
Scalable Database Server

HiRDB Version 8

UAP Development Guide Part I

3020-6-356(E)

HITACHI

■ Relevant program products

List of program products:

For the HP-UX 11.0, HP-UX 11i, or HP-UX 11i V2 (PA-RISC) operating system:

P-1B62-1182 HiRDB/Single Server Version 8 08-00
P-1B62-1382 HiRDB/Parallel Server Version 8 08-00
P-1B62-1582 HiRDB/Single Server Version 8 (64) 08-00
P-1B62-1782 HiRDB/Parallel Server Version 8 (64) 08-00
P-1B62-1B82 HiRDB/Run Time Version 8 08-00
P-1B62-1C82 HiRDB/Developer's Kit Version 8 08-00
P-1B62-1D82 HiRDB/Run Time Version 8(64) 08-00
P-1B62-1E82 HiRDB/Developer's Kit Version 8 (64) 08-00
P-F1B62-11823 HiRDB Staticizer Option Version 8 08-00
P-F1B62-11825 HiRDB Non Recover Front End Server Version 8 08-00
P-F1B62-11826 HiRDB Advanced High Availability Version 8 08-00
P-F1B62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

For the HP-UX 11i V2 (IPF) operating system:

P-1J62-1582 HiRDB/Single Server Version 8 (64) 08-00
P-1J62-1782 HiRDB/Parallel Server Version 8 (64) 08-00
P-1J62-1D82 HiRDB/Run Time Version 8 (64) 08-00
P-1J62-1E82 HiRDB/Developer's Kit Version 8(64) 08-00
P-F1J62-11823 HiRDB Staticizer Option Version 8 08-00
P-F1J62-11825 HiRDB Non Recover Front End Server Version 8 08-00
P-F1J62-11826 HiRDB Advanced High Availability Version 8 08-00
P-F1J62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

For the Solaris 8, 9, or 10 operating system:

P-9D62-1182 HiRDB/Single Server Version 8 08-00
P-9D62-1382 HiRDB/Parallel Server Version 8 08-00
P-9D62-1582 HiRDB/Single Server Version 8 (64) 08-00
P-9D62-1782 HiRDB/Parallel Server Version 8 (64) 08-00
P-9D62-1B82 HiRDB/Run Time Version 8 08-00
P-9D62-1C82 HiRDB/Developer's Kit Version 8 08-00
P-9D62-1D82 HiRDB/Run Time Version 8(64) 08-00
P-9D62-1E82 HiRDB/Developer's Kit Version 8(64) 08-00
P-F9D62-11823 HiRDB Staticizer Option Version 8 08-00
P-F9D62-11825 HiRDB Non Recover Front End Server Version 8 08-00
P-F9D62-11826 HiRDB Advanced High Availability Version 8 08-00
P-F9D62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

For the AIX(R) 5L V5.1, V5.2, or V5.3 operating system:

P-1M62-1182 HiRDB/Single Server Version 8 08-00
P-1M62-1382 HiRDB/Parallel Server Version 8 08-00
P-1M62-1582 HiRDB/Single Server Version 8 (64) 08-00
P-1M62-1782 HiRDB/Parallel Server Version 8 (64) 08-00
P-1M62-1B82 HiRDB/Run Time Version 8 08-00
P-1M62-1C82 HiRDB/Developer's Kit Version 8 08-00

P-1M62-1D82 HiRDB/Run Time Version 8(64) 08-00

P-1M62-1E82 HiRDB/Developer's Kit Version 8(64) 08-00

P-F1M62-11823 HiRDB Staticizer Option Version 8 08-00

P-F1M62-11825 HiRDB Non Recover Front End Server Version 8 08-00

P-F1M62-11826 HiRDB Advanced High Availability Version 8 08-00

P-F1M62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

For the Red Hat Linux 7.1, Red Hat Linux 7.2, Red Hat Enterprise Linux AS 2.1, Red Hat Enterprise Linux AS 3 (x86), Red Hat Enterprise Linux ES 3 (x86), Red Hat Enterprise Linux AS 4 (x86), Red Hat Enterprise Linux ES 4 (x86), Red Hat Enterprise Linux AS 3 (AMD64 & Intel EM64T),* Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T), or Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T) operating system:

P-9S62-1182 HiRDB/Single Server Version 8 08-00

P-9S62-1382 HiRDB/Parallel Server Version 8 08-00

P-9S62-1B82 HiRDB/Run Time Version 8 08-00

P-9S62-1C82 HiRDB/Developer's Kit Version 8 08-00

P-F9S62-11823 HiRDB Staticizer Option Version 8 08-00

P-F9S62-11825 HiRDB Non Recover Front End Server Version 8 08-00

P-F9S62-11826 HiRDB Advanced High Availability Version 8 08-00

P-F9S62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

* Only operating systems that run on the Intel EM64T are supported.

For the Red Hat Enterprise Linux AS 3 (AMD64 & Intel EM64T),* Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T), or Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T) operating system:

P-9W62-1182 HiRDB/Single Server Version 8 08-00

P-9W62-1382 HiRDB/Parallel Server Version 8 08-00

P-9W62-1B82 HiRDB/Run Time Version 8 08-00

P-9W62-1C82 HiRDB/Developer's Kit Version 8 08-00

* Only operating systems that run on the Intel EM64T are supported.

For the Red Hat Enterprise Linux AS 3 (IPF) or Red Hat Enterprise Linux AS 4 (IPF) operating system:

P-9V62-1182 HiRDB/Single Server Version 8 08-00

P-9V62-1382 HiRDB/Parallel Server Version 8 08-00

P-9V62-1B82 HiRDB/Run Time Version 8 08-00

P-9V62-1C82 HiRDB/Developer's Kit Version 8 08-00

P-F9V62-11823 HiRDB Staticizer Option Version 8 08-00

P-F9V62-11825 HiRDB Non Recover Front End Server Version 8 08-00

P-F9V62-11826 HiRDB Advanced High Availability Version 8 08-00

P-F9V62-11827 HiRDB Advanced Partitioning Option Version 8 08-00

For the Windows 2000, Windows XP Professional, Windows XP x64 Edition, Windows Server 2003, Windows Server 2003 x64 Edition, Windows Server 2003 R2, or Windows Server 2003 R2 x64 Edition operating system:

P-2462-7187 HiRDB/Single Server Version 8 08-00

P-2462-7387 HiRDB/Parallel Server Version 8 08-00

P-2462-7H87 HiRDB Non Recover Front End Server Version 8 08-00

P-2462-7J87 HiRDB Advanced High Availability Version 8 08-00

P-2462-7K87 HiRDB Advanced Partitioning Option Version 8 08-00

For the Windows XP x64 Edition or Windows Server 2003 x64 Edition operating system:

P-2962-7187 HiRDB/Single Server Version 8 08-00

P-2962-7387 HiRDB/Parallel Server Version 8 08-00

P-2962-1187 HiRDB/Run Time Version 8 08-00

P-2962-1287 HiRDB/Developer's Kit Version 8 08-00

For the Windows Server 2003 (IPF) operating system:

P-2862-7187 HiRDB/Single Server Version 8 08-00

P-2862-7387 HiRDB/Parallel Server Version 8 08-00

P-2862-1187 HiRDB/Run Time Version 8 08-00

P-2862-1287 HiRDB/Developer's Kit Version 8 08-00

P-2862-7H87 HiRDB Non Recover Front End Server Version 8 08-00

P-2862-7J87 HiRDB Advanced High Availability Version 8 08-00

P-2862-7K87 HiRDB Advanced Partitioning Option Version 8 08-00

For the Windows 2000, Windows XP, Windows XP x64 Edition, Windows Server 2003, or Windows Server 2003 x64 Edition operating system:

P-2662-1187 HiRDB/Run Time Version 8 08-00

P-2662-1287 HiRDB/Developer's Kit Version 8 08-00

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

ActiveX is a trademark of Microsoft Corp. in the U.S. and other countries.

AIX is a registered trademark of the International Business Machines Corp. in the U.S.

CORBA is a registered trademark of Object Management Group, Inc. in the United States.

DataStage, MetaBroker, MetaStage and QualityStage are trademarks of International Business Machines Corporation in the United States, other countries, or both.

DB2 is a registered trademark of the International Business Machines Corp. in the U.S.

HACMP/6000 is a trademark of the International Business Machines Corp. in the U.S.

HP-UX is a product name of Hewlett-Packard Company.

IBM is a registered trademark of the International Business Machines Corp. in the U.S.

Itanium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

JBuilder is a trademark of Borland Software Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Lotus, 1-2-3 are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft Access is a registered trademark of Microsoft Corporation in the U.S. and other countries.

Microsoft Excel is a product name of Microsoft Corp.

Microsoft is a registered trademark of Microsoft Corp. in the U.S. and other countries.

Motif is a registered trademark of the Open Software Foundation, Inc.

MS-DOS is a registered trademark of Microsoft Corp. in the U.S. and other countries.

ODBC is Microsoft's strategic interface for accessing databases.

OLE is the name of a software product developed by Microsoft Corporation and the acronym for Object Linking and Embedding.

ORACLE is a registered trademark of Oracle Corporation.

Oracle8i is a trademark of ORACLE Corporation.

Oracle9i is a trademark of ORACLE Corporation.

Oracle 10g is a trademark of ORACLE Corporation.

OS/390 is a trademark of the International Business Machines Corp. in the U.S.

POSIX stands for Portable Operating System Interface for Computer Environment, which is a set of standard specifications

published by the Institute of Electrical and Electronics Engineers, Inc.

RISC System/6000 is a registered trademark of the International Business Machines Corp. in the U.S.

Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

Sun is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

Sun Microsystems is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

The right to use the trademark DCE in Japan is sub-licensed from OSF.

UNIFY2000 is a product name of Unify Corp.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VERITAS is a trademark or registered trademark of Symantec Corporation in the U.S. and other countries.

Visual Basic is a registered trademark of Microsoft Corp. in the U.S. and other countries.

Visual C++ is a registered trademark of Microsoft Corp. in the U.S. and other countries.

Visual Studio is a registered trademark of Microsoft Corp. in the U.S. and other countries.

WebLogic is a registered trademark of BEA Systems, Inc.

Windows is a registered trademark of Microsoft Corp. in the U.S. and other countries.

Windows NT is a registered trademark of Microsoft Corp. in the U.S. and other countries.

Windows Server is a registered trademark of Microsoft Corp. in the U.S. and other countries.

X/Open is a registered trademark of X/Open Company Limited in the U.K. and other countries.

X Window System is a trademark of X Consortium, Inc.

The following program products include material copyrighted by Sun Microsystems, Inc.: P-9D62-1182, P-9D62-1382, P-9D62-1582, P-9D62-1782, P-9D62-1B82, P-9D62-1C82, P-9D62-1D82, P-9D62-1E82, P-F9D62-11823, P-F9D62-11825, P-F9D62-11826, and P-F9D62-11827.

The following program products include material copyrighted by UNIX System Laboratories, Inc.: P-9D62-1182, P-9D62-1382, P-9D62-1582, P-9D62-1782, P-9D62-1B82, P-9D62-1C82, P-9D62-1D82, P-9D62-1E82, P-F9D62-11823, P-F9D62-11825, P-F9D62-11826, and P-F9D62-11827.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ 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.

No part of this material may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in Japan.

■ Edition history

Edition 1 (3020-6-356(E)): March, 2007

■ Copyright

All Rights Reserved. Copyright (C) 2007, Hitachi, Ltd.

Preface

This manual describes the following items:

- Basic information needed to develop user application programs using SQL. HiRDB Scalable Database Server Version 8 uses SQL as a 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 8* ("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

Because this manual assumes knowledge of the information presented in *HiRDB Version 8 Description*, readers should read that manual first.

Organization of this manual

This manual consists of the following 16 chapters and 9 appendixes:

Chapter 1. *Overview*

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

Chapter 2. *Database Operations*

This chapter explains the data expressions used in a HiRDB database and the basic database operations.

Chapter 3. *UAP Design*

This chapter explains issues to be taken into consideration in designing a UAP.

Chapter 4. *UAP Design for Improving Performance and Handling*

This chapter describes issues that UAP designers should consider to improve UAP performance and usability.

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

This chapter describes notes about creating UAPs that access object relational databases.

Chapter 6. Client Environment Setup

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

Chapter 7. UAP Creation

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

Chapter 8. Preparation for UAP Execution

This chapter explains the flow from UAP preprocessing to execution and the methods used in those operations.

Chapter 9. Java Stored Procedures and Java Stored Functions

This chapter explains the development of stored procedures and stored functions with Java.

Chapter 10. UAP Troubleshooting

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

Chapter 11. Using a Distributed Database

This chapter explains the creation of a UAP that accesses a distributed database.

Chapter 12. Command Execution from UAPs

This chapter explains the execution of commands from UAPs.

Chapter 13. HiRDB Access from ODBC Application Programs

This chapter explains the ODBC driver installation procedure and ODBC functions.

Chapter 14. HiRDB Access from OLE DB Application Programs

This chapter explains HiRDB access from OLE DB application programs.

Chapter 15. HiRDB Access from ADO.NET-compatible Application Programs

This chapter describes how to access HiRDB from application programs that are compliant with ADO.NET.

Chapter 16. Type2 JDBC Driver

This chapter explains the Type2 JDBC driver installation and JDBC functions.

Chapter 17. *Type4 JDBC Driver*

This chapter explains the Type4 JDBC driver installation and JDBC functions.

Chapter 18. *SQLJ*

This chapter explains how to use SQLJ to develop a UAP.

Appendix A. *SQL Communications Area*

This appendix explains the organization and contents of the SQL Communications Area, as well as expansion of the SQL Communications Areas.

Appendix B. *SQL Descriptor Area*

This appendix explains the organization and contents of the SQL Descriptor Area, as well as expansion of the SQL Descriptor Area.

Appendix C. *Column Name Descriptor Area*

This appendix explains the organization and contents of the Column Name Descriptor Area, as well as expansion of the Column Name Descriptor Area.

Appendix D. *Type Name Descriptor Area*

This appendix explains the organization and contents of the Type Name Descriptor Area and expansion of the area.

Appendix E. *SQL Data Types and Data Descriptions*

This appendix 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.

Appendix F. *Data Dictionary Table Retrieval*

This appendix explains the contents of the data dictionary tables and how to reference them.

Appendix G. *Functions provided by HiRDB*

This appendix 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.

