2 2	[DC2] 12 12	22 82	[SYN] 32 16	｢ 42 A2	ｫ 52 AB	b 62 62	l 72 6C	ｲ 82 B2	ﾁ 92 C1	ﾍ A2 CD	B2 E0	B C2 42	K D2 4B	S E2 53	2 F2 32
3	[ETX] 3 3	[DC3] 13 13	23 83	33 93	｣ 43 A3	ｬ 53 AC	c 63 63	m 73 6D	ｳ 83 B3	ﾂ 93 C2	ﾎ A3 CE	B3 E1	C C3 43	L D3 4C	T E3 54	3 F3 33
4	4 9C	14 9D	24 84	34 94	､ 44 A4	ｭ 54 AD	d 64 64	n 74 6E	ｴ 84 B4	ﾃ 94 C3	ﾏ A4 CF	B4 E2	D C4 44	M D4 4D	U E4 55	4 F4 34
5	[HT] 5 9	[NL] 15 0A	25 85	35 95	･ 45 A5	ｮ 55 AE	e 65 65	o 75 6F	ｵ 85 B5	ﾄ 95 C4	ﾐ A5 D0	B5 E3	E C5 45	N D5 4E	V E5 56	5 F5 35
6	6 86	[BS] 16 8	[ETB] 26 17	36 96	ｦ 46 A6	ｯ 56 AF	f 66 66	p 76 70	ｶ 86 B6	ﾅ 96 C5	ﾑ A6 D1	B6 E4	F C6 46	O D6 4F	W E6 57	6 F6 36
7	[DEL] 7 7F	17 87	[ESC] 27 1B	[EOT] 37 4	ｧ 47 A7	57 A0	g 67 67	q 77 71	ｷ 87 B7	ﾆ 97 C6	ﾒ A7 D2	B7 E5	G C7 47	P D7 50	X E7 58	7 F7 37
8	8 97	[CAN] 18 18	28 88	38 98	ｨ 48 A8	ｰ 58 B0	h 68 68	r 78 72	ｸ 88 B8	ﾇ 98 C7	ﾓ A8 D3	B8 E6	H C8 48	Q D8 51	Y E8 59	8 F8 38
9	9 8D	[EM] 19 19	29 89	39 99	ｩ 49 A9	a 59 61	i 69 69	` 79 60	ｹ 89 B9	ﾈ 99 C8	ﾔ A9 D4	B9 E7	I C9 49	R D9 52	Z E9 5A	9 F9 39
A	0A 8E	1A 92	2A 8A	3A 9A	[4A 5B	] 5A 5D	| 6A 7C	: 7A 3A	ｺ 8A BA	ﾉ 9A C9	ﾕ AA D5	ﾚ BA DA	CA E8	DA EE	EA F4	FA FA
B	[VT] 0B 0B	1B 8F	2B 8B	3B 9B	. 4B 2E	\ 5B 5C	, 6B 2C	# 7B 23	t 8B 74	u 9B 75	x AB 78	ﾛ BB DB	CB E9	DB EF	EB F5	FB FB
C	[FF] 0C 0C	[FS] 1C 1C	2C 8C	[DC4] 3C 14	< 4C 3C	* 5C 2A	% 6C 25	@ 7C 40	ｻ 8C BB	v 9C 76	ﾖ AC D6	ﾜ BC DC	CC EA	DC F0	EC F6	FC FC
D	[CR] 0D 0D	[GS] 1D 1D	[ENQ] 2D 5	[NAK] 3D 15	(4D 28	) 5D 29	_ 6D 5F	' 7D 27	ｼ 8D BC	ﾊ 9D CA	ﾗ AD D7	ﾝ BD DD	CD EB	DD F1	ED F7	FD FD
E	[SO] 0E 0E	[RS] 1E 1E	[ACK] 2E 6	3E 9E	+ 4E 2B	; 5E 3B	> 6E 3E	= 7E 3D	ｽ 8E BD	ﾋ 9E CB	ﾘ AE D8	ﾞ BE DE	CE EC	DE F2	EE F8	FE FE
F	[SI] 0F 0F	[US] 1F 1F	[BEL] 2F 7	[SUB] 3F 1A	! 4F 21	^ 5F 5E	? 6F 3F	" 7F 22	ｾ 8F BE	ﾌ 9F CC	ﾙ AF D9	ﾟ BF DF	CF ED	DF F3	EF F9	E0 FF FF

How to interpret the table:

Row 1: Character to be converted

Row 2: Code before conversion

Row 3: Code after conversion

K. HiRDB SQL Tuning Advisor Environment Setup

HiRDB SQL Tuning Advisor is not supported.

L. Maximum and Minimum HiRDB Values

The HiRDB system defines a specific range of acceptable values for each item. This appendix lists the maximum and minimum values allowed.

Table L-1: HiRDB maximum and minimum values

Classification	Item	Minimum value	Maximum value	Unit
Database manipulation	Number of retrieval items	1	30,000	Tables
	Number of update columns	1	30,000	
	Number of sort columns	1	255	
	Number of grouped columns	0 or 1[#1]	255	
	Number of duplicate locked columns	1	255	
	Number of nested logical operations	0	255	
	Number of value expressions of IN predicate	1	30,000	
	Number of nested scalar functions	0	255	
	Length of character string literal in SQL	0	255	Bytes
	Length of national character string literal in SQL	0	127	Characters
	Length of mixed character string literal in SQL	0	255	Bytes
	Length of one SQL statement	1	2,000	Kilobytes
	Number of tables that can be specified in one SQL statement	1	64	Tables
	Number of correlation names that can be specified in one SQL statement	0	65	
	Number of locked base tables in `LOCK` statement	1	64	
	Number of arguments in `CALL` statement	0	30,000	
	Row length of work table[#2]	6	32,720	Bytes

Classification	Item	Minimum value	Maximum value	Unit
UAP	Number of SQL statements in one UAP	1	4,095	Tables
	Number of cursors in one UAP	0	1,023	
	Number of ? parameters in SQL statement	0	30,000	
	Number of embedded variables in SQL statement	0	30,000	

#1: If the GROUP BY clause is specified, the minimum value is 1. If the HAVING clause is specified without the GROUP BY clause, or if a set function is specified in the SELECT clause, the minimum value is 0.