Appendix H. *Maximum and Minimum HiRDB Values*

This appendix explains the HiRDB maximum and minimum values.

Related publications

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

HiRDB (for Windows)

- *For Windows Systems HiRDB Version 8 Description* (3020-6-351(E))
- *For Windows Systems HiRDB Version 8 Installation and Design Guide* (3020-6-352(E))
- *For Windows Systems HiRDB Version 8 System Definition* (3020-6-353(E))
- *For Windows Systems HiRDB Version 8 System Operation Guide* (3020-6-354(E))
- *For Windows Systems HiRDB Version 8 Command Reference* (3020-6-355(E))

HiRDB (for UNIX)

- *For UNIX Systems HiRDB Version 8 Description* (3000-6-351(E))
- *For UNIX Systems HiRDB Version 8 Installation and Design Guide* (3000-6-352(E))
- *For UNIX Systems HiRDB Version 8 System Definition* (3000-6-353(E))
- *For UNIX Systems HiRDB Version 8 System Operation Guide* (3000-6-354(E))
- *For UNIX Systems HiRDB Version 8 Command Reference* (3000-6-355(E))
- *HiRDB Staticizer Option Version 7 Description and User's Guide* (3000-6-282(E))
- *For UNIX Systems HiRDB Version 8 Disaster Recovery System Configuration and Operation Guide* (3000-6-364)*

HiRDB (for UNIX and Windows)

- *HiRDB Version 8 SQL Reference* (3020-6-357(E))
- *HiRDB Version 8 Messages* (3020-6-358(E))
- *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide* (3020-6-360(E))
- *HiRDB Dataextractor Version 8 Description, User's Guide and Operator's Guide* (3020-6-362(E))

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

You must use the UNIX or the Windows manuals, as appropriate to the platform you are using.

Others

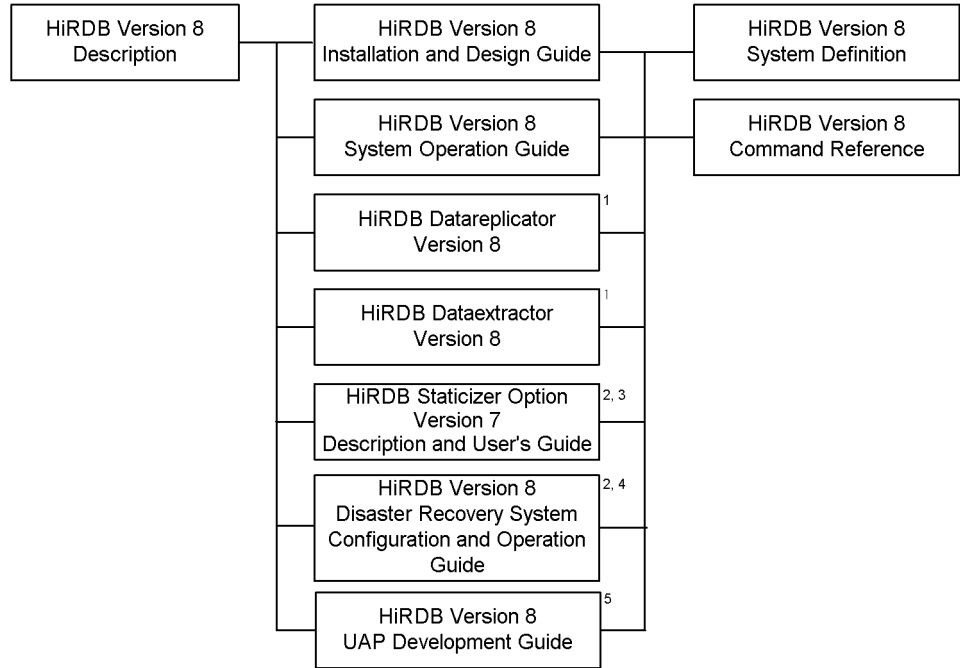
- *HiRDB External Data Access Version 7 Description and User's Guide* (3000-6-284(E))
- *Distributed Database System DF/UX* (3000-3-248(E))

- *COBOL85 Operations Guide* (3020-3-747(E))
- *OpenTP1 Version 6 System Definition* (3000-3-943(E))
- *OpenTP1 Version 6 Programming Reference C Language* (3000-3-945(E))
- *OpenTP1 Version 6 Programming Reference COBOL Language* (3000-3-946(E))
- *TP1/LINK USER'S GUIDE* (3000- 3-390(E))
- *TPBroker User's Guide* (3000-3-555(E))

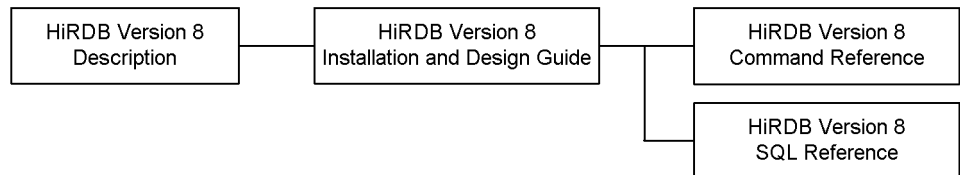
Organization of HiRDB manuals

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.

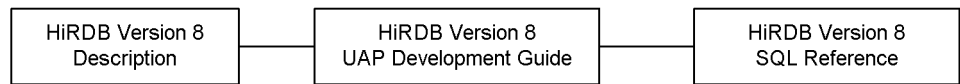
Manuals for system administrators:



Manuals for users who create tables:



Manuals for users who create or execute UAPs:



- ¹ Read if you use the replication facility to link data.
- ² Published for UNIX only. There is no corresponding Windows manual.
- ³ Read if you use the inner replica facility.
- ⁴ Read if you are configuring a disaster recovery system.
- ⁵ Must be read if you are linking HiRDB to an OLTP system.

Conventions: Abbreviations

Unless otherwise required, this manual uses the following abbreviations for product and other names.

Name of product or other entity	Representation	
HiRDB/Single Server Version 8	HiRDB/Single Server	HiRDB or HiRDB Server
HiRDB/Single Server Version 8(64)		
HiRDB/Parallel Server Version 8	HiRDB/Parallel Server	
HiRDB/Parallel Server Version 8(64)		
HiRDB/Developer's Kit Version 8	HiRDB/Developer's Kit	HiRDB Client
HiRDB/Developer's Kit Version 8(64)		
HiRDB/Run Time Version 8	HiRDB/Run Time	
HiRDB/Run Time Version 8(64)		
HiRDB Datareplicator Version 8	HiRDB Datareplicator	
HiRDB Dataextractor Version 8	HiRDB Dataextractor	
HiRDB Text Search Plug-in Version 7	HiRDB Text Search Plug-in	
HiRDB Spatial Search Plug-in Version 3	HiRDB Spatial Search Plug-in	
HiRDB Staticizer Option Version 8	HiRDB Staticizer Option	
HiRDB LDAP Option Version 8	HiRDB LDAP Option	
HiRDB Advanced Partitioning Option Version 8	HiRDB Advanced Partitioning Option	
HiRDB Advanced High Availability Version 8	HiRDB Advanced High Availability	
HiRDB Non Recover Front End Server Version 8	HiRDB Non Recover FES	
HiRDB Disaster Recovery Light Edition Version 8	HiRDB Disaster Recovery Light Edition	
HiRDB External Data Access Version 8	HiRDB External Data Access	
HiRDB External Data Access Adapter Version 8	HiRDB External Data Access Adapter	
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	

Name of product or other entity	Representation
Hitachi TrueCopy	TrueCopy
Hitachi TrueCopy basic	
TrueCopy	
TrueCopy remote replicator	
JP1/Automatic Job Management System 2	JP1/AJS2
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/DM	JP1/NETM/DM
JP1/NETM/DM Manager	
JP1/Performance Management	JP1/PFM
JP1/Performance Management Agent for HiRDB	JP1/PFM-Agent for HiRDB
JP1/Performance Management - Agent 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

Name of product or other entity	Representation	
Virtual-storage Operating System 3/Forefront System Product	VOS3/FS	VOS3
Virtual-storage Operating System 3/Leading System Product	VOS3/LS	
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 control XDM/DCCM3	XDM/DCCM3	
XDM/Relational Database XDM/RD	XDM/RD	XDM/RD
XDM/Relational Database Extended Version 2 XDM/RD E2	XDM/RD E2	
VOS3 Database Connection Server	DB Connection Server	
DB2 Universal Database for OS/390 Version 6	DB2	
DNCWARE ClusterPerfect (Linux Version)	ClusterPerfect	
Microsoft _(R) Excel	Microsoft Excel or Excel	
Microsoft _(R) Visual C++ _(R)	Visual C++ or C++	
Oracle 8i	ORACLE	
Oracle 9i		
Oracle 10g		
Sun Java™ System Directory Server	Sun Java System Directory Server or Directory Server	
HP-UX 11i V2 (IPF)	HP-UX or HP-UX (IPF)	
Red Hat Linux	Linux	
Red Hat Enterprise Linux		
Red Hat Enterprise Linux AS 3 (IPF)	Linux (IPF)	Linux
Red Hat Enterprise Linux AS 4 (IPF)		
Red Hat Enterprise Linux AS 3 (AMD64 & Intel EM64T)	Linux (EM64T)	
Red Hat Enterprise Linux AS 4 (AMD64 & Intel EM64T)		
Red Hat Enterprise Linux ES 4 (AMD64 & Intel EM64T)		
turbolinux 7 Server for AP8000	Linux for AP8000	

Name of product or other entity	Representation	
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	Windows 2000 or Windows 2000 Advanced Server	
Microsoft _(R) Windows Server TM 2003, Standard Edition	Windows Server 2003	
Microsoft _(R) Windows Server TM 2003, Enterprise Edition		
Microsoft _(R) Windows Server TM 2003 R2, Standard Edition	Windows Server 2003 R2 or Windows Server 2003	
Microsoft _(R) Windows Server TM 2003 R2, Enterprise Edition		
64 bit Version Microsoft _(R) Windows Server TM 2003, Enterprise Edition (IPF)	Windows Server 2003 (IPF) or Windows Server 2003	
Microsoft _(R) Windows Server TM 2003, Standard x64 Edition	Windows Server 2003 or Windows Server 2003 x64 Editions	Windows (x64)
Microsoft _(R) Windows Server TM 2003, Enterprise x64 Edition		
Microsoft _(R) Windows Server TM 2003 R2, Standard x64 Edition	Windows Server 2003, Windows Server 2003 R2 or Windows Server 2003 x64 Editions	
Microsoft _(R) Windows Server TM 2003 R2, Enterprise x64 Edition		
Microsoft _(R) Windows _(R) XP Professional x64 Edition	Windows XP or Windows XP x64 Edition	
Microsoft _(R) Windows _(R) XP Professional Operating System	Windows XP Professional	Windows XP
Microsoft _(R) Windows _(R) XP Home Edition Operating System	Windows XP Home Edition	
Single server	SDS	

Name of product or other entity	Representation
System manager	MGR
Front-end server	FES
Dictionary server	DS
Back-end server	BES

- Windows 2000, Windows XP, and Windows Server 2003 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 abbreviations:

Abbreviation	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
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

Abbreviation	Full name or meaning
CPU	Central Processing Unit
CSV	Comma Separated Values
DAO	Data Access Object
DAT	Digital Audio Taperecorder
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
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

Abbreviation	Full name or meaning
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	Jounaled File System
JFS2	Enhanced Jounaled 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
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

Abbreviation	Full name or meaning
MGR	System Manager
MIB	Management Information Base
MRCF	Multiple RAID Coupling Feature
MSCS	Microsoft Cluster Server
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
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

Abbreviation	Full name or meaning
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
SQL	Structured Query Language
SQL/K	Structured Query Language / VOS K
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

Abbreviation	Full name or meaning
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
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**.

3. 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

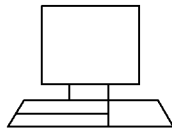
The OS log is referred to generically as *syslogfile*. *syslogfile* is the log output destination specified in `/etc/syslog.conf`. Typically, the following files are specified as *syslogfile*.

OS	File
HP-UX	<code>/var/adm/syslog/syslog.log</code>
Solaris	<code>/var/adm/messages</code> or <code>/var/log/syslog</code>
AIX 5L	<code>/var/adm/ras/syslog</code>
Linux	<code>/var/log/messages</code>

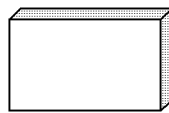
Conventions: Diagrams

This manual uses the following conventions in diagrams:

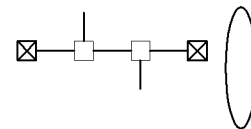
- Workstation or personal computer



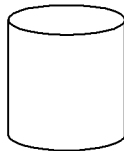
- Program



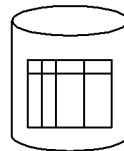
- Network (LAN)



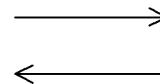
- File



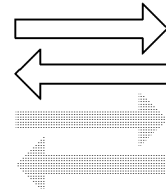
- Database



- Flow of control



- Flow of data



Conventions: Fonts and symbols

Font and symbol conventions are classified as:

- General font conventions

- Conventions in syntax explanations

These conventions are described below.

General font conventions

The following table lists the general font conventions:

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, bold is used in sentences such as the following: <ul style="list-style-type: none"> • From the File menu, choose Open. • Click the Cancel button. • In the Enter name entry box, type your name.
<i>Italics</i>	Italics are used to indicate a placeholder for some actual text provided by the user or system. Italics are also used for emphasis. For example: <ul style="list-style-type: none"> • Write the command as follows: <i>copy source-file target-file</i> • Do <i>not</i> 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: <ul style="list-style-type: none"> • At the prompt, enter <code>dir</code>. • Use the <code>send</code> command to send mail. • The following message is displayed: <code>The password is incorrect.</code>

Examples of coding and messages appear as follows (although there may be some exceptions, such as when coding is included in a diagram):

```
MakeDatabase
...
StoreDatabase temp DB32
```

In examples of coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity.

Font conventions in syntax explanations and examples

Conventions used in syntax explanations are explained as follows. When typing an actual command, omit the syntax conventions, attributes, and syntax elements described here.

Conventions in syntax explanations

Syntax definitions appear as follows:

```
StoreDatabase [temp|perm] (database-name ...)
```

The following table lists the conventions used in syntax explanations:

Example font or symbol	Convention
StoreDatabase	Code-font characters must be entered exactly as shown.
<i>database-name</i>	This font style marks a placeholder that indicates where appropriate characters are to be entered in an actual command.
SD	Bold code-font characters indicate the abbreviation for a command.
<u>perm</u>	Underlined characters indicate the default value.
[]	Square brackets enclose an item or set of items whose specification is optional.
	Only one of the options separated by a vertical bar can be specified at the same time.
...	An ellipsis (...) indicates that the item or items enclosed in () or [] immediately preceding the ellipsis may be specified as many times as necessary.
()	Parentheses indicate the range of items to which the vertical bar () or ellipsis (...) is applicable.
~	The tilde is followed by the attribute of a user-specified value.
<< >>	Double angle brackets enclose the default value that the system assumes when the specification is omitted.
< >	Angle brackets enclose the syntax element notation for a user-specified value.
(())	Double parentheses enclose the permitted range of user-specified values.

Syntax element conventions

Syntax element conventions explain the types of user-specified values.

Syntax element	Convention
<unsigned-integer>	Numeric characters
<unsigned-decimal> ¹	Numeric value (0-9), period (.), numeric value (0-9)
<identifier> ²	Alphanumeric character string beginning with an alphabetic character
<character-string>	String of any characters
<alphanumerics-and-special-characters>	The alphabetic characters (A-Z and a-z) and the special characters #, @, and \.
<symbolic-name>	Alphanumeric name beginning with an alphabetic character or a special character

Syntax element	Convention
<path-name> ³	Alphanumeric characters, backslashes (\) or forward slashes (/), and periods (.) In Windows, path names may include spaces and parentheses.

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

1

If the numeric value preceding the period is 0, it can be omitted. Similarly, if the numeric value following the period is 0, both the period and the 0 can be omitted.

2

An RDAREA name can begin with an alphabetic character or symbol, an alphanumeric, an underscore (_), or a space. However, 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 consist of alphabetic characters (A to Z, a to z), numeric characters, periods (.), hyphens (-), and underscores (_). Host names 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 (").

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 $\uparrow 34 \div 3 \uparrow$ is 12.
↓ ↓	Discard digits following the decimal point. Example: The result of $\downarrow 34 \div 3 \downarrow$ is 11.
MAX	Select the largest value as the result. Example: The result of $\text{Max}(10, 2 \times 4, 3 + 8)$ is 11.
MIN	Select the smallest value as the result. Example: The result of $\text{Min}(10, 2 \times 4, 3 + 8)$ 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.

Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is 1,024² bytes.
- 1 GB (gigabyte) is 1,024³ bytes.
- 1 TB (terabyte) is 1,024⁴ 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*.

Sources of the HiRDB Relational database language

The HiRDB relational database language described in this manual was developed by adding Hitachi's unique interpretations and specifications to the following standards. Hitachi expresses its appreciation to the developers and acknowledges the sources of these specifications.

HiRDB Relational Database

JIS X3005-1997 Database Language SQL

IS ISO9075-1992 Information processing systems - Database Language SQL

ANS X3.135-1986 Information systems - Database Language SQL

Relationships to ANSI Standard

The specifications for the HiRDB relational database language have been developed by adding Hitachi's unique interpretations to the specifications of ANS X3.135-1986 Information systems - Database Language SQL.

Hitachi has been granted ANSI's permission for the creation of this manual; however, ANSI is not responsible for this product or the contents of this manual.

Note

JIS: Japanese Industrial Standard

IS: International Standard

ANS: American National Standard

ANSI: American National Standards Institute

Acknowledgements

The COBOL language specifications were developed by CODASYL. The following statements acknowledges Hitachi's indebtedness to the developers, as requested by CODASYL. This acknowledgement restates a portion of the acknowledgement provided in the original specifications of COBOL, *CODASYL COBOL Journal of Development 1984*:

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose, is free to do so. However, all such organizations are requested to reproduce the following acknowledgement paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention *COBOL* in acknowledgement of the source, but need not quote the acknowledgement.

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein

FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC ® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

Note

The DB2 linkage facility was unavailable at the time of this publication because the English version of DF/UX Extension could not be supported in time for the document release.

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 (supported only in the Windows versions of HiRDB)
- Updatable online reorganization (supported only in the Windows versions of HiRDB)
- Sun Java System Directory Server linkage facility
- Simple setup tool

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

- HiRDB Control Manager

- HiRDB Disaster Recovery Light Edition
- HiRDB External Data Access
- HiRDB LDAP Option

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	iii
Organization of HiRDB manuals	v
Conventions: Abbreviations	vi
Path name representations	xvi
Log representations	xvi
Conventions: Diagrams	xvii
Conventions: Fonts and symbols	xvii
Font conventions in syntax explanations and examples	xviii
Notes on Windows path names	xx
Conventions: KB, MB, GB, and TB	xxi
Conventions: Version numbers	xxi
Sources of the HiRDB Relational database language	xxii
Relationships to ANSI Standard	xxii
Acknowledgements	xxii
Important notes on this manual	xxiii
Notes on printed manuals	xxiv
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	20
1.4 UAP operation environment	21
2. Database Operations	31
2.1 Database data expressions	32
2.1.1 Relational database tables	32
2.1.2 Object relational database tables	34
2.2 Cursor usage	36
2.3 Data retrieval	39
2.3.1 Retrieval from a single table	39
2.3.2 Retrieval from multiple tables	43
2.3.3 Retrieval of a table with FIX attribute	45
2.4 Data updating	47

2.4.1	Updating using a cursor	47
2.4.2	Updating with a condition specified	48
2.4.3	Updating a table with the FIX attribute	49
2.4.4	Updating a table with repetition columns	50
2.5	Data deletion	53
2.5.1	Deletion using a cursor	53
2.5.2	Deletion with a condition specified	54
2.5.3	Deleting all rows in a table	55
2.6	Data insertion	57
2.6.1	Inserting rows on a column basis	57
2.6.2	Inserting rows on a row basis (to a table with the FIX attribute)	58
2.6.3	Inserting rows into a table with repetition columns	59
2.7	Specific data search	61
2.7.1	Searching for data within a specified range of values	61
2.7.2	Searching for a specific character pattern	64
2.7.3	Searching for non-NULL data	65
2.7.4	Searching for data that satisfies multiple conditions	66
2.7.5	Searching for data using a Boolean predicate	67
2.7.6	Searching for data using a structured repetition predicate	67
2.7.7	Searching for data using a subquery	68
2.8	Data operations	73
2.8.1	Arithmetic operations on numeric data	73
2.8.2	Date and time data operations	74
2.9	Data processing	76
2.9.1	Data grouping	76
2.9.2	Data sorting	77
2.9.3	Duplicated data elimination	78
2.10	Outer joining of tables	80
2.11	Defining and manipulating a view table	85
2.12	Manipulating data in a table with abstract data types	93
2.12.1	Abstract data types provided by the HiRDB Text Search Plug-in	93
2.12.2	User-defined abstract data types	102

3. UAP Design 107

3.1	Basic SQL configuration in a UAP	108
3.2	Overview of UAPs	115
3.2.1	UAP descriptive languages	115
3.2.2	Interface areas	116
3.2.3	Integrity constraints	117
3.2.4	Retrieval methods using SQL statements	117
3.2.5	Static and dynamic SQLs	119
3.3	Transaction control	128
3.3.1	Connection to and disconnection from a HiRDB system	128
3.3.2	Transaction startup and termination	128

3.3.3	Synchronization point setting and rollback	128
3.3.4	UAP transaction management in an OLTP environment	129
3.3.5	Moving a transaction	131
3.4	Locking	133
3.4.1	Units of locking	133
3.4.2	Lock modes	134
3.4.3	Lock period	155
3.4.4	Deadlocks and corrective measures	156
3.4.5	Unlocked conditional search	163
3.4.6	Non-locking of index key values	165
3.4.7	Lock and suppression implementable with a UAP	168
3.4.8	Lock sequence based on SQL statement and index types	170
3.4.9	Creating locked resources for index key values	181
3.5	Use of a cursor	183
3.5.1	Notes on table operations when a cursor is used	183
3.5.2	FOR UPDATE and FOR READ ONLY clauses	186
3.5.3	Cursor declarations and locks	187
3.5.4	Holdable cursor	190
3.5.5	Examples of cursor use	194
3.6	SQL error identification and corrective measures	198
3.6.1	Error identification	198
3.6.2	Automatic error identification	201

4. UAP Design for Improving Performance and Handling 203

4.1	Using indexes	204
4.1.1	Indexes and processing time	204
4.1.2	Index priority	204
4.1.3	Changing indexes during retrieval	205
4.2	Manipulation of tables with the FIX attribute	206
4.3	Stored procedures and stored functions	207
4.3.1	Defining a stored procedure	207
4.3.2	Defining a stored function	217
4.3.3	Defining and deleting stored functions	228
4.4	Triggers	230
4.5	SQL optimization	232
4.5.1	SQL optimizing modes	233
4.5.2	Optimization method types	246
4.5.3	Specifying SQL optimization	248
4.5.4	Allocating floatable servers (HiRDB/Parallel Server only)	249
4.5.5	Grouping processing methods (HiRDB/Parallel Server only)	256
4.5.6	Join methods	261
4.5.7	Search Methods	276
4.5.8	Execution of subqueries with no external references	284
4.5.9	Execution of subqueries with external references	290

4.5.10	Preparing for application of hash join and subquery hash execution	296
4.5.11	Deriving high-speed search conditions.....	303
4.6	Data guarantee levels.....	314
4.6.1	Specifying the data guarantee level	314
4.6.2	Data guarantee level types	315
4.6.3	Example of search results when a data guarantee level is specified	316
4.7	Block transfer facility	319
4.8	Facilities using arrays	323
4.8.1	FETCH facility using arrays.....	323
4.8.2	INSERT facility using arrays.....	332
4.8.3	UPDATE facility using arrays	345
4.8.4	DELETE facility using arrays	348
4.9	Rapid grouping facility	352
4.9.1	Overview	352
4.9.2	Application criteria	352
4.9.3	Specification method	353
4.9.4	Tuning method.....	353
4.10	Multi-connection facility	355
4.11	Using tables for managing numbers.....	370
4.12	Narrowed search	379
4.12.1	What is a narrowed search?.....	379
4.12.2	Preparations for executing a narrowed search.....	379
4.12.3	Search using lists	380
4.12.4	Action if a rollback occurs for a transaction that uses a list.....	382
4.12.5	Automatic list deletion at HiRDB startup and termination	383
4.12.6	Notes about using lists.....	383
4.13	File output facility for BLOB data.....	386
4.13.1	What is the file output facility for BLOB data?	386
4.13.2	Application criteria	387
4.13.3	Specification method	388
4.13.4	Notes about using the file output facility for BLOB data.....	388
4.13.5	Examples of using the file output facility for BLOB data.....	388
4.14	Addition update and partial extraction facility for BLOB and BINARY data	391
4.14.1	What is the addition update and partial extraction facility for BLOB and BINARY data?	391
4.14.2	Examples of using the addition update and partial extraction facility for BLOB data.....	391
4.14.3	Notes about using the addition update and partial extraction facility for BLOB and BINARY data.....	393
4.15	Retrieve first n records facility	395
4.15.1	Overview	395
4.15.2	Notes.....	395
4.15.3	Checking the access path.....	396
4.16	Automatic reconnect facility.....	397

4.16.1	Application criteria.....	397
4.16.2	Reconnect timings.....	397
4.16.3	CONNECT processing during automatic reconnect.....	400
4.16.4	Notes about using the automatic reconnect facility.....	400
4.17	Locator facility.....	402
4.17.1	What is the locator facility?.....	402
4.17.2	Application standard.....	404
4.17.3	Usage method.....	404
4.17.4	Usage example.....	404
4.18	Facility for returning the total number of hits.....	407
4.18.1	Overview.....	407
4.18.2	Usage examples.....	407
4.18.3	Note.....	408
5.	Notes about Creating UAPs that Access Object Relational Databases	409
5.1	Using abstract data types and user-defined functions.....	410
5.2	Restrictions on functions provided by plug-ins.....	412
6.	Client Environment Setup	421
6.1	Types of HiRDB clients.....	422
6.2	Environment setup procedure for HiRDB clients.....	423
6.3	HiRDB client installation.....	424
6.3.1	Installing a HiRDB client on a UNIX client.....	424
6.3.2	Installing a HiRDB client on a Windows client.....	424
6.4	Organization of directories and files for a HiRDB client.....	427
6.4.1	Directories and files for UNIX clients.....	427
6.4.2	Directories and files for Windows clients.....	444
6.5	Setting the hosts file.....	458
6.6	Client environment definitions (setting environment variables).....	459
6.6.1	Environment setup format.....	459
6.6.2	Specifications for using a UAP under OLTP as the client.....	471
6.6.3	Client environment definitions.....	487
6.6.4	Environment definition information.....	498
6.6.5	Environment variables and connection types for HiRDB servers.....	607
6.6.6	Specifying client environment definitions for foreign table access.....	608
6.7	Registering an environment variable group.....	610
6.7.1	Registering an environment variable group in a UNIX environment.....	610
6.7.2	Registering an environment variable group in a Windows environment (registry registration).....	611
6.7.3	Registering an environment variable group in a Windows environment (file registration).....	618
7.	UAP Creation	619
7.1	Overview.....	620