#2: Some SQL statements require a work table file. For details about work table files, see the *HiRDB Version 9 Installation and Design Guide*.

M. List of Sample UAPs

The following table lists the sections where sample UAPs can be found.

Table M-1: List of sample UAPs

Classification	Description	Section
Creating stored routines	Numbering using a stored procedure	4.2.2(7)
	Defining and calling an SQL stored procedure	4.3.1(3)
	Using the results-set return facility to define and call an SQL statement procedure	4.3.1(6)(c)
	Defining and calling an SQL stored function	4.3.2(2)
	External Java stored routine program	9.3
	Java stored procedure program that uses the results-set return facility	9.4.5(3)
	External C stored routine program	10.3
Using arrays	Coding the `FETCH` facility using arrays	4.8.1(4)
	Coding the `INSERT` facility using arrays	4.8.2(4)
	Coding the `UPDATE` facility using arrays	4.8.3(4)
	Coding the `DELETE` facility using arrays	4.8.4(4)
Using the multi-connection facility	Coding the multi-connection facility	4.10(3)
Using the locator facility	Coding the locator facility	4.16.4
Creating UAPs in C language	Example of basic operation	7.2.2(1)
	Example that uses a user-defined SQL descriptor area	7.2.2(2)
	Example of manipulating LOB data	7.2.2(3)
	Eliminating the embedded SQL declare section	8.2.5(2)
	Example that uses the structures of C language	8.2.7(3)
	Using hash partitioning	*H.1(5)*

Classification	Description	Section
	Retrieving records	*HiRDB First Step Guide*
Creating UAPs in COBOL	Example of basic operation	7.3.2(1)
	Example that uses a row interface	7.3.2(2)
	Example that uses the `TYPE`, `TYPEDEF`, and `SAME AS` clauses	7.3.2(3)
	Retrieving records	*HiRDB First Step Guide*
Executing commands from UAPs	UAP that loads data	12.2
Using the HiRDB.NET data provider	Connecting to the database	16.11.1
	Executing SQL statements	16.11.2
	Executing transactions	16.11.3
	Executing retrieval statements	16.11.4
	Executing the `INSERT` facility by using arrays	16.11.5
	Executing repetition columns	16.11.6
	Checking for an error in SQL statements and acquiring error information	16.11.7
Using DataSource and JNDI (Type2 JDBC driver)	Registering DataSource into JNDI	17.4.1(3)
	Acquiring DataSource from JNDI	17.4.1(4)
	Connecting to the database	17.4.1(5)
Using DataSource and JNDI (Type4 JDBC driver)	Registering DataSource into JNDI	18.3(3)
	Acquiring DataSource from JNDI	18.3(4)
	Connecting to the database	18.3(5)
Using a JDBC driver	UAP that uses a JDBC driver	18.16
Using SQLJ	Coding connection to and disconnection from a HiRDB server	19.3.7

Classification	Description	Section
	Coding data retrieval using the cursor	19.3.8
	Receiving a dynamic result set	19.3.9
	Operation with JDBC	19.3.10
	Coding that uses a native interface	19.4.4
	Retrieving records	*HiRDB First Step Guide*

Index

Symbols

? parameter, specifying value using repetition column as 1272

A

abbreviations for products vii
abstract data type, manipulating data in table with 89
access path information 994
access privilege
- granting 8
- revoking 9

ADO.NET-compatible application program, HiRDB access from 1119
ALLOCATE CONNECTION HANDLE 13
ALLOCATE CURSOR statement 9
ALTER PROCEDURE 5
ALTER ROUTINE 5
ALTER TABLE 5
ALTER TRIGGER 5
AND 62
AND multiple index usage [SQL optimization] 329
AND multiple indexes, suppressing use of [SQL optimization option] 653
AND PLURAL INDEXES SCAN [SQL optimization] 333
arithmetic operations on numeric data 69
array
- DELETE facility using 402
- facilities using 377
- FETCH facility using 377
- INSERT facility using 386
- UPDATE facility using 399

Array class [Type2 JDBC driver] 1270, 1326
ASSIGN LIST statement 9
assignment statement 12
audit event
- defining 6
- deleting 7

auditor's password, changing 8
automatic reconnect facility 443
AVG 72

B

base table 29
- defining 6

base table search process information 1001
batch acquisition from functions provided by plug-ins, facility for [SQL optimization option] 658
batch updating
- JDBC2.0 basic facility 1246
- JDBC2.1 Core API 1846

BEGIN DECLARE SECTION 13
BETWEEN predicate 58
- data search using 59

BINARY data
- addition update of 436
- partial extraction of 436

Blob class [Type2 JDBC driver] 1326
BLOB data
- addition update of 436
- file output facility for 431
- partial extraction of 436

Blob interface [JDBC1.2 core API] 1829
BLOB type, notes on using [Type2 JDBC driver] 1278
block mode 626
block transfer facility 373
Boolean predicate, searching for data using 63

C

C data description 2114
CALL statement 9
CallableStatement class
- JDBC1.0 facility 1237
- Type2 JDBC driver 1316

cataloged text, copying 14

character code classification 798, 810
character code conversion facility [Type2 JDBC driver] 1312
character code type specification function 2282
character conversion facility [Type4 JDBC driver] 1939
character string, outputting to file 15
check pending status 155, 157
Class file, creating 887
client environment definition 533, 562
    supported [Type4 JDBC driver] 1940
client environment setup 483
CLOSE statement 9
COBOL data description 2135
coding example
    C language 722, 731, 743
    COBOL 759, 775
coding rule
    C++ 786
    COBOL 750
column 28
    name 28
Column Name Descriptor Area 2093
    contents of 2093
    expanding 2095
    organization of 2093
COMMAND EXECUTE 14
command trace facility 1018
command trace file, backing up 1019
command trace information
    how to obtain 1018
    interpreting 1018
commands, executing from UAP 14
COMMENT 6
comment, adding 6
COMMIT statement 12
COMMIT_BEHAVIOR, notes on 1231
comparison condition 65
comparison predicate 57
    data search using 58
compiling 842
    using multi-connection facility 859
component specification 89
compound statement 15
CONNECT privilege
    granting 8
    revoking 8
CONNECT statement 12
Connection class
    JDBC1.0 facility 1235
    Type2 JDBC driver 1314
connection handle
    allocating 13
    freeing 13
    getting 14
    to be used, cancelling all 13
    to be used, declaring 13
connection information priorities [Type4 JDBC driver] 1951
Connection interface [JDBC1.2 core API] 1370
connection pooling [JDBC2.0 Optional Package] 1262
connection security facility
    defining 6
    deleting 7
ConnectionPoolDataSource interface [JDBC2.0 Optional Package] 1860
conventions
    abbreviations for products vii
    diagrams xx
    fonts and symbols xxi
    KB, MB, GB and TB xxv
    version numbers xxv
COPY 14
COUNT 72
CREATE AUDIT 6
CREATE CONNECTION SECURITY 6
CREATE FUNCTION 6
CREATE INDEX 6
CREATE PROCEDURE 6
CREATE SCHEMA 6
CREATE TABLE 6
CREATE TRIGGER 6
CREATE TYPE 6
CREATE VIEW 6
CROSS JOIN [SQL optimization] 327
cursor
    allocating 9

closing 9
closure of 38
declaration and lock 216
declaring 9, 114, 122
definition of 36
deletion using 49
examples of using 223
opening 11, 36
receiving retrieval information for 10
retrieval using 32
retrieval without using 34
table operations using 212
updating row retrieved, using 33
updating using 43
usage of 32
use of 212
using multiple cursors simultaneously 215
cursor library, setting 1098
cursor updatability 212

D

data
deletion of 49
deletion of, with condition specified 50
duplicated, eliminating 74
extraction of 37
fetching 10
grouping 72
inserting 53
operation 69
processing 72
retrieval of 35
satisfying multiple conditions, searching for 62
sorted 73
sorting 73
specific data search 57
that uses preprocessable cursor, updating 12
updating 11, 43
updating with condition specified 44
data collecting servers, separating
SQL optimization 305
SQL optimization option 654
data dictionary table
details of 2166
examples of SQL statement retrieving 2163
list of 2159
retrieval of 2159
data guarantee level 368, 641
data guarantee level 0 369
data guarantee level 1 369
data guarantee level 2 370
data type 28
defining type 6
Type2 JDBC driver 1311
Type4 JDBC driver 1917
user-defined type, deleting 7
data type, added
JDBC2.0 basic facility 1250
JDBC2.1 Core API 1853
database operation 27
DatabaseMetaData class
JDBC1.0 facility 1242
Type2 JDBC driver 1319
DatabaseMetaData interface [JDBC1.2 core API] 1666
DataSource interface [JDBC2.0 Optional Package] 1855
DataSource object and JNDI, database connection using [Type4 JDBC driver] 1359
date data operation 70
DBA privilege
granting 8
revoking 8
DbType property 1190
deadlock
between servers 166
causes of 164
corrective measures for 164
countermeasures for 169
example of 165
in logical file used by plug-in, avoiding 171
preventing 170
DEALLOCATE PREPARE statement 9
DECIMAL signed normalized number, function for conversion to 2280
DECLARE CONNECTION HANDLE SET 13
DECLARE CONNECTION HANDLE UNSET 13

DECLARE CURSOR 9
definition information for user-defined type, receiving 10
DELETE statement 9
  WHERE clause of 50
DESCRIBE CURSOR statement 10
DESCRIBE statement 10
DESCRIBE TYPE statement 10
diagnostic information, getting 14
diagram conventions xx
DISCONNECT statement 12
distributed transaction [JDBC2.0 Optional Package] 1264
Driver class [JDBC1.0 facility] 1226
Driver interface [JDBC1.2 core API] 1363
DriverManager class, database connection using [Type4 JDBC driver] 1330
DROP AUDIT 7
DROP CONNECTION SECURITY 7
DROP DATA TYPE 7
DROP FUNCTION 7
DROP INDEX 7
DROP LIST statement 10
DROP PROCEDURE 7
DROP SCHEMA 7
DROP TABLE 8
DROP TRIGGER 8
DROP VIEW 8
dynamic SELECT statement 11, 123
dynamic SQL 123
  execution characteristics of 123
  notes on executing 124

E

embedded exception, declaring 14
embedded SQL
  declaring beginning of 13
  declaring end of 13
embedded SQL declare section
  dispensing with 833
  items that can be described within 714
embedded SQL UAP, overview of 710
embedded variable, declaration of 113
empty set 67
END DECLARE SECTION 13
environment definition information 574
environment setting
  Type2 JDBC driver 1224
  Type4 JDBC driver 1328
environment variable
  access path display utility 568
  access path information file for HiRDB SQL Tuning Advisor 568
  BES connection holding facility 572
  block transfer facility 570
  client using X/Open-compliant API in OLTP environment 562
  command execution from UAP 565
  inner replica facility 571
  inter-process memory communication facility 565
  JDBC driver 573
  lock control 568
  ODBC function 572
  output unit of UAP statistical information 568
  plug-in 572
  referential or check constraint 572
  setting 533
  space conversion for data 571
  specifying pointer as 834
  SQL-related 569
  system configuration 562
  system monitoring 566
  trouble-shooting 567
  updatable online reorganization 571
  user execution environment 563
  XDM/RD E2 connection facility 571
environment variable group 700
  registering 700
environment variable specification
  using UAP under OpenTP1 as client 545
  using UAP under TP1/EE as client 558
  using UAP under TP1/LiNK as client 549
  using UAP under TPBroker for C++ as client 551
  using UAP under TUXEDO as client 555