7.1.1	UAP basic configuration	620
7.1.2	UAP configuration elements	620
7.2	Writing a UAP in C.....	622
7.2.1	Coding rules.....	622
7.2.2	Program example.....	628
7.3	Writing a UAP in COBOL.....	658
7.3.1	Coding rules.....	658
7.3.2	Program example.....	663
7.4	Writing a UAP in C++	694
7.4.1	Coding rules.....	694
7.5	Writing a UAP in OOCOBOL.....	695
7.5.1	Coding rules.....	695
8.	Preparation for UAP Execution	697
8.1	UAP execution procedure	698
8.1.1	Executing a UAP written in C	698
8.1.2	Executing a UAP written in COBOL	699
8.2	Preprocessing	701
8.2.1	Overview	701
8.2.2	Preprocessing in UNIX.....	702
8.2.3	Preprocessing in Windows	714
8.2.4	Validating preprocessor declaration statements.....	726
8.2.5	Dispensing with the embedded SQL declare section	727
8.2.6	Specifying pointers as environment variables	728
8.2.7	Referencing structures	731
8.2.8	Use of pointers, structures, and structure qualifiers when the -E2 or -E3 option of the preprocessor is specified	733
8.3	Compiling and linking	737
8.3.1	Libraries for compiling and linking.....	737
8.3.2	Compiling and linking in UNIX.....	743
8.3.3	Compiling and linking in Windows.....	750
8.3.4	Compiling and linking when the multi-connection facility is used.....	752
8.4	Notes on UAP execution.....	759
8.4.1	Executing UAPs that use an X/Open-based API (TX_function)	759
8.4.2	Creating UAPs that support the 64-bit mode.....	767
8.4.3	Converting UAPs created with XDM/RD or UNIFY2000	768
8.4.4	Notes on UAP execution	769
9.	Java Stored Procedures and Java Stored Functions	771
9.1	Overview.....	772
9.2	Procedure from Java stored routine creation to execution.....	775
9.2.1	Coding a Java stored routine	775
9.2.2	Registering the JAR file in HiRDB	778
9.2.3	Defining the Java stored routine.....	779

9.2.4	Executing the Java stored routine.....	779
9.3	Sample programs of Java stored routine.....	781
9.3.1	Sample program	781
9.3.2	Sample Java stored routines provided with HiRDB.....	785
9.4	Notes about Java program creation	806
9.4.1	Unsupported methods.....	806
9.4.2	Package, class, and method definitions.....	807
9.4.3	Parameter input/output mode mapping (Java stored procedures only)	808
9.4.4	Results-set return facility (Java stored procedures only)	809
9.4.5	Connection in a Java stored procedure.....	815
9.4.6	Releasing the result sets	815
9.5	Notes about testing and debugging.....	816
9.5.1	Java program for a Java stored procedure.....	816
9.5.2	Java program for a Java stored function.....	817
9.6	Notes about JAR file creation.....	819
9.6.1	Integrating Class files.....	820
9.6.2	Integrating Java files	820
10.	UAP Troubleshooting	823
10.1	Gathering error information.....	824
10.1.1	SQL tracing	824
10.1.2	Error logging	839
10.1.3	Facility for output of extended SQL error information.....	842
10.1.4	UAP statistical report facility.....	856
10.1.5	Command trace facility	885
10.1.6	SQL trace dynamic acquisition facility.....	887
10.1.7	Reconnect trace facility.....	889
10.1.8	HiRDB SQL Tuning Advisor access path information file	892
10.2	UAP error recovery	894
11.	Using a Distributed Database (Limited to HP-UX and AIX 5L)	897
11.1	Format of a distributed database.....	898
11.1.1	Accessing a distributed database and its relationship to RD-nodes	898
11.1.2	Relationship between a connection between RD-nodes and an SQL connection	898
11.1.3	Generating and terminating an SQL connection	899
11.1.4	Current SQL connection and database access.....	901
11.1.5	SQL connection and transaction control	902
11.2	Creating a UAP that accesses a remote database.....	904
11.2.1	Rules governing distributed clients and servers	904
11.2.2	Using the default SQL connection	905
11.2.3	Using an SQL connection to a distributed RD-node.....	907
11.3	Available SQL statements.....	909
11.3.1	SQL statements usable for remote database access.....	909

11.3.2	Details about available SQL statements	910
11.4	Available data types	922
11.4.1	Data types of variables usable in remote database access	922
11.4.2	Correspondence between distributed server data types and HiRDB data types	922
11.5	Handling distributed server errors.....	930
11.5.1	Return codes set by the distributed client	930
11.5.2	Obtaining and using detailed error information.....	930
11.6	Notes about using a distributed database	933
11.6.1	Notes about using a distributed client.....	933
11.6.2	Notes about using a distributed server.....	934
12.	Command Execution from UAPs	937
12.1	Overview.....	938
12.2	Preparations for executing commands from a UAP	939
12.3	Command executability	946
13.	HiRDB Access from ODBC Application Programs	953
13.1	ODBC application programs	954
13.2	Installing the ODBC2.0 driver.....	955
13.3	Installing the ODBC3.0 driver and setting the environment variables.....	959
13.3.1	Installation	959
13.3.2	Setting the environment variables	962
13.3.3	Determining the version number of the ODBC3.0 driver	962
13.4	ODBC functions provided by HiRDB	963
13.5	ODBC function data types and HiRDB data types.....	967
13.6	Asynchronous execution of ODBC functions	972
13.7	Setting cursor libraries	976
13.8	File DSNs.....	977
13.9	Executing a UAP in Unicode.....	978
13.10	Tuning and troubleshooting	981
13.11	Facilities that cannot be used when HiRDB is accessed with ODBC	982
14.	HiRDB Access from OLE DB Application Programs	983
14.1	Overview.....	984
14.2	Connection interface	985
14.2.1	Registry information.....	985
14.2.2	Connection properties.....	986
14.3	Schema information	988
14.4	Data type correspondences	990
14.5	Error handling procedures	991
14.5.1	Troubleshooting facility.....	991
14.6	Notes	992

15. HiRDB Access from ADO.NET-compatible Application Programs	993
15.1 Overview	994
15.1.1 HiRDB.NET Data Provider	994
15.1.2 Prerequisite programs for HiRDB.NET Data Provider	994
15.2 Installing HiRDB.NET Data Provider	995
15.2.1 Installation procedure	995
15.2.2 Files that are installed	995
15.2.3 Checking the version information	995
15.3 List of classes provided by HiRDB.NET Data Provider	996
15.4 List of members provided by HiRDB.NET Data Provider	997
15.4.1 List of HiRDBCommand members	997
15.4.2 List of HiRDBCommandBuilder members	998
15.4.3 List of HiRDBConnection members	998
15.4.4 List of HiRDBDataAdapter members	999
15.4.5 List of HiRDBDataReader members	1000
15.4.6 List of HiRDBException members	1001
15.4.7 List of HiRDBParameter members	1002
15.4.8 List of HiRDBParameterCollection members	1003
15.4.9 List of HiRDBRowUpdatedEventArgs members	1004
15.4.10 List of HiRDBRowUpdatingEventArgs members	1004
15.4.11 List of HiRDBTransaction members	1004
15.5 Interfaces of HiRDB.NET Data Provider	1006
15.5.1 HiRDBCommand	1006
15.5.2 HiRDBCommandBuilder	1010
15.5.3 HiRDBConnection	1011
15.5.4 HiRDBDataAdapter	1015
15.5.5 HiRDBDataReader	1016
15.5.6 HiRDBException	1025
15.5.7 HiRDBParameter	1026
15.5.8 HiRDBParameterCollection	1030
15.5.9 HiRDBRowUpdatedEventArgs	1035
15.5.10 HiRDBRowUpdatingEventArgs	1036
15.5.11 HiRDBTransaction	1036
15.6 Notes about HiRDB.NET Data Provider	1039
15.7 Data types of HiRDB.NET Data Provider	1041
15.7.1 DbType and HiRDBType properties	1041
15.7.2 Data types and accessories used by a UAP	1043
15.7.3 Type conversion by HiRDB.NET Data Provider	1044
15.8 Example of a UAP using HiRDB.NET Data Provider	1051
15.8.1 Connecting to the database	1051
15.8.2 Executing the SQL statement	1052
15.8.3 Executing a transaction	1053
15.8.4 Executing a search statement	1055
15.8.5 Executing the INSERT facility using arrays	1056

15.8.6	Executing a repetition column.....	1057
16.	Type2 JDBC Driver	1059
16.1	Installation and environment setup	1060
16.1.1	Installing	1060
16.1.2	Environment setup	1060
16.1.3	Abbreviation of methods	1061
16.2	JDBC1.0 facility	1062
16.2.1	Driver class	1062
16.2.2	Connection class	1071
16.2.3	Statement class.....	1072
16.2.4	PreparedStatement class	1073
16.2.5	CallableStatement class	1073
16.2.6	ResultSet class	1074
16.2.7	ResultSetMetaData class	1075
16.2.8	DatabaseMetaData class	1078
16.2.9	SQLWarning class.....	1078
16.3	JDBC2.0 basic facility	1080
16.3.1	Result set enhancements	1080
16.3.2	Batch updating	1082
16.3.3	Added data types	1086
16.4	JDBC2.0 Optional Package	1096
16.4.1	Database connection using DataSource and JNDI	1096
16.4.2	Connection pooling.....	1099
16.4.3	Distributed transactions	1101
16.5	JAR file access facility	1104
16.5.1	Class name	1104
16.5.2	Method name	1104
16.6	Array class	1107
16.7	Specifying a value when using a repetition column as the ? parameter	1109
16.8	Functions provided by the HiRDB JDBC driver	1112
16.8.1	Provided class	1112
16.8.2	setBlockUpdate.....	1112
16.8.3	getBlockUpdate	1113
16.9	Notes on using the BLOB type	1115
16.10	Setting system properties	1117
16.10.1	Setting the array facility	1117
16.10.2	Setting the maximum number of SQL search items or ? parameters	1118
16.11	Connection information setup/acquisition interface	1121
16.11.1	setDescription	1122
16.11.2	getDescription	1124
16.11.3	setDBHostName	1125
16.11.4	getDBHostName.....	1126
16.11.5	setEncodeLang.....	1126

16.11.6	getEncodeLang	1128
16.11.7	setUser	1128
16.11.8	getUser	1129
16.11.9	setPassword	1130
16.11.10	getPassword	1131
16.11.11	setXAOpenString	1131
16.11.12	getXAOpenString	1132
16.11.13	setXACloseString	1133
16.11.14	getXACloseString	1133
16.11.15	setRMID	1134
16.11.16	getRMID	1135
16.11.17	setXAThreadMode	1135
16.11.18	getXAThreadMode	1136
16.11.19	setCommit_Behavior	1136
16.11.20	getCommit_Behavior	1138
16.11.21	setBlockUpdate	1139
16.11.22	getBlockUpdate	1140
16.11.23	setLONGVARBINARY_Access	1141
16.11.24	getLONGVARBINARY_Access	1141
16.11.25	setSQLInNum	1142
16.11.26	getSQLInNum	1143
16.11.27	setSQLOutNum	1144
16.11.28	getSQLOutNum	1145
16.11.29	setSQLWarningLevel	1145
16.11.30	getSQLWarningLevel	1146
16.11.31	setClear_Env	1147
16.11.32	getClear_Env	1148
16.12	Data types and character codes	1149
16.12.1	Data types	1149
16.12.2	Character code conversion facility	1150
16.13	Classes and methods with limitations	1152
16.13.1	Driver class	1152
16.13.2	Connection class	1152
16.13.3	Statement class	1153
16.13.4	PreparedStatement class	1154
16.13.5	CallableStatement class	1154
16.13.6	ResultSet class	1155
16.13.7	ResultSetMetaData class	1156
16.13.8	DatabaseMetaData class	1157
16.13.9	Blob class	1164
16.13.10	Array class	1164
17.	Type4 JDBC Driver	1165
17.1	Installation and environment setup	1166

17.1.1	Installation	1166
17.1.2	Environment setup	1166
17.1.3	Abbreviation of methods	1167
17.2	Database connection using the DriverManager class	1168
17.2.1	Registering the Driver class	1168
17.2.2	Connecting to HiRDB with the getConnection method	1169
17.3	Database connection using a DataSource object and JNDI	1188
17.4	JDBC1.2 core API	1192
17.4.1	Driver interface	1192
17.4.2	Connection interface	1193
17.4.3	Statement interface	1199
17.4.4	PreparedStatement interface	1203
17.4.5	ResultSet interface	1208
17.4.6	DatabaseMetaData interface	1216
17.4.7	ResultSetMetaData interface	1222
17.4.8	Blob interface	1223
17.4.9	SQLException interface	1224
17.4.10	SQLWarning interface	1224
17.4.11	Unsupported interfaces	1226
17.5	JDBC2.1 Core API	1227
17.5.1	Expansion of the result set	1227
17.5.2	Batch update	1227
17.5.3	Added data types	1231
17.5.4	Unsupported interfaces	1231
17.6	JDBC2.0 Optional Package	1233
17.6.1	JNDI support	1233
17.6.2	Connection pool	1234
17.6.3	Distributed transactions	1236
17.6.4	Unsupported interfaces	1239
17.7	Connection information setup and acquisition interface	1240
17.7.1	setDescription	1242
17.7.2	getDescription	1245
17.7.3	setDBHostName	1245
17.7.4	getDBHostName	1246
17.7.5	setJDBC_IF_TRC	1246
17.7.6	getJDBC_IF_TRC	1247
17.7.7	setTRC_NO	1248
17.7.8	getTRC_NO	1249
17.7.9	setUapName	1249
17.7.10	getUapName	1250
17.7.11	setUser	1251
17.7.12	getUser	1252
17.7.13	setPassword	1252
17.7.14	getPassword	1253

17.7.15	setXAOpenString	1254
17.7.16	getXAOpenString	1255
17.7.17	setXACloseString	1256
17.7.18	getXACloseString	1256
17.7.19	setLONGVARBINARY_Access	1257
17.7.20	getLONGVARBINARY_Access	1258
17.7.21	setSQLInNum	1258
17.7.22	getSQLInNum	1259
17.7.23	setSQLOutNum	1260
17.7.24	getSQLOutNum	1261
17.7.25	setSQLWarningLevel	1261
17.7.26	getSQLWarningLevel	1262
17.7.27	setXALocalCommitMode	1263
17.7.28	getXALocalCommitMode	1264
17.7.29	setSQLWarningIgnore	1265
17.7.30	getSQLWarningIgnore	1265
17.7.31	setHiRDBCursorMode	1266
17.7.32	getHiRDBCursorMode	1267
17.7.33	setNotErrorOccurred	1268
17.7.34	getNotErrorOccurred	1269
17.7.35	setEnvironmentVariables	1269
17.7.36	getEnvironmentVariables	1270
17.7.37	setEncodeLang	1271
17.7.38	getEncodeLang	1273
17.7.39	setMaxBinarySize	1273
17.7.40	getMaxBinarySize	1275
17.7.41	setStatementCommitBehavior	1275
17.7.42	getStatementCommitBehavior	1276
17.7.43	setLONGVARBINARY_AccessSize	1277
17.7.44	getLONGVARBINARY_AccessSize	1278
17.7.45	setLONGVARBINARY_TruncError	1279
17.7.46	getLONGVARBINARY_TruncError	1280
17.8	Data types	1281
17.8.1	Mapping SQL data types	1281
17.8.2	Mapping during retrieval data acquisition	1282
17.8.3	Mapping when a ? parameter is set	1285
17.8.4	Data conversion of TIME, DATE, and TIMESTAMP columns	1290
17.8.5	Overflow handling	1294
17.9	Character conversion facility	1304
17.10	Supported client environment definitions	1305
17.11	Connection information priorities	1312
17.12	JDBC interface method trace	1318
17.12.1	Setup for trace acquisition	1318
17.12.2	Acquisition rules	1318