using UAP under WebLogic Server as client 556

error

automatic identification of 230

corrective measures for detected 229

identification of 117

re-reporting 16

reporting 15

error-handling process specification 114

EX 138

Exception trace log

methods acquired in 1971

notes 1992

output example and analysis method 1986

output format 1979

required memory size and file size 1991

setup for acquiring 1971

Type4 JDBC driver 1971

EXECUTE IMMEDIATE statement 10

EXECUTE statement 10

execution user, changing 13

EXISTS predicate 67

extended SQL error information, facility for output of 974

F

FALSE 63

FETCH statement 10, 37

floatable server allocation

optimization features related to 304

optimization features related to number of candidates of 305

SQL optimization 302

floatable server candidates, increasing number of

SQL optimization 305

SQL optimization option 651

floatable server, allocating [SQL optimization] 302

font conventions xxi

FOR READ ONLY clause 215

specifying 216

FOR statement 15

FOR UPDATE clause 215

specifying 216

forced nest-loop-join [SQL optimization option] 648

FREE CONNECTION HANDLE 13

FREE LOCATOR statement 10

function 89

defining 6

deleting 7

returning return value of 15

rules for determining called function 275

G

GB meaning xxv

GET CONNECTION HANDLE 14

GET DIAGNOSTICS 14

getBlockUpdate [Type2 JDBC driver] 1276, 1302

getClear_Env [Type2 JDBC driver] 1309

getCommit_Behavior [Type2 JDBC driver] 1301

getDBHostName

Type2 JDBC driver 1289

Type4 JDBC driver 1878

getDescription

Type2 JDBC driver 1288

Type4 JDBC driver 1876

getEncodeLang

Type2 JDBC driver 1291

Type4 JDBC driver 1904

getEnvironmentVariables [Type4 JDBC driver] 1902

getHiRDBCursorMode [Type4 JDBC driver] 1899

getJDBC_IF_TRC [Type4 JDBC driver] 1879

getLONGVARBINARY_Access

Type2 JDBC driver 1304

Type4 JDBC driver 1889

getLONGVARBINARY_AccessSize [Type4 JDBC driver] 1910

getLONGVARBINARY_TruncError [Type4 JDBC driver] 1911

getMaxBinarySize [Type4 JDBC driver] 1906

getNotErrorOccurred [Type4 JDBC driver] 1900

getPassword

Type2 JDBC driver 1293

Type4 JDBC driver 1885

getRMID [Type2 JDBC driver] 1297

getSQLInNum

Type2 JDBC driver 1305

Type4 JDBC driver 1891

getSQLOutNum

Type2 JDBC drover 1307
Type4 JDBC driver 1892
getSQLWarningLevel
Type2 JDBC driver 1308
Type4 JDBC driver 1894
getStatementCommitBehavior [Type4 JDBC driver] 1908
getTRC_NO [Type4 JDBC driver] 1880
getUapName [Type4 JDBC driver] 1882
getUser
Type2 JDBC driver 1292
Type4 JDBC driver 1883
getXACloseString
Type2 JDBC driver 1296
Type4 JDBC driver 1888
getXALocalCommitMode [Type4 JDBC driver] 1896
getXAOpenString
Type2 JDBC driver 1295
Type4 JDBC driver 1887
getXAThreadMode [Type2 JDBC driver] 1298
global deadlock 166
example of 167
GRANT access-privilege 8
GRANT AUDIT 8
GRANT CONNECT 8
GRANT DBA 8
GRANT RDAREA 8
GRANT SCHEMA 8
group
average value of 72
maximum value of 72
minimum value of 72
rows count of 72
total value of 72
group processing, ORDER BY processing, and DISTINCT set function processing at local back-end server
SQL optimization 310
SQL optimization option 653
grouping processing method [SQL optimization] 309
grouping processing, rapid
SQL optimization 310
SQL optimization option 654

H

hash execution [SQL optimization] 336, 343
hash function for table partitioning 2245
hash join
preparing for application of 347
SQL optimization 315
subquery hash execution [SQL extension optimizing option] 666
HASH JOIN [SQL optimization] 319
hash join processing method 320
batch hash join 320
bucket partitioning hash join 321
continuous hash join 321
intermittent hash join 322
HASH SUBQ [SQL optimization] 340, 345
hash table size 347
hashing mode 351
tuning 354
high-speed search condition
application scope of 355
deriving 355
deriving [SQL optimization option] 656
deriving by CNF conversion 363
deriving by condition shifting 364
HiRDB
data type of 1065
functions provided by 2245
HiRDB client
installing 486
no response status 618
organization of directory for 490
organization of file for 490
types of 484
HiRDB client environment variable, tool for registering 701
HiRDB OLE DB provider 1110
HiRDB.NET Data Provider 1120
data types of 1190
example of UAP using 1210
installing 1123
interfaces of 1142
list of classes provided by 1124
list of members provided by 1126
notes about 1186

prerequisite programs for 1120
type conversion by 1193
HiRDB/Developer's Kit 484
HiRDB/Run Time 484
HiRDB_PDHOST 563, 590
HiRDB_PDNAMEPORT 563, 591
HiRDB_PDTMID 563, 592
HiRDB_PDXAMODE 563, 592
HiRDBCommand 1142
list of members 1126
HiRDBCommandBuilder 1146
list of members 1127
HiRDBConnection 1152
list of members 1129
HiRDBDataAdapter 1157
list of members 1131
HiRDBDataReader 1159
list of members 1132
HiRDBException 1169
list of members 1135
HiRDBParameter 1169
list of members 1135
HiRDBParameterCollection 1174
list of members 1137
HiRDBProviderFactory 1181
HiRDBRowUpdatedEventArgs 1183
list of members 1140
HiRDBRowUpdatingEventArgs 1183
list of members 1140
HiRDBTransaction 1184
list of members 1141
HiRDBType property 1190
holdable cursor 219
example of using 225
hosts file, setting 532

I

I/O information, receiving 10
IF statement 15
IN predicate 59
data search using 60
index
benefits of using 234
changing during retrieval 235
defining 6
deleting 7
drawbacks of using 234
priority 234
processing time 234
index key value
creating locked resources for 210
non-locking of 175
INDEX SCAN [SQL optimization] 330
index scan [SQL optimization] 329
index use, suppressing [SQL optimization option] 654
index-type plug-in-dependent function 478
forcing use of multiple indexes 654
indicator variable, declaration of 113
INSERT statement 11, 53
specifying ROW in 54
INSTALL JAR 14
installing
Type2 JDBC driver 1224
Type4 JDBC driver 1328
integrity constraint 121
inter-process memory communication facility 616
interface 17
area 120
area type of 120
between UAP and HiRDB 17
use of 120

J

JAR file
creating 889
deleting 15
re-registering 15
registering 14
JAR file access facility [JDBC2.0 Optional Package] 1267
JAR format, archiving in 889
Java file
compiling 887
creating 887
Java program, coding 887
Java stored function 883
Java stored procedure 883

JDBC driver
functions provided by 1275
JDBC interface method trace
setup for acquiring 1968
Type4 JDBC driver 1968
JDBC1.0 facility 1226
JDBC1.2 core API 1363
JDBC2.0 basic facility 1244
JDBC2.0 Optional Package [Type2 JDBC driver] 1259, 1855
JDBC2.1 Core API 1846
join method [SQL optimization] 314
join method types [SQL optimization] 314
join process information 1000

K

KB meaning xxv
key conditions including scalar operation, applying [SQL optimization option] 657
KEY SCAN [SQL optimization] 331
key scan [SQL optimization] 329
KEY SCAN MERGE JOIN [SQL optimization] 316

L

L-KEY R-LIST MERGE JOIN [SQL optimization] 317
L-KEY R-SORT MERGE JOIN [SQL optimization] 317
L-LIST R-KEY MERGE JOIN [SQL optimization] 317
L-LIST R-SORT MERGE JOIN [SQL optimization] 317
L-SORT R-KEY MERGE JOIN [SQL optimization] 317
L-SORT R-LIST MERGE JOIN [SQL optimization] 317
labeling rule
C language 712
C++ 786
COBOL 750
OOCOBOL 787
LANG, character codes that can be specified for 798
LEAVE statement 15
LIKE predicate
data search using 61
linking 842
using multi-connection facility 859
list
creating 9
deleting 10
LIST SCAN [SQL optimization] 335
LIST SCAN MERGE JOIN [SQL optimization] 316
locator facility 448
locator, invalidating 10
lock
period 163
referencing during 163
releasing 163
starting 163
lock control on tables 12
lock mode 137
exclusive mode 138
protected retrieve mode 137
protected update mode 138
shared retrieve mode 138
shared update mode 138
simultaneous execution by two users based on 138
transition rule of 139
typical combination of 140, 143, 145, 148, 150, 153, 155, 157
LOCK statement 12
locked resource
inclusive relationships 136
setting minimum unit of 136
locking 136
based on deadlock priority value 170
page 145, 148
row 140, 143
row-level 139
units of 136
log collection mode 596
long running SQL statements, interval monitoring of 618

M

mapping
during retrieval data acquisition 1918

SQL data type 1917
when ? parameter is set 1921
MAX 72
maximum-hash-table-row-length 347
maximum-number-of-hash-joins-in-SELECT-statement 350
MB meaning xxv
merge join [SQL optimization] 314
MIN 72
MULTI COLUMNS INDEX SCAN [SQL optimization] 331
MULTI COLUMNS KEY SCAN [SQL optimization] 332
multi-connection facility 409

N

narrowed search 424
using inner replica facility 430
Native Runtime 2047
nest-loop-join, prioritized [SQL optimization option] 650
NESTED LOOPS JOIN [SQL optimization] 318
nested loops row value execution [SQL optimization] 342
NESTED LOOPS ROW VALUE SUBQ [SQL optimization] 344
nested loops work table execution [SQL optimization] 342
NESTED LOOPS WORK TABLE SUBQ [SQL optimization] 343
nested-loops-join [SQL optimization] 314
no-log mode 596
non-locking of index key values 150, 153
non-NULL data, searching for 61
nonblock mode 626
NOT 62
NOT NULL constraint 121
NULL predicate 61
with NOT, data search using 62

O

object relational database table 30
ODBC 3.5 driver
installing 1053
setting environment variable 1053
ODBC application program, HiRDB access from 1047
ODBC function
data type of 1065
facility available to 1067
provided by HiRDB 1060
ODBC2.0 driver, installing 1049
OLE DB 1110
OLE DB application program, HiRDB access from 1109
OPEN statement 11
operation code 959
optimizing mode 2 based on cost, application of [SQL extension optimizing option] 665
OR 62
OR multiple index usage [SQL optimization] 330
OR multiple index use, priority of [SQL optimization option] 652
OR PLURAL INDEXES SCAN [SQL optimization] 334
outer joining 76
overflow handling 1929

P

p_rdb_conv_space 2274
p_rdb_conv_space_utf8 2277
p_rdb_dbhash 2250
p_rdb_dec_sign_norm 2281
p_rdb_set_lang 2283
parameter trace output examples 966
PATH 533, 541
PDADDITIONALOPTLVL 569, 662
PDAGGR 569, 667
PDASTHOST 565, 613
PDASTPORT 565, 614
PDASTUSER 565, 615
PDAUTOCONNECT 569, 669
PDAUTORECONNECT 564, 610
PDBESCONHOLD 572, 688
PDBESCONHTI 572, 689
PDBINARYBLKF 570, 680
PDBLKBUFFSIZE 570, 680
PDBLKF 570, 679