17.12.3	Output example.....	1319
17.13	Exception trace log	1321
17.13.1	Methods to be acquired and setup for log acquisition.....	1321
17.13.2	Output formats.....	1328
17.13.3	Output example and analysis method.....	1334
17.13.4	Required memory size and file size.....	1339
17.13.5	Notes.....	1340
18.	SQLJ	1343
18.1	Overview.....	1344
18.1.1	What is SQLJ?.....	1344
18.1.2	Environment settings.....	1346
18.2	SQLJ Translator.....	1348
18.3	UAP coding rule.....	1349
18.3.1	Labeling rule.....	1349
18.3.2	SQL coding rule.....	1349
18.3.3	SQL statements that can be used in SQLJ.....	1354
18.3.4	Correspondence between HiRDB data types and SQLJ data types.....	1357
18.3.5	Output variable settings (limited to the native interface version).....	1359
18.3.6	Using data types when a cursor is declared (limited to the native interface version).....	1361
18.3.7	Description of connection to and disconnection from a HiRDB server.....	1362
18.3.8	Description of cursor-based retrieval.....	1368
18.3.9	Receiving a dynamic result set.....	1372
18.3.10	Using JDBC and SQLJ together.....	1373
18.3.11	Creating and executing a UAP.....	1376
18.3.12	Migrating an SQLJ source from the standard interface version to the native interface version.....	1379
18.3.13	Notes about UAP development.....	1382
18.4	Native Runtime.....	1383
18.4.1	Package configuration.....	1383
18.4.2	Public classes of Native Runtime.....	1383
18.4.3	Cluster specifications.....	1384
18.4.4	Coding examples using the native interface.....	1390
Appendixes	1395
A.	SQL Communications Area.....	1396
A.1	Organization and contents of the SQL Communications Area.....	1396
A.2	Expanding the SQL Communications Area.....	1403
B.	SQL Descriptor Area.....	1406
B.1	Organization and contents of the SQL Descriptor Area.....	1406
B.2	Expanding the SQL Descriptor Area.....	1417
C.	Column Name Descriptor Area.....	1429
C.1	Organization and contents of the Column Name Descriptor Area.....	1429

C.2 Expanding the Column Name Descriptor Area	1431
D. Type Name Descriptor Area	1433
D.1 Organization of the Type Name Descriptor Area.....	1433
D.2 Contents of the Type Name Descriptor Area	1433
D.3 Expanding the Type Name Descriptor Area.....	1434
E. SQL Data Types and Data Descriptions.....	1436
E.1 SQL data types and C data descriptions	1436
E.2 SQL data types and COBOL data descriptions	1454
F. Data Dictionary Table Retrieval	1469
F.1 Examples of SQL statements for retrieval	1473
F.2 Data dictionary table details	1476
G. Functions provided by HiRDB.....	1553
G.1 Hash function for table partitioning.....	1553
G.2 Space conversion function	1578
G.3 Function for conversion to a DECIMAL signed normalized number	1585
G.4 Character code type specification function.....	1587
H. Maximum and Minimum HiRDB Values	1590

Index	1593
--------------------	-------------

List of figures

Figure 1-1: UAP development flow	2
Figure 1-2: SQL functional organization	4
Figure 1-3: Interface between a UAP and HiRDB.....	20
Figure 1-4: Operating mode using a machine other than the server machine as a client.....	22
Figure 1-5: Operating mode using the same server machine as the HiRDB server as the client	23
Figure 1-6: Operating mode using a UAP under OLTP as a client.....	24
Figure 1-7: Operating mode using an ODBC-compatible UAP as a client	25
Figure 1-8: Operating mode using an OLE DB-compatible UAP as a client	26
Figure 1-9: Operating mode using an ADO.NET-compatible UAP as a client	27
Figure 1-10: Operating mode using a Java (JDBC-compatible) application program as a client	28
Figure 1-11: Operating mode using a VOS3 system or Linux for AP8000 UAP as a client ...	29
Figure 2-1: Basic table configuration example	32
Figure 2-2: Configuration example of a table with repetition columns	33
Figure 2-3: Example of a base table and view table	34
Figure 2-4: Basic configuration example of a table with abstract data types	35
Figure 2-5: Retrieval from a single table	39
Figure 2-6: UAP data processing sequence for a retrieval results table	40
Figure 2-7: Cursor position immediately following cursor opening.....	41
Figure 2-8: Example of extracting retrieved contents and storing them in the UAP.....	42
Figure 2-9: Example of retrieval from two tables.....	44
Figure 2-10: Example of retrieval on a row basis	46
Figure 2-11: Procedure for updating a table.....	47
Figure 2-12: Example of using cursor to update a table	48
Figure 2-13: Example of updating with condition specified.....	49
Figure 2-14: Example of updating on a row basis	50
Figure 2-15: Example of updating a table with repetition columns.....	52
Figure 2-16: Procedure for deleting a table.....	53
Figure 2-17: Example of using a cursor to delete rows	54
Figure 2-18: Example of deletion with a condition specified	55
Figure 2-19: Example of deleting all rows in a table	56
Figure 2-20: Example of row insertion on a column basis	58
Figure 2-21: Example of row insertion on a row basis	59
Figure 2-22: Example of inserting a row into a table with repetition columns.....	60
Figure 2-23: Data search example using a comparison predicate.....	62
Figure 2-24: Data search example using a BETWEEN predicate	63
Figure 2-25: Data search example using an IN predicate	64
Figure 2-26: Data search example using a LIKE predicate	65
Figure 2-27: Data search example using a NULL predicate with NOT	66

Figure 2-28: Data search example involving multiple conditions.....	67
Figure 2-29: Data search example using a structured repetition predicate.....	68
Figure 2-30: Data search example using a subquery.....	69
Figure 2-31: Data search example using a subquery and a quantified predicate.....	70
Figure 2-32: Example of a subquery using the EXISTS predicate.....	71
Figure 2-33: Example of numeric data operations.....	74
Figure 2-34: Example of time data operation.....	75
Figure 2-35: Data grouping example.....	77
Figure 2-36: Data sorting example.....	78
Figure 2-37: Duplicated data elimination example.....	79
Figure 2-38: Example of outer joining.....	81
Figure 2-39: Example of outer joining with three or more tables.....	83
Figure 2-40: Tables used in examples of manipulating view tables.....	85
Figure 2-41: Example of defining a view table for limiting the columns to be searched.....	87
Figure 2-42: Example of using search conditions to define a view table.....	88
Figure 2-43: Example of defining a read-only view table.....	89
Figure 2-44: Example of defining a view table from which duplications are eliminated.....	90
Figure 2-45: Example of defining a view table from another view table.....	91
Figure 2-46: Example of manipulating a view table.....	92
Figure 2-47: Example of retrieval with a plug-in (1).....	94
Figure 2-48: Example of retrieval with a plug-in (2).....	95
Figure 2-49: Example of updating with a plug-in.....	97
Figure 2-50: Example of deletion with a plug-in.....	99
Figure 2-51: Example of insertion with a plug-in.....	101
Figure 2-52: Example of retrieval from a table with abstract data types.....	102
Figure 2-53: Example of updating a table with abstract data types.....	103
Figure 2-54: Example of deleting rows from a table with abstract data types.....	104
Figure 2-55: Example of inserting rows into a table with abstract data types.....	105
Figure 3-1: Basic SQL configuration in a UAP.....	108
Figure 3-2: Example of values provided at the time of SQL execution.....	120
Figure 3-3: Dynamic SQL execution mode.....	121
Figure 3-4: Example of inserting data into a dynamically specified table.....	125
Figure 3-5: Example of dynamic processing when the preprocessed SQL is a dynamic SELECT statement.....	126
Figure 3-6: Examples of transaction startup and termination.....	128
Figure 3-7: Locked resources and inclusive relationships.....	133
Figure 3-8: Example of deadlock.....	157
Figure 3-9: Example of deadlock in page-locking.....	158
Figure 3-10: Example of global deadlock.....	159
Figure 3-11: Processing flows of an ordinary retrieval and of a retrieval using an unlocked conditional search.....	164
Figure 3-12: Example of deadlock avoidance by applying non-locking of index key values.....	167
Figure 3-13: Creation of a key value locked resource when pd_key_resource_type=TYPE1 is used.....	181

Figure 3-14: Creation of a key value locked resource when pd_key_source_type=TYPE2 is used.....	182
Figure 4-1: Benefits of using an SQL stored procedure	208
Figure 4-2: Defining and executing an SQL stored procedure	209
Figure 4-3: Example of an SQL stored procedure	210
Figure 4-4: Overview of results-set return facility (for SQL stored procedures)	214
Figure 4-5: Defining and executing an SQL stored function	218
Figure 4-6: SQL stored function example.....	218
Figure 4-7: Correspondences between a table with abstract data types and the called function.....	225
Figure 4-8: Trigger overview	230
Figure 4-9: SQL statement query processing in a HiRDB/Parallel Server.....	250
Figure 4-10: Floatable server allocation when the optimization method is omitted.....	253
Figure 4-11: Floatable server allocation when increasing the target floatable servers (back-end servers for fetching data) is applied.....	254
Figure 4-12: Floatable server allocation when limiting the target floatable servers (back-end servers for fetching data) is applied.....	255
Figure 4-13: Floatable server allocation when separating data collecting servers is applied	256
Figure 4-14: Grouping processing method when the optimization method is omitted.....	258
Figure 4-15: Grouping processing when rapid grouping processing is applied.....	259
Figure 4-16: Grouping processing when group processing, ORDER BY processing, and DISTINCT set function processing are applied at the local back-end server	260
Figure 4-17: Processing of SORT MERGE JOIN.....	263
Figure 4-18: Processing of KEY SCAN MERGE JOIN.....	263
Figure 4-19: Processing of LIST SCAN MERGE JOIN	264
Figure 4-20: Processing of NESTED LOOPS JOIN.....	265
Figure 4-21: Processing of R-LIST NESTED LOOPS JOIN	266
Figure 4-22: Processing of HASH JOIN.....	267
Figure 4-23: Processing method of batch hash join	269
Figure 4-24: Processing method of bucket partitioning hash join	271
Figure 4-25: Processing method of continuous hash join	272
Figure 4-26: Processing method of intermittent hash join	273
Figure 4-27: Processing method of DISTRIBUTED NESTED LOOPS JOIN	275
Figure 4-28: CROSS JOIN processing method	275
Figure 4-29: TABLE SCAN processing method.....	278
Figure 4-30: INDEX SCAN processing method.....	278
Figure 4-31: KEY SCAN processing method.....	279
Figure 4-32: MULTI COLUMNS INDEX SCAN processing method.....	279
Figure 4-33: MULTI COLUMNS KEY SCAN processing method	279
Figure 4-34: PLUGIN INDEX SCAN processing method.....	280
Figure 4-35: PLUGIN KEY SCAN processing method	280
Figure 4-36: AND PLURAL INDEXES SCAN processing method.....	281
Figure 4-37: OR PLURAL INDEXES SCAN processing method.....	282
Figure 4-38: LIST SCAN processing method.....	282

Figure 4-39: ROWID FETCH processing method	283
Figure 4-40: FOREIGN SERVER SCAN processing method	283
Figure 4-41: FOREIGN SERVER LIMIT SCAN processing method	284
Figure 4-42: WORK TABLE ATS SUBQ processing method	287
Figure 4-43: WORK TABLE SUBQ processing method	288
Figure 4-44: ROW VALUE SUBQ processing method	289
Figure 4-45: HASH SUBQ processing method	290
Figure 4-46: NESTED LOOPS WORK TABLE SUBQ processing method	293
Figure 4-47: NESTED LOOPS ROW VALUE SUBQ processing method	294
Figure 4-48: HASH SUBQ processing method	295
Figure 4-49: Data guarantee range of data guarantee level 0	315
Figure 4-50: Data guarantee range of data guarantee level 1	315
Figure 4-51: Data guarantee range of data guarantee level 2	316
Figure 4-52: Example of search results when a data guarantee level is specified	317
Figure 4-53: Overview of block transfer facility	319
Figure 4-54: Overview of multi-connection facility processing (when multithreading is not used)	356
Figure 4-55: Overview of multi-connection facility processing (when multithreading is used)	357
Figure 4-56: Overview of multi-connection facility processing (when a connection is shared by multiple threads)	358
Figure 4-57: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a single-thread OLTP system)	359
Figure 4-58: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a multi-thread OLTP system)	360
Figure 4-59: Coding example (C) of a UAP that uses the multi-connection facility	362
Figure 4-60: Coding example (COBOL) of a UAP that uses the multi-connection facility	364
Figure 4-61: Coding example (C) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP	366
Figure 4-62: Coding example (COBOL) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP	367
Figure 4-63: Example of a table that manages numbers	371
Figure 4-64: Example of a search that uses lists	381
Figure 4-65: Overview of the file output facility for BLOB data	387
Figure 4-66: 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)	398
Figure 4-67: Reconnect timing (when the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement)	399
Figure 4-68: Reconnect timing (when the HiRDB client executes the CONNECT statement)	400
Figure 4-69: Overview of the locator facility	403
Figure 6-1: Differences between fixing and not fixing the communication-target server	484