pdcbl 811
PDCLTAPNAME 563, 593
PDCLTCNVMODE 564, 597
PDCLTGAIJIDLL 564, 603
PDCLTGAIJIFUNC 564, 604
PDCLTGRP 564, 607
PDCLTLANG 563, 594
PDCLTPATH 567, 627
PDCLTRCVADDR 562, 585
PDCLTRCVPORT 562, 583
PDCLTRDNODE 571, 686
PDCMDTRACE 565, 615
PDCMDWAITTIME 565, 615
PDCMMTBFDDL 569, 668
PDCNCTHDL-type variable, declaration of 14
PDCNSTRNTNAME 572, 688
PDCONNECTWAITTIME 566, 626
pdcpp 799
PDCURSORLVL 570, 674
PDCWAITTIME 566, 618
PDCWAITTIMEWRNPNT 566, 622
PDDBACCS 571, 684
PDDBBUFLRU 564, 612
PDDBLOG 564, 596
PDDBORGUAP 571, 685
PDDDLDEAPRP 672
PDDELRSVWDFILE 570, 677
PDDFLNVAL 569, 667
PDDLKPLIO 569
PDDLKPRIO 639
PDERRSKIPCODE 567, 628
PDEXWARN 564, 596
PDFESGRP 562, 580
PDFESHOST 562, 576
PDFORUPDATEEXLOCK 569, 640
PDGDATAOPT 572, 690
PDHASHTBLSIZE 569, 667
PDHATRNQUEUING 564, 613
PDHJHASHINGMODE 570, 678
PDHOST 562, 574
PDIPC 565, 616
PDISLLVL 569, 641
PDJDBFILEDIR 573, 694
PDJDBFILEOUTNUM 573, 694
PDJDBONMEMNUM 573, 694
PDJDBTRACELEVEL 573, 694
PDJETCOMPATIBLE 572, 693
PDKALVL 566, 623
PDKATIME 566, 625
PDLANG 563, 595
PDLOCKLIMIT 568, 639
PDLOCKSKIP 569, 640
PDNAMEPORT 562, 575
PDNBLOCKWAITTIME 566, 625
PDNODELAYACK 571, 682
pdocb 811
pdocc 799
PDODBCWRNSKIP 572, 692
PDODBESCAPE 572, 690
PDODBLOCATOR 572, 691
PDODBSPLITSIZE 572, 692
PDODBSTATCACHE 572, 689
PDPLGIXMK 572, 693
PDPLGPFSZ 573, 693
PDPLGPFSZEXP 573, 693
PDPLUGINNSUB 693
PDPLUGINNSUB2 572
PDPRMTRC 567, 629
PDPRMTRCSIZE 567, 630
PDPRPCRCLS 569, 669
PDRCCOUNT 564, 611
PDRCINTERVAL 564, 611
PDRCTRACE 567, 632
PDRDCLTCODE 571, 687
PDRECVMEMSIZE 565, 617
PDREPPATH 567, 631
PDSENDMEMSIZE 565, 617
PDSERVICEGRP 562, 578
PDSERVICEPORT 562, 579
PDSPACELVL 571, 685
PDSQLEXECTIME 567, 632
PDSQLOPTLVL 569, 642
PDSQLTEXTSIZE 567, 632
PDSQLTRACE 567, 627
PDSQLTRCOPENMODE 567, 632
PDSRVTYPE 562, 578
PDSTJTRNOUT 568, 638
PDSUBSTRLEN 564, 597

PDSWAITTIME 566, 619
PDSWATCHTIME 566, 619
PDSYSTEMID 565, 615
PDTAAPINFMODE 568, 638
PDTAAPINFPATH 568, 637
PDTAAPINFSIZE 568, 638
PDTCPCONOPT 564, 607
PDTIMEDOUTRETRY 566, 625
PDTMID 562, 586
PDTP1SERVICE 571, 687
pdtrcmgr 1021
PDTRCMODE 567, 630
PDTRCPATH 567, 631
PDTXACANUM 563, 587
PDUAPENVFILE 564, 611
PDUAPERLOG 567, 628
PDUAPEXERLOGPRMSZ 568, 635
PDUAPEXERLOGUSE 568, 634
PDUAPREPLVL 567, 630
PDUSER 563, 592
PDVWOPTMODE 568, 636
PDWRTLNCOMSZ 568, 634
PDWRTLNFILSZ 568, 633
PDWRTLNPATH 568, 633
PDXAMODE 563, 587
PDXARCVWTIME 563, 588
PDXATRCFILEMODE 563, 589
plug-in distribution function 477
PLUGIN INDEX SCAN [SQL optimization] 332
PLUGIN KEY SCAN [SQL optimization] 332
pointer, specifying as environment variable 834
PooledConnection interface [JDBC2.0 Optional Package] 1864
PR 137
preparable dynamic DELETE statement: locating 9
preparable dynamic UPDATE statement: locating 12
PREPARE statement 11
PreparedStatement class
    JDBC1.0 facility 1237
    Type2 JDBC driver 1316
PreparedStatement interface [JDBC1.2 core API] 1431
preprocessing 797
    for C programs in UNIX environment 798
    for C programs in Windows environment 816
    for COBOL programs in UNIX environment 809
    for COBOL programs in Windows environment 826
    overview of 797
preprocessor declaration statement, validating 831
priority [SQL optimization] 302
procedure 259
    calling 9
    defining 6
    deleting 7
process, avoiding survival of 619, 620
program example
    C language 719
    COBOL 755
PU 138
PURGE TABLE statement 11, 51

Q

quantified predicate 65
query process information 998
query processing method in HiRDB [SQL optimization] 302

R

R-LIST NESTED LOOPS JOIN [SQL optimization] 318
rapid grouping facility 406
RDAREA usage privilege
    granting 8
    revoking 8
reconnect trace facility 1022
relational database table 28
REMOVE JAR 15
REPLACE JAR 15
RESIGNAL statement 16
result set enhancement [JDBC2.0 basic facility] 1244
result set expansion [JDBC2.1 Core API] 1846
results-set return facility
    Java stored procedures only 922
    limited to SQL stored procedures 265
ResultSet class
    JDBC1.0 facility 1238

  Type2 JDBC driver 1317
ResultSet interface [JDBC1.2 core API] 1561
ResultSetMetaData class
  JDBC1.0 facility 1239
  Type2 JDBC driver 1318
ResultSetMetaData interface [JDBC1.2 core API] 1811
retrieval information, receiving 10
retrieve first n records facility 441
return code, referencing 227
RETURN statement 15
REVOKE access-privilege 9
REVOKE CONNECT 8
REVOKE DBA 8
REVOKE RDAREA 8
REVOKE SCHEMA 8
rollback
  setting 133
ROLLBACK statement 13
rollback, setting 131, 133
row 28
  deleting 9
  deleting all 11
  in table, deleting all 51
  inserting 11
  into table with repetition column, inserting 55
  on column basis, inserting 53
  on row basis, inserting 55
  on row basis, inserting (to table with FIX attribute) 54
  repeating execution of each 15
  retrieval on basis of 42
  retrieving one 11
  that uses preprocessable cursor, deleting 9
  updating on basis of 46
row identifier, searching using [SQL optimization] 335
row value execution [SQL optimization] 336
ROW VALUE SUBQ [SQL optimization] 340
ROWID FETCH [SQL optimization] 335