Figure 6-2: Relationships among PDCWAITTIME, PDSWAITTIME, and PDSWATCHTIME	542
Figure 6-3: Overview of processing for each setting of PDCURSORLVL.....	592
Figure 7-1: Example of the basic configuration of an embedded SQL UAP.....	620
Figure 7-2: Flowchart example of an embedded SQL UAP written in C.....	629
Figure 7-3: Flowchart example of an embedded SQL UAP written in C.....	630
Figure 7-4: PAD chart for program example 2 (1/4).....	636
Figure 7-5: PAD chart for program example 2 (2/4).....	637
Figure 7-6: PAD chart for program example 2 (3/4).....	638
Figure 7-7: PAD chart for program example 2 (4/4).....	639
Figure 7-8: PAD chart for program example 3 (1/3).....	648
Figure 7-9: PAD chart for program example 3 (2/3).....	649
Figure 7-10: PAD chart for program example 3 (3/3).....	650
Figure 7-11: Flowchart of program example 4 (1/3).....	664
Figure 7-12: Flowchart of program example 4 (2/3).....	665
Figure 7-13: Flowchart of program example 4 (3/3).....	666
Figure 7-14: PAD chart for program example 5 (1/4).....	679
Figure 7-15: PAD chart for program example 5 (2/4).....	680
Figure 7-16: PAD chart for program example 5 (3/4).....	682
Figure 7-17: PAD chart for program example 5 (4/4).....	683
Figure 8-1: Execution procedure for UAP written in C.....	699
Figure 8-2: Execution procedure for a UAP written in COBOL.....	700
Figure 9-1: Procedure from Java stored routine creation to execution.....	773
Figure 9-2: Example of Java program coding.....	775
Figure 9-3: Example of compilation.....	776
Figure 9-4: Overview of testing and debugging.....	777
Figure 9-5: Example of archiving in the JAR format.....	777
Figure 9-6: Overview of JAR file registration.....	778
Figure 9-7: Example of a Java stored routine definition.....	779
Figure 9-8: Example of Java stored routine execution.....	780
Figure 9-9: Method execution control using security policy.....	807
Figure 9-10: Example of parameter input/output mod mapping.....	809
Figure 9-11: Overview of the results-set return facility (for a Java stored procedure).....	810
Figure 9-12: Procedure for testing and debugging a Java program for a Java stored procedure.....	817
Figure 9-13: Procedure for testing and debugging a Java program for a Java stored function.....	818
Figure 9-14: Location at which class files are created.....	819
Figure 9-15: Example of integrating Class files in JAR files.....	820
Figure 11-1: Distributed database connection format.....	899
Figure 11-2: Examples of transaction startup and termination using an SQL connection to a distributed RD-node.....	903
Figure 12-1: Overview of command execution from UAPs.....	938
Figure 12-2: Sample server-client configuration for a HiRDB/Single Server.....	939

Figure 12-3: Sample server-client configuration for a HiRDB/Parallel Server.....	942
Figure 17-1: Flow of mutual character code conversion between HiRDB character data and Unicode	1304
Figure 18-1: Flow of UAP development that uses SQLJ	1344
Figure 18-2: Execution of a UAP that uses SQLJ	1345
Figure A-1: Configuration of SQL Communications Area	1397
Figure B-1: Organization of the SQL Descriptor Area.....	1406
Figure C-1: Organization of the Column Name Descriptor Area.....	1429
Figure D-1: Organization of the Type Name Descriptor Area.....	1433

List of tables

Table 1-1: List of SQL statements (definition SQL).....	5
Table 1-2: List of SQL statements (data manipulation SQL).....	10
Table 1-3: List of SQL statements (control SQL).....	14
Table 1-4: List of SQL statements (embedded language).....	15
Table 1-5: List of SQL statements (routine control SQL).....	18
Table 2-1: Descriptions of abstract data type functions provided by the HiRDB Text Search Plug-in	93
Table 3-1: UAP descriptive languages.....	115
Table 3-2: Interface area types and uses.....	116
Table 3-3: Classification of UAP retrieval methods using SQL statements	118
Table 3-4: Execution characteristics of static and dynamic SQLs	119
Table 3-5: SQL statements preprocessed by the PREPARE statement, and SQL statements preprocessed and executed by the EXECUTE IMMEDIATE statement	121
Table 3-6: Synchronization points and transactions.....	129
Table 3-7: Scope of the LOCK TABLE UNTIL DISCONNECT specification when OpenTP1 is used	132
Table 3-8: Simultaneous execution by two users based on lock modes.....	135
Table 3-9: Lock mode transition rules.....	136
Table 3-10: Typical lock mode combinations (row locking) (1/2).....	137
Table 3-11: Typical lock mode combinations (row locking) (2/2).....	139
Table 3-12: Typical lock mode combinations (page locking) (1/2)	141
Table 3-13: Typical lock mode combinations (page locking) (2/2)	143
Table 3-14: Typical lock mode combinations (non-locking of index key values) (1/2)	144
Table 3-15: Typical lock mode combinations (non-locking of index key values) (2/2)	147
Table 3-16: Typical lock mode combinations (when check pending status is set) (1/2).....	149
Table 3-17: Typical lock mode combinations (when check pending status is set) (2/2).....	150
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)	151
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)	152
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)	153
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)	154
Table 3-22: Deadlocks and their countermeasures.....	160
Table 3-23: Relationships between cursor updatability and operations that do not use a cursor	183
Table 3-24: Specifying FOR UPDATE and FOR READ ONLY clauses	187

Table 3-25: Relationships between the lock option specified during cursor declaration or dynamic SELECT statement preprocessing and the lock option specified during table operations	188
Table 3-26: Values set in variables and SQL statement execution status	198
Table 3-27: Relationship among SQLSTATE, SQLCODE, and SQLWARN0 values when normal termination with a warning occurs.....	199
Table 3-28: Additional return code information and items referred to by the information	200
Table 4-1: Priorities of pre-defined data types.....	223
Table 4-2: Priorities of abstract data types.....	224
Table 4-3: Features of the SQL optimizing modes	233
Table 4-4: SQL optimizing modes in which the SQL optimization option and SQL extension optimizing option are valid	245
Table 4-5: Optimization features related to floatable server allocation.....	251
Table 4-6: Optimization features related to number of floatable server allocation candidates	252
Table 4-7: Optimization features related to grouping processing methods	257
Table 4-8: Join method types and features.....	261
Table 4-9: Hash join processing methods and features.....	267
Table 4-10: Search method types and features	276
Table 4-11: Execution methods and features of subqueries with no external references	284
Table 4-12: Optimal execution method of subqueries with no external references.....	286
Table 4-13: Execution methods and features of subqueries with external references	291
Table 4-14: Tuning methods for hash table size	301
Table 4-15: Tuning information for the hashing mode	303
Table 4-16: Relationships between the SQL optimization and SQL extension optimizing options and deriving high-speed search conditions	305
Table 4-17: Relationships between the SQL optimization options and deriving high-speed search conditions	309
Table 4-18: Relationship between data guarantee level and lock option.....	314
Table 5-1: Types of plug-in distribution functions	412
Table 5-2: Correspondences between receive and send functions for passing inter-function values.....	413
Table 5-3: Combinations that trigger an error when a plug-in distribution function is executed.....	416
Table 5-4: Passing inter-function values in set operation results.....	420
Table 6-1: Files and directories for workstation - HiRDB/Developer's Kit	427
Table 6-2: Files and directories for HiRDB/Run Time (UNIX client)	431
Table 6-3: Files and directories for HiRDB/Developer's Kit (UNIX client in IPF machine).	433
Table 6-4: Files and directories for HiRDB/Run Time (UNIX client in IPF machine).....	437
Table 6-5: Files and directories for HiRDB/Developer's Kit (Linux (EM64T))	439
Table 6-6: Files and directories for HiRDB/Run Time (Linux (EM64T))	442
Table 6-7: Archived files used for each purpose (UNIX client).....	442
Table 6-8: Shared library files used for each purpose (UNIX client).....	443
Table 6-9: Library files used by each transaction manager (UNIX client).....	444

Table 6-10: Files and directories for HiRDB/Developer's Kit (Windows client)	445
Table 6-11: Files and directories for HiRDB/Run Time (Windows client).....	447
Table 6-12: Files and directories for HiRDB/Developer's Kit (Windows client in IPF machine)	449
Table 6-13: Files and directories for HiRDB/Run Time (Windows client in IPF machine)..	451
Table 6-14: Files and directories for HiRDB/Developer's Kit (EM64T machine Windows client)	452
Table 6-15: Files and directories for HiRDB/Run Time (EM64T machine Windows client)	454
Table 6-16: Files and directories for ODBC driver (Windows client)	456
Table 6-17: Linkage library files used according to purpose (Windows client)	456
Table 6-18: Library files used by each transaction manager (Windows client).....	457
Table 6-19: List of libraries and compilers (Windows client).....	457
Table 6-20: OpenTP1 definitions in which the environment variables are specified	471
Table 6-21: TP1/LiNK definitions in which the environment variables are specified.....	475
Table 6-22: TPBroker definitions in which the environment variables are specified.....	478
Table 6-23: Environment variable specification status (for a UAP under TUXEDO).....	480
Table 6-24: Environment variable specification status (for a UAP under WebLogic Server)	482
Table 6-25: Client environment definitions	487
Table 6-26: Differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified (for characters received from a HiRDB server)	521
Table 6-27: Differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified (for characters entered at the HiRDB client).....	522
Table 6-28: Specification values of the SQL optimization option	564
Table 6-29: Recommended specification values for the SQL optimization option (for HiRDB/Parallel Server).....	566
Table 6-30: Relationships between SQL statements that create work tables and suppressing creation of update SQL-work tables	575
Table 6-31: Specification values of the SQL extension optimizing option.....	583
Table 6-32: Relationships between environment variables and connection types	607
Table 7-1: Items that can be described within an embedded SQL declare section	624
Table 7-2: Locations where SQL statements can be described.....	625
Table 7-3: Divisions in COBOL for describing SQL statements.....	662
Table 8-1: Character codes that can be specified for LANG	702
Table 8-2: Preprocessing options (for C in the UNIX environment)	704
Table 8-3: SQL preprocessor return codes (for C programs in a UNIX environment).....	708
Table 8-4: SQL preprocessor standard input and output (for C programs in a UNIX environment).....	709
Table 8-5: Preprocessing options (for COBOL in the UNIX environment)	711
Table 8-6: SQL preprocessor return codes (for COBOL programs in a UNIX environment)	714
Table 8-7: SQL preprocessor standard input and output (for COBOL programs in a UNIX environment).....	714
Table 8-8: Preprocessing options (for C in the Windows environment).....	716
Table 8-9: SQL preprocessor return codes (for C programs in a Windows environment)	719

Table 8-10: SQL preprocessor standard input and output (for C programs in a Windows environment)	720
Table 8-11: Preprocessing options (for COBOL in the Windows environment).....	722
Table 8-12: SQL preprocessor return codes (for COBOL programs in a Windows environment)	725
Table 8-13: SQL preprocessor standard input and output (for COBOL programs in a Windows environment)	725
Table 8-14: Use of pointers, structures, and pointer qualifiers when the -E2 or -E3 option is specified	734
Table 8-15: Libraries to be specified for compiling and linking (in non-OLTP environment)	737
Table 8-16: Libraries to be specified for compiling and linking (in OLTP environment).....	740
Table 8-17: Items set with Setup.....	750
Table 8-18: Item to be set with Edit Project in COBOL85.....	751
Table 8-19: Item to be set with Project Setup in COBOL2002	751
Table 8-20: Item to be set for Compilation Environment in COBOL85	751
Table 8-21: Libraries to be linked when the multi-connection facility is used.....	753
Table 8-22: Items to be set with Set.....	756
Table 8-23: Items to be specified with the Option menu	757
Table 8-24: UAP transferability from XDM/RD or UNIFY2000	769
Table 8-25: LANG and PDLANG settings for each platform	769
Table 10-1: Relationship between the use of an API (TX_function) conforming to X/Open and created SQL trace files	825
Table 10-2: Relationship between use of API (TX_function) conforming to X/Open and created error log files	840
Table 10-3: Relationship between the value of PDUAPREPLVL and information to be obtained	857
Table 10-4: UAP error types and recovery methods.....	894
Table 11-1: Generating an SQL connection.....	900
Table 11-2: Current SQL connection setting	901
Table 11-3: Current SQL connection and range of databases that can be accessed	902
Table 11-4: SQL statements supported by distributed client facility.....	909
Table 11-5: Details about SQL statements usable for remote database access.....	910
Table 11-6: Data types of variables supported by distributed client facility	922
Table 11-7: Data types set in SQL Descriptor Area of HiRDB after execution of DESCRIBE statement in the case of a HiRDB distributed server	923
Table 11-8: Data types set in SQL Descriptor Area of HiRDB after execution of DESCRIBE statement in the case of an XDM/RD distributed server.....	924
Table 11-9: Data types set in SQL Descriptor Area of HiRDB after execution of DESCRIBE statement in the case of an ORACLE distributed server.....	926
Table 11-10: Data types set in SQL Descriptor Area of HiRDB after execution of DESCRIBE statement in the case of an RDB1 E2 distributed server	927
Table 11-11: Data types set in SQL Descriptor Area of HiRDB after execution of DESCRIBE statement in the case of an SQL/K distributed server	928

Table 11-12: SQLCODEs set by distributed client when errors occur at distributed server..	930
Table 11-13: Statement information items obtained by GET DIAGNOSTICS statement when error occurs at distributed server	931
Table 11-14: Condition information items obtained by specifying condition number 1 (error at the distributed server)	931
Table 11-15: Space conversion when the distributed server is HiRDB	934
Table 12-1: Command executability from UAPs.....	946
Table 13-1: ODBC3.0 driver installation directory.....	959
Table 13-2: ODBC functions provided by HiRDB	963
Table 13-3: ODBC function data types and HiRDB data types.....	967
Table 13-4: Available facilities	969
Table 13-5: Options that can be set with the SQLSetConnectOption and SQLGetConnectOption functions	970
Table 13-6: ODBC functions that can be used by a UAP in Unicode	978
Table 14-1: Schema information provided by the HiRDB OLE DB provider.....	988
Table 14-2: Correspondences between the HiRDB data types and the OLE DB type indicators	990
Table 15-1: List of HiRDB.NET Data Provider classes.....	996
Table 15-2: Notes about HiRDB.NET Data Provider	1039
Table 15-3: HiRDBType property values that are automatically set when the DbType property is set.....	1041
Table 15-4: DbType property values that are automatically set when the HiRDBType property is set.....	1042
Table 15-5: Data types and accessories for HiRDB-type UAPs.....	1043
Table 15-6: List of type conversions for INSERT (1/2).....	1044
Table 15-7: List of type conversions for INSERT (2/2).....	1045
Table 15-8: List of type conversions for SELECT (1/2).....	1046
Table 15-9: List of type conversions for SELECT (2/2).....	1047
Table 16-1: JDBC driver's installation directory and file.....	1060
Table 16-2: Arguments of the getConnection method	1063
Table 16-3: Information to be specified for Properties info.....	1064
Table 16-4: Return values of the getColumnDisplaySize method for each SQL data type in HiRDB	1076
Table 16-5: Availability of result set types with JDBC driver	1081
Table 16-6: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (1/2)	1087
Table 16-7: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (2/2)	1089
Table 16-8: setXXX methods and JDBC SQL types to be mapped for PreparedStatement class	1091
Table 16-9: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (1/2)	1092
Table 16-10: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (2/2)	1093