S

scalar function 70
schema
  defining 6
  deleting 7
schema definition privilege
  granting 8
  revoking 8
search method [SQL optimization] 328
SELECT statement
  FROM clause of 39
  selection clause of 41
SELECT-APSL [SQL optimization] 315, 327, 329, 333
set operation process information 996
SET SESSION AUTHORIZATION statement 13
setBlockUpdate [Type2 JDBC driver] 1275, 1301
setClear_Env [Type2 JDBC driver] 1309
setCommit_Behavior [Type2 JDBC driver] 1299
setDBHostName
  Type2 JDBC driver 1288
  Type4 JDBC driver 1877
setDescription
  Type2 JDBC driver 1286
  Type4 JDBC driver 1874
setEncodeLang
  Type2 JDBC driver 1289
  Type4 JDBC driver 1903
setEnvironmentVariables [Type4 JDBC driver] 1901
setHiRDBCursorMode [Type4 JDBC driver] 1898
setJDBC_IF_TRC [Type4 JDBC driver] 1878
setLONGVARBINARY_Access
  Type2 JDBC driver 1303
  Type4 JDBC driver 1888
setLONGVARBINARY_AccessSize [Type4 JDBC driver] 1909
setLONGVARBINARY_TruncError [Type4 JDBC driver] 1910
setMaxBinarySize [Type4 JDBC driver] 1905
setNotErrorOccurred [Type4 JDBC driver] 1899
setPassword
  Type2 JDBC driver 1293
  Type4 JDBC driver 1884
setRMID [Type2 JDBC driver] 1297
setSQLInNum
  Type2 JDBC driver 1304
  Type4 JDBC driver 1890

setSQLOutNum
  Type2 JDBC driver 1306
  Type4 JDBC driver 1891
setSQLWarningLevel
  Type2 JDBC driver 1307
  Type4 JDBC driver 1893
setStatementCommitBehavior [Type4 JDBC driver] 1907
setTRC_NO [Type4 JDBC driver] 1880
setUapName [Type4 JDBC driver] 1881
setUser
  Type2 JDBC driver 1291
  Type4 JDBC driver 1882
setXACloseString
  Type2 JDBC driver 1296
  Type4 JDBC driver 1887
setXALocalCommitMode [Type4 JDBC driver] 1894
setXAOpenString
  Type2 JDBC driver 1294
  Type4 JDBC driver 1885
setXAThreadMode [Type2 JDBC driver] 1298
SIGNAL statement 15
single row retrieval 34
single-row SELECT statement 11, 122
SORT MERGE JOIN [SQL optimization] 315
source program 3
space conversion function 2273
specific character pattern, searching for 60
SQL 3
  corrective measures for error of 227
  error identification of 227
  executing 10
  for retrieving data (execution statement) 115
  for updating data (execution statement) 115
  functional organization of 4
  information by 993
  preprocessing 10
  value provided at time of execution 124
SQL coding rule
  C language 712
  C++ 786
  COBOL 750
  OOCOBOL 787
SQL Communications Area 2060
  contents of 2061
  expanding 2066
  organization of 2060
SQL data type 2114, 2135
SQL Descriptor Area 2069
  contents of 2070
  data code set in 2076
  data length set in 2076
  example of 2083
  expanding 2080
  operation macro for 2087
  organization of 2069
  procedure for expanding 2086
SQL extension optimizing option 285, 662
SQL object
  re-creating for function 5
  re-creating for procedure 5
  re-creating for trigger 5
SQL objects, making multiple [SQL optimization option] 649
SQL optimization 285
  method types of 300
  option 285
  specification 285
  specifying 301
SQL optimizing mode 286
  optimizing mode 1 based on cost 286
  optimizing mode 2 based on cost 287
SQL prefix 14
SQL preprocessor
  activating 799, 811, 816, 826
  return code of (for C programs in UNIX environment) 809
  return code of (for C programs in Windows environment) 825
  return code of (for COBOL programs in UNIX environment) 815
  return code of (for COBOL programs in Windows environment) 831
  standard input and output of (for C programs in UNIX environment) 809
  standard input and output of (for C programs in Windows environment) 825

standard input and output of (for COBOL programs in UNIX environment) 815
standard input and output of (for COBOL programs in Windows environment) 831
SQL runtime interim results 995
SQL statement
description locations of 715
divisions in COBOL for describing 754
executing by conditional branching 15
executing multiple 15
for retrieval, examples of 2163
list of (control SQL) 12
list of (data manipulation SQL) 9
list of (definition SQL) 5
list of (embedded language) 13
list of (routine control SQL) 15
list of (usable in HiRDB) 4
preprocessed and executed by EXECUTE IMMEDIATE statement 125
preprocessed by PREPARE statement 125
preprocessed, releasing 9
preprocessing 11
repeating 15
SQL stored function
defining 269
executing 269
SQL stored procedure
debugging 263
defining 260
executing 260
SQL terminator 14
SQL trace dynamic acquisition facility 1020
SQL trace file 954
making backup of 966
SQL trace information
collecting 954
examining 956
SQL tracing 954
SQL-optimization-option 642
SQL_AUDITS table 2233
SQL_CHECK_COLUMNS table 2238
SQL_CHECKS table 2237
SQL_COLUMN_STATISTICS table 2212
SQL_COLUMNS table 2177
SQL_DATATYPE_DESCRIPTORS table 2216
SQL_DATATYPES table 2214
SQL_DIV_COLUMN table 2198
SQL_DIV_INDEX table 2197
SQL_DIV_TABLE table 2195
SQL_DIV_TYPE table 2239
SQL_EXCEPT table 2226
SQL_INDEX_COLINF table 2196
SQL_INDEX_DATATYPE table 2223
SQL_INDEX_FUNCTION table 2224
SQL_INDEX_RESOURCES table 2223
SQL_INDEX_STATISTICS table 2214
SQL_INDEX_TYPE_FUNCTION table 2225
SQL_INDEX_TYPES table 2222
SQL_INDEXES table 2189
SQL_IOS_GENERATIONS table 2227
SQL_KEYCOLUMN_USAGE table 2236
SQL_PARTKEY table 2232
SQL_PARTKEY_DIVISION table 2232
SQL_PHYSICAL_FILES table 2166
SQL_PLUGIN_ROUTINE_PARAMS table 2220
SQL_PLUGIN_ROUTINES table 2219
SQL_PLUGINS table 2218
SQL_RDAREA_PRIVILEGES table 2193
SQL_RDAREAS table 2166
SQL_REFERENTIAL_CONSTRAINTS table 2235
SQL_ROUTINE_PARAMS table 2208
SQL_ROUTINE_RESOURCES table 2206
SQL_ROUTINES table 2199
SQL_SYSPARAMS table 2240
SQL_TABLE_CONSTRAINTS table 2237
SQL_TABLE_PRIVILEGES table 2193
SQL_TABLE_RESOURCES table 2217
SQL_TABLE_STATISTICS table 2212
SQL_TABLES table 2168
SQL_TRIGGER_COLUMNS table 2229
SQL_TRIGGER_DEF_SOURCE table 2230
SQL_TRIGGER_USAGE table 2231
SQL_TRIGGERS table 2227
SQL_TYPE_RESOURCES table 2225
SQL_USERS table 2191
SQL_VIEW_TABLE_USAGE table 2194
SQL_VIEWS table 2195
SQLCODE variable 14

SQLException interface [JDBC1.2 core API] 1842
SQLJ 2007
SQLJ Runtime Library 2009
SQLJ Translator 2008, 2012
SQLSTATE variable 14
SQLWarning class [JDBC1.0 facility] 1242
SQLWarning interface [JDBC2.1 core API] 1843
SR 138
Statement class
    JDBC1.0 facility 1236
    Type2 JDBC driver 1315
Statement interface [JDBC1.2 core API] 1403
statement, leaving 15
static SQL 123
    execution characteristics of 123
stored function 269
    defining 269
stored procedure 259
structure, referencing 836
structured repetition predicate, searching for data using 63
SU 138
subqueries with external references, execution of [SQL optimization] 341
subqueries with no external references, execution of [SQL optimization] 335
subquery 64
    searching for data using 64
    using EXISTS predicate 67
    using quantified predicate 65
subquery hash execution, preparing for application of 347
SUM 72
symbol conventions xxi
synchronization point, setting 131, 133
system property, setting [Type2 JDBC driver] 1280

T

table
    basic configuration of 28
    deleting 8
    outer joining of 76
    procedure for deleting 49
    procedure for updating 43
    retrieval from multiple tables 39
    retrieval from single table 35
    retrieval from two tables 40
    using repetition columns 28
    with FIX attribute, retrieval of 41
    with FIX attribute, updating 45
    with repetition columns, updating 46
table data, retrieving dynamically 11
table definition, altering 5
TABLE SCAN [SQL optimization] 330
table scan [SQL optimization] 328
table with abstract data type
    deleting rows from 107
    inserting row into 108
    retrieving data from 105
    updating 106
target floatable server
    increasing (back-end servers for fetching data) [SQL optimization option] 649
    increasing (back-end servers for fetching data) [SQL optimization] 304
    limiting (back-end servers for fetching data) [SQL optimization option] 654
    limiting (back-end servers for fetching data) [SQL optimization] 305
TB meaning xxv
time data operation 70, 71
TIME, DATE, and TIMESTAMP columns, data conversion of 1926
total number of hits, facility for returning 453
trace acquisition command 1021
transaction
    cancelling 13
    control 131
    invalidation of 117
    moving 134
    startup of 131, 133
    terminating normally 12
    termination of 131
    validation of 117
trigger 283
    defining 6
    deleting 8
trigger operation search conditions 283

trigger SQL statement 283
trigger-activating SQL statement 283
TRUE 63
Type Name Descriptor Area 2097
  contents of 2097
  expanding 2098
  organization of 2097
Type2 JDBC driver 1223
  Array class 1326
Type4 JDBC driver 1327

U

UAP
  basic configuration of 710
  basic SQL configuration in 112
  characteristics of 3
  command execution from 1027
  configuration element of 710
  connecting to HiRDB 12
  connection to HiRDB system 131
  connection with HiRDB 114
  created with XDM/RD or UNIFY2000, converting 882
  creating 709
  data type and accessory used by 1192
  descriptive language of 119
  design for improving handling 233
  design for improving performance 233
  designing 111
  disconnecting from HiRDB 12
  disconnection from HiRDB 118
  disconnection from HiRDB system 131
  embedded format of 3, 119
  error recovery 1025
  execution procedure for 794
  extracting retrieved contents and storing them in 38
  format of 3
  in C++, writing 786
  in C, writing 712
  in OOCOBOL, writing 787
  notes on execution of 866
  operation environment of 18
  overview of 119
  preparation for execution of 793
  troubleshooting of 953
  using X/Open-based API (TX_function), executing 871
  written in C, executing 794
  written in COBOL, executing 795
UAP statistical report
  how to obtain 988
  interpreting 991
UAP statistical report facility 988
uniqueness constraint 121
UNKNOWN 63
unlocked conditional search 172
unsupported interfaces
  JDBC1.2 core API 1844
  JDBC2.0 Optional Package 1870
  JDBC2.1 Core API 1853
UPDATE statement 11
  SET clause of 45
  WHERE clause 44
update-SQL work table, suppressing creation of [SQL optimization option] 655

V

value
  assigning 12
  set in variables and SQL statement execution status 227
version number conventions xxv
view table 29
  defining 6, 81
  deleting 8
  manipulating 81, 87

W

WHENEVER 14
WHILE statement 15
window function 453
work table ATS execution [SQL optimization] 336
WORK TABLE ATS SUBQ [SQL optimization] 338
work table buffer size 350
work table execution [SQL optimization] 336
WORK TABLE SUBQ [SQL optimization] 339

work table, searching internally created [SQL optimization] 335
WRITE LINE statement 15

X

XAConnection interface [JDBC2.0 Optional Package] 1868
XADataSource interface [JDBC2.0 Optional Package] 1868
XAException interface [JDBC2.0 Optional Package] 1870
XAResource interface [JDBC2.0 Optional Package] 1869