Table 16-11: Classes related to connection pools	1100
Table 16-12: Classes related to distributed transactions	1102
Table 16-13: Object types returned by <code>getArray</code>	1107
Table 16-14: Attribute values of the result sets returned by <code>getResultSet</code>	1108
Table 16-15: Object types returned during data acquisition using the <code>Array.getArray()</code> method without any argument.....	1109
Table 16-16: Data type of the SQL statement specified by the <code>setObject</code> method and the data type of the array object.....	1110
Table 16-17: Method processing and notes	1115
Table 16-18: Methods of setting/acquiring connection information.....	1121
Table 16-19: Correspondence of SQL data types between HiRDB and JDBC	1149
Table 16-20: Correspondence between HiRDB character codes and Java character sets (UNIX)	1150
Table 16-21: Correspondence between HiRDB character codes and Java Character sets (Windows).....	1151
Table 16-22: Limitations to the methods in the <code>Connection</code> class that are defined in the JDBC1.0 standard	1152
Table 16-23: Limitations to the methods in the <code>Connection</code> class that are added in the JDBC2.0 basic standard	1153
Table 16-24: Limitations to the methods in the <code>Statement</code> class that are defined in the JDBC1.0 standard	1153
Table 16-25: Limitations to the methods in the <code>Statement</code> class that are added in the JDBC2.0 basic standard	1153
Table 16-26: Limitations to the methods in the <code>PreparedStatement</code> class that are added in the JDBC2.0 basic standard	1154
Table 16-27: Limitations to the methods in the <code>CallableStatement</code> class that are added in the JDBC2.0 basic standard	1154
Table 16-28: Limitations to the methods in the <code>ResultSet</code> class that are added in the JDBC2.0 basic standard	1155
Table 16-29: Limitations to the methods in the <code>ResultSetMetaData</code> class that are defined in the JDBC1.0 standard.....	1156
Table 16-30: Limitations to the methods in the <code>DatabaseMetaData</code> class that are defined in the JDBC1.0 standard.....	1157
Table 16-31: Limitations to the methods in the <code>DatabaseMetaData</code> class that are added in the JDBC2.0 basic standard	1163
Table 16-32: Limitations to the methods added by JDBC2.0 basic standards for <code>Blob</code> class	1164
Table 16-33: Restrictions on the methods added by the JDBC2.0 basic specification for the <code>Array</code> class.....	1164
Table 17-1: Specification details of the <code>getConnection</code> method arguments.....	1169
Table 17-2: HiRDB character codes and corresponding conversion character sets	1172
Table 17-3: Status of <code>ResultSet</code> objects and <code>Statement</code> objects after commit execution	1176
Table 17-4: Return values of the <code>DatabaseMetaData</code> method	1176
Table 17-5: Properties that can be specified in the <code>getConnection</code> method.....	1178

Table 17-6: Differences in how the HiRDB driver gets BLOB and BINARY data (HiRDB data types)	1185
Table 17-7: Driver interface methods.....	1192
Table 17-8: Connection interface methods.....	1193
Table 17-9: Effective holdability specifications (1/2).....	1197
Table 17-10: Effective holdability specifications (2/2).....	1198
Table 17-11: Statement interface methods	1199
Table 17-12: Priorities for number of rows that the JDBC driver requests the HiRDB server to transfer in one transmission.....	1201
Table 17-13: PreparedStatement interface methods.....	1204
Table 17-14: ResultSet interface methods.....	1208
Table 17-15: Fields supported by the ResultSet interface.....	1212
Table 17-16: Data retrieved and accumulated from the database during execution of the next method	1214
Table 17-17: Data retrieved and accumulated from the database during execution of the absolute, relative, last, or afterLast method.....	1215
Table 17-18: Number of retrieved rows that ResultSet objects can obtain from the HiRDB server	1215
Table 17-19: DatabaseMetaData interface methods.....	1216
Table 17-20: ResultSetMetaData interface methods.....	1222
Table 17-21: Blob interface methods	1224
Table 17-22: Conditions for generation of SQLWarning objects.....	1225
Table 17-23: DataSource interface methods	1233
Table 17-24: ConnectionPoolDataSource interface methods.....	1234
Table 17-25: PooledConnection interface methods	1236
Table 17-26: XAConnection interface methods.....	1237
Table 17-27: XADataSource interface methods.....	1237
Table 17-28: XAResource interface methods	1238
Table 17-29: Methods for setting and getting connection information.....	1240
Table 17-30: SQL data type correspondences between HiRDB and JDBC (Type4 JDBC driver)	1281
Table 17-31: Mapping between getXXX methods of the ResultSet class and JDBC's SQL data types (1/2).....	1282
Table 17-32: Mapping between getXXX methods of the ResultSet class and JDBC's SQL data types (2/2).....	1283
Table 17-33: JDBC SQL types mapped by the setXXX methods of the PreparedStatement class	1286
Table 17-34: Mapping between the setXXX methods of the PreparedStatement class and JDBC's SQL data types (1/2).....	1286
Table 17-35: Mapping between the setXXX methods of the PreparedStatement class and JDBC's SQL data types (2/2).....	1288
Table 17-36: Conversion processing for combinations of the TIME, DATE, and TIMESTAMP types and the setXXX methods	1291

Table 17-37: Conversion processing for combinations of the TIME, DATE, TIMESTAMP, and character string types and the getXXX methods	1293
Table 17-38: Possibility of overflow when the setXXX method is used (1/2)	1294
Table 17-39: Possibility of overflow when the setXXX method is used (2/2)	1295
Table 17-40: Possibility of overflow when the setObject method is used (1/2)	1297
Table 17-41: Possibility of overflow when the setObject method is used (2/2)	1297
Table 17-42: Possibility of overflow when the getXXX method is used (1/2)	1299
Table 17-43: Possibility of overflow when the getXXX method is used (2/2)	1300
Table 17-44: Possibility of overflow when the getObject method is used (1/2)	1301
Table 17-45: Possibility of overflow when the getObject method is used (2/2)	1302
Table 17-46: Client environment variables that can be specified with the JDBC driver	1305
Table 17-47: Priorities for connection information	1312
Table 17-48: Specifications that become effective when an authorization identifier is not specified	1316
Table 17-49: Methods that are acquisition targets of the Exception trace log and their trace acquisition levels	1322
Table 17-50: System property settings for acquisition of the Exception trace log	1327
Table 17-51: Example in which the Exception trace log is arranged in time sequence	1337
Table 17-52: Transfer of the method execution history accumulated in the JDBC driver memory	1340
Table 18-1: Files that are generated and referenced by the SQLJ Translator	1348
Table 18-2: SQL statement coding formats	1349
Table 18-3: SQL statements that can be used in SQLJ	1354
Table 18-4: Correspondence between HiRDB data types and SQLJ data types	1357
Table 18-5: Initial value for each data type and the data length set in SQL Descriptor Area	1359
Table 18-6: Description when a cursor is declared, and the acceptance area setting	1361
Table 18-7: Combinations of keyword in the WITH clause and setting values	1369
Table 18-8: SQLJ Translator options	1377
Table 18-9: Migrating an SQLJ source to the native interface version	1379
Table 18-10: Configuration of the Native Runtime packages	1383
Table 18-11: Public classes of Native Runtime	1383
Table A-1: Contents of the SQL Communications Area	1398
Table B-1: Contents of the SQL Descriptor Area	1407
Table B-2: Data codes and data lengths set in the SQL Descriptor Area	1413
Table B-3: Contents of SQLVAR_LOB	1416
Table B-4: SQL Descriptor Area expansion procedure	1423
Table B-5: SQL Descriptor Area operation macros	1424
Table B-6: Macros for specifying data types	1424
Table B-7: Repetition column expansion format	1427
Table C-1: Contents of the Column Name Descriptor Area	1430
Table D-1: Contents of the Type Name Descriptor Area	1433
Table E-1: SQL data types and C data descriptions	1436
Table E-2: SQL data types and C data descriptions when arrays are used	1441

Table E-3: SQL data types and C data descriptions when repetition columns are used	1443
Table E-4: Macros for referencing or setting embedded variables	1445
Table E-5: Pointer variables and C language data description.....	1448
Table E-6: Macros for pointer-type repetition columns	1452
Table E-7: Structures to be specified in batches	1454
Table E-8: SQL data types and COBOL data descriptions	1455
Table E-9: SQL data types and COBOL data descriptions when arrays are used	1460
Table E-10: SQL data types and COBOL data descriptions when repetition columns are used.....	1464
Table F-1: Data dictionaries	1469
Table F-2: SQL_PHYSICAL_FILES table contents	1476
Table F-3: SQL_RDAREAS table contents	1477
Table F-4: SQL_TABLES table contents.....	1479
Table F-5: SQL_COLUMNS table contents	1487
Table F-6: Values that are stored when the DEFAULT clause is specified.....	1493
Table F-7: SQL_INDEXES table contents.....	1498
Table F-8: SQL_USERS table contents	1500
Table F-9: SQL_RDAREA_PRIVILEGES table contents	1502
Table F-10: SQL_TABLE_PRIVILEGES table contents	1502
Table F-11: SQL_VIEW_TABLE_USAGE table contents	1503
Table F-12: SQL_VIEWS table contents.....	1504
Table F-13: SQL_DIV_TABLE table contents.....	1504
Table F-14: SQL_INDEX_COLINF table contents.....	1505
Table F-15: SQL_DIV_INDEX table contents	1506
Table F-16: SQL_DIV_COLUMN table contents	1507
Table F-17: SQL_ROUTINES table contents.....	1507
Table F-18: SQL_ROUTINE_RESOURCES table contents	1515
Table F-19: SQL_ROUTINE_PARAMS table contents	1517
Table F-20: SQL_ALIASES table contents	1520
Table F-21: SQL_TABLE_STATISTICS table contents.....	1521
Table F-22: SQL_COLUMN_STATISTICS table contents.....	1521
Table F-23: SQL_INDEX_STATISTICS table contents.....	1523
Table F-24: SQL_DATATYPES table contents.....	1523
Table F-25: SQL_DATATYPE_DESCRIPTOR table contents	1524
Table F-26: SQL_TABLE_RESOURCES table contents.....	1526
Table F-27: SQL_PLUGINS table contents.....	1527
Table F-28: SQL_PLUGIN_ROUTINES table contents	1528
Table F-29: SQL_PLUGIN_ROUTINE_PARAMS table contents	1529
Table F-30: SQL_INDEX_TYPES table contents.....	1531
Table F-31: SQL_INDEX_RESOURCES table contents	1532
Table F-32: SQL_INDEX_DATATYPE table contents	1532
Table F-33: SQL_INDEX_FUNCTION table contents.....	1533
Table F-34: SQL_TYPE_RESOURCES table contents.....	1534
Table F-35: SQL_INDEX_TYPE_FUNCTION table contents.....	1534

Table F-36: SQL_EXCEPT table contents	1535
Table F-37: SQL_FOREIGN_SERVERS table contents.....	1535
Table F-38: SQL_USER_MAPPINGS table contents.....	1537
Table F-39: SQL_IOS_GENERATIONS table contents	1538
Table F-40: SQL_TRIGGERS table contents.....	1538
Table F-41: SQL_TRIGGER_COLUMNS table contents	1540
Table F-42: SQL_TRIGGER_DEF_SOURCE table contents.....	1541
Table F-43: SQL_TRIGGER_USAGE table contents.....	1542
Table F-44: SQL_PARTKEY table contents	1543
Table F-45: SQL_PARTKEY_DIVISION table contents.....	1543
Table F-46: SQL_AUDITS table contents.....	1544
Table F-47: SQL_REFERENTIAL_CONSTRAINTS table contents.....	1546
Table F-48: SQL_KEYCOLUMN_USAGE table contents	1547
Table F-49: SQL_TABLE_CONSTRAINTS table contents	1548
Table F-50: SQL_CHECKS table contents.....	1548
Table F-51: SQL_CHECK_COLUMNS table contents	1549
Table F-52: SQL_DIV_TYPE table contents	1550
Table F-53: SQL_SYSPARAMS table contents.....	1551
Table G-1: Execution conditions in the HiRDB client	1554
Table G-2: Items to be set in the HiRDB server with Set Project or Set	1556
Table G-3: Items to be set in the HiRDB client with Set Project or Set	1557
Table G-4: Double-byte space characters specified in ncspace	1560
Table G-5: Hash function codes for hash functions.....	1564
Table G-6: Area for setting partitioning keys	1565
Table G-7: Data type codes and data length codes	1565
Table G-8: Area for setting the data address of a partitioning key	1566
Table G-9: Macros for maximum values	1567
Table H-1: HiRDB maximum and minimum values	1590

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.

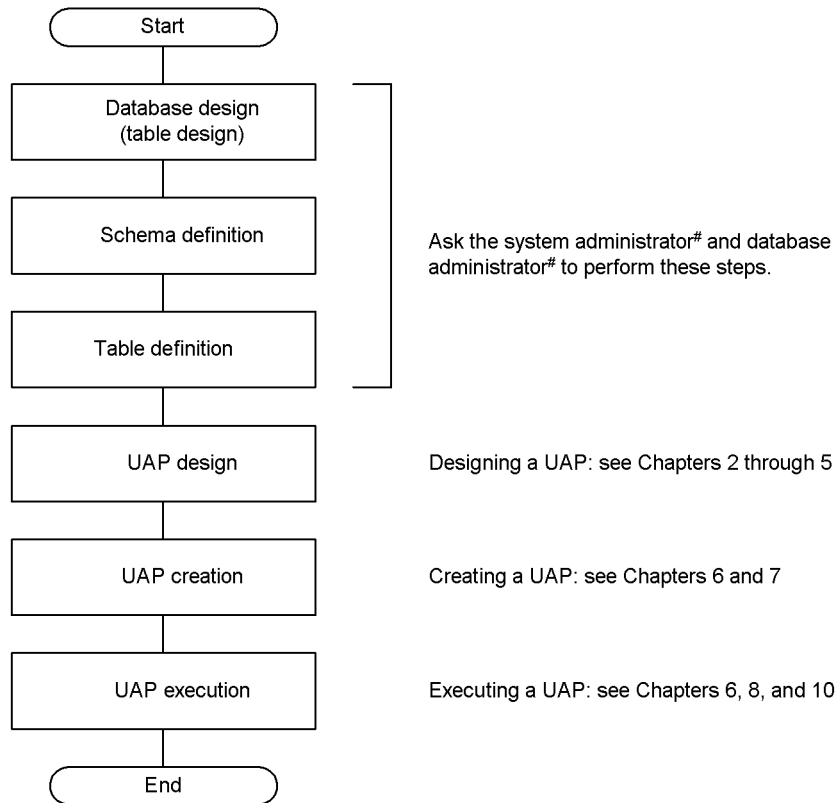
This chapter contains the following sections:

- 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. Figure 1-1 shows the relationship between UAP creation and the organization of this manual.

Figure 1-1: UAP development flow



#: For work details, see *HiRDB Version 8 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™ (SQLJ).

(1) *Source program*

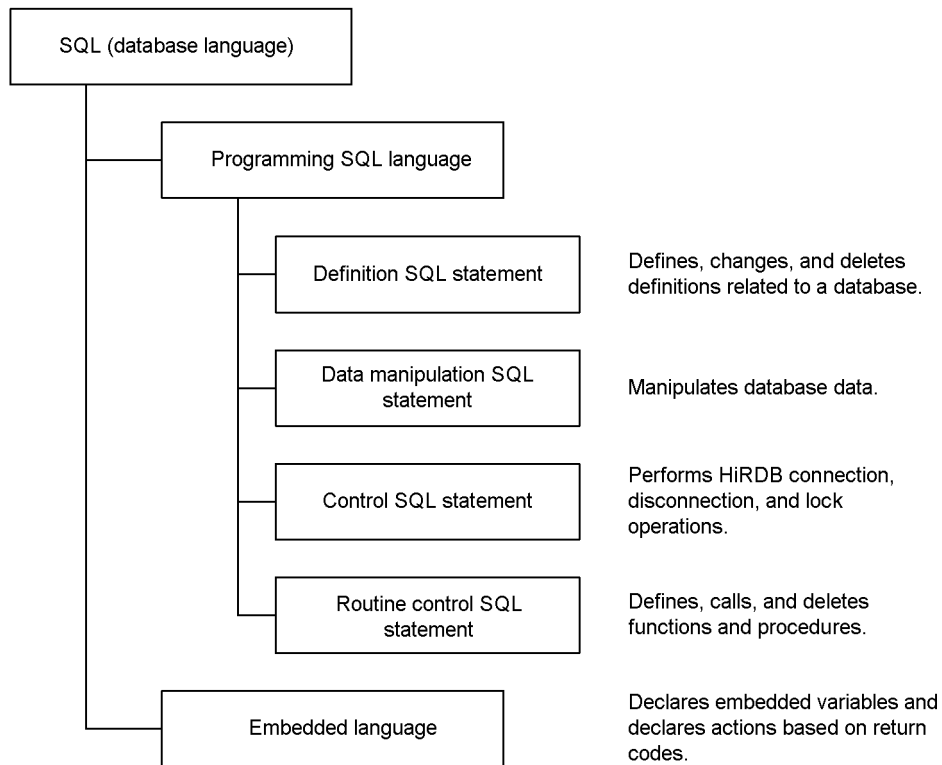
The following high-level languages can be used to write an embedded SQL UAP:

- 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. Figure 1-2 shows the SQL functional organization.

Figure 1-2: SQL functional organization



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 8 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 8 SQL Reference

Database definitions

HiRDB Version 8 Installation and Design Guide

Database operations

2. Database Operations

Database management

3. UAP Design

Embedded language

HiRDB Version 8 SQL Reference

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

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
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)	<ul style="list-style-type: none"> • 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	—	—	—
COMMENT (add comment)	Provides a comment in a table or column.	U	U	—	—	—

1. Overview

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
CREATE ALIAS (define alias)	Assigns an alias for a table.	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 FOREIGN INDEX (define foreign index)	Defines a foreign index.	U	U	—	—	—
CREATE FOREIGN TABLE (define foreign table)	Defines a foreign table.	U	U	—	—	—
CREATE FUNCTION (define function)	Defines a 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 SCHEMA (define schema)	Defines a schema.	U	U	—	—	—
CREATE SERVER (define foreign server)	Defines a foreign server.	U	U	—	—	—
CREATE TABLE (define base table)	Defines a base table.	U	U	—	—	—
CREATE TRIGGER (define trigger)	Defines a trigger.	U	U	—	—	—

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
CREATE TYPE (define type)	Defines an abstract data type.	U	U	—	—	—
CREATE USER MAPPING (define user mapping)	Defines a user mapping.	U	U	—	—	—
CREATE VIEW (define view table)	Defines a view table.	U	U	—	—	—
DROP ALIAS (delete alias)	Deletes a table alias.	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 FOREIGN INDEX (delete foreign index)	Deletes the definition of a foreign index.	U	U	—	—	—
DROP FOREIGN TABLE (delete foreign table)	Deletes the definition of a foreign table.	U	U	—	—	—
DROP FUNCTION (delete function)	Deletes a function.	U	U	—	—	—
DROP INDEX (delete index)	Deletes an index.	U	U	—	—	—
DROP PROCEDURE (delete procedure)	Deletes a procedure.	U	U	—	—	—

1. Overview

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
DROP SCHEMA (delete schema)	Deletes a schema.	U	U	—	—	—
DROP SERVER (delete foreign server)	Deletes the definition of a foreign server.	U	U	—	—	—
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 USER MAPPING (delete user mapping)	Deletes a user mapping.	U	U	—	—	—
DROP VIEW (delete view table)	Deletes a 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	—	—	—

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
GRANT <i>access-privilege</i> (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	—	—	—
REVOKE <i>access-privilege</i> (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	Distributed database (server type)	
					HiRDB	Other than HiRDB
ALLOCATE CURSOR statement (allocate cursor)	Allocates a cursor for a <code>SELECT</code> statement preprocessed by the <code>PREPARE</code> 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	U	—
CLOSE statement (close cursor)	Closes a cursor.	U	U	U	U	U
DEALLOCATE PREPARE statement (release preprocessing)	Releases the allocation of an SQL statement preprocessed by the <code>PREPARE</code> statement.	U	U	U	—	—
DECLARE CURSOR (declare cursor)	Declares a cursor that the results of a retrieval by the <code>SELECT</code> statement can be fetched row by row with the <code>FETCH</code> statement.	U	U	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	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	Distributed database (server type)	
					HiRDB	Other than HiRDB
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	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	U	U
EXECUTE IMMEDIATE statement (preprocess and execute SQL)	Preprocesses and executes an SQL statement provided in a character string.	U	U	U	U	U

1. Overview

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
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	U	U
FREE LOCATOR statement (invalidate locator)	Invalidates a locator.	U	U	U	—	—
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	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	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	U	U
PURGE TABLE statement (delete all rows)	Deletes all rows in a base table.	U	U	—	U	—

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
Single-row <code>SELECT</code> statement (retrieve one row)	Searches table data. To fetch only one row of data from a table, the single-row <code>SELECT</code> statement can be used without having to declare a cursor.	U	U	U	U	U
Dynamic <code>SELECT</code> statement (retrieve dynamically)	Searches table data. The dynamic <code>SELECT</code> statement is preprocessed by the <code>PREPARE</code> statement. During the search, a cursor declared by <code>DECLARE CURSOR</code> or allocated by <code>ALLOCATE CURSOR</code> is used to fetch the retrieval results row by row.	U	U	U	U	U
<code>UPDATE</code> 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	U	U
Preparable dynamic <code>UPDATE</code> 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.

* If a procedure is called under OLTP, or if a procedure defined on a distributed server is called when a distributed database facility is used, that procedure 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	Distributed database (server type)	
					HiRDB	Other than HiRDB
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	—	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	—	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	—	U	U
LOCK statement (lock control on tables)	Performs exclusive locks on specified tables.	U	U	U	U	—
CONNECT statement with RD-node specification (connect to distributed RD-node)	Relays an authorization identifier and a password to a distributed RD-node so that a UAP can use that distributed RD-node.	U	U	U	U	U
DISCONNECT statement with RD-node specification (disconnect from distributed RD-node)	Terminates the current transaction normally, establishes a synchronization point, and creates a single commitment unit. The UAP is then disconnected from the distributed RD-node.	U	U	—	U	U
ROLLBACK statement (cancel transaction)	Cancels the current transaction and nullifies the database updating performed by the transaction.	U	U	—	U	U

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (server type)	
					HiRDB	Other than HiRDB
SET CONNECTION statement (set current RD-node)	Sets the current RD-node.	U	U	U	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.

* The system automatically connects to the distributed server DBMS when the first data manipulation SQL that accesses a distributed server database is executed, rather than when the CONNECT statement is executed. After the CONNECT statement is executed, the system also connects to the distributed server DBMS, if the CONNECT statement is executed again with the RD node specified.

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

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (servertype)	
					HiRDB	Other than HiRDB
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	DDF	DDF
END DECLARE SECTION (declare end of embedded SQL)	Indicates the end of an embedded SQL declare section.	U	U	U	DDF	DDF

1. Overview

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (servertype)	
					HiRDB	Other than HiRDB
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	—	—	—	—
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	DDF	DDF
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	U	U
COMMAND EXECUTE (execute commands from UAP)	Executes HiRDB and OS commands from inside the UAP.	U	—	—	—	—

SQL	Function	Usability				
		C	COBOL	OLTP	Distributed database (servertype)	
					HiRDB	Other than HiRDB
SQL prefix	Indicates the beginning of SQL statements.	U	U	U	DDF	DDF
SQL terminator	Indicates the end of SQL statements.	U	U	U	DDF	DDF
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	DDF	DDF
SQLCODE variable	Receives the return code issued by HiRDB after an SQL statement has been executed.	U	U	U	DDF	DDF
SQLSTATE variable	Receives the return code issued by HiRDB after an SQL statement has been executed.	U	U	U	DDF	DDF
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	—	—	—	—
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.

DDF: Cannot be executed on a distributed server; however, it can be used in UAPs that use the distributed database function.

* 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	Distributed database (servertype)	
					HiRDB	Other than HiRDB
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	—	—	—
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.

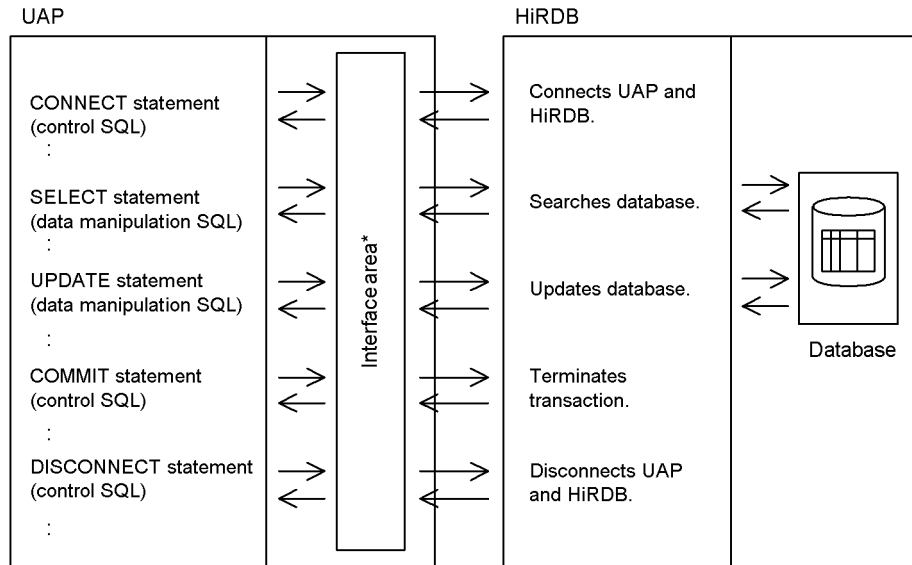
³ This statement cannot be used in the `CREATE FUNCTION` statement.

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.

Figure 1-3 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 ODBC¹-compatible application program is used as the client
- Operating mode in which an OLE DB²-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 AP8000³ UAP is used as the client (limited to HiRDB clients for UNIX systems)

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 *13. 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 *14. 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

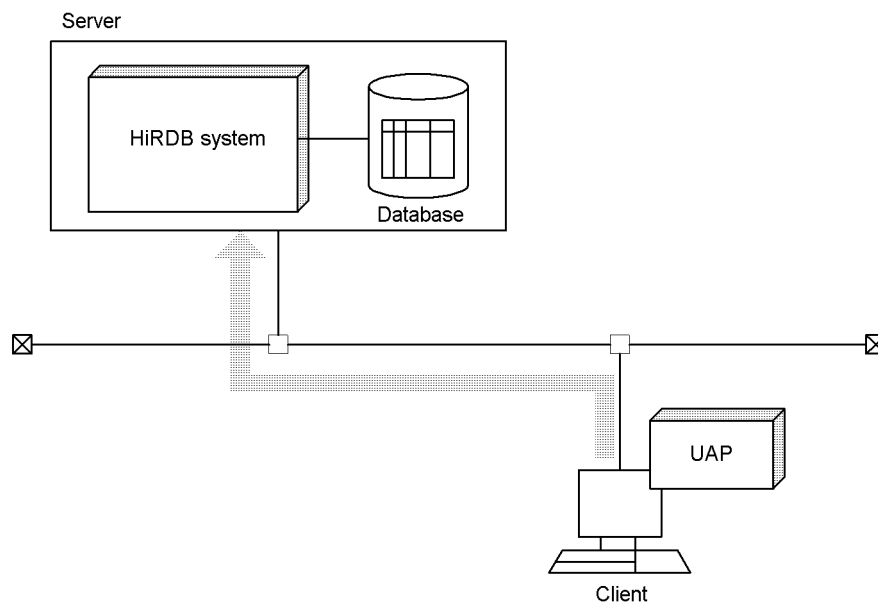


Figure 1-5: Operating mode using the same server machine as the HiRDB server as the client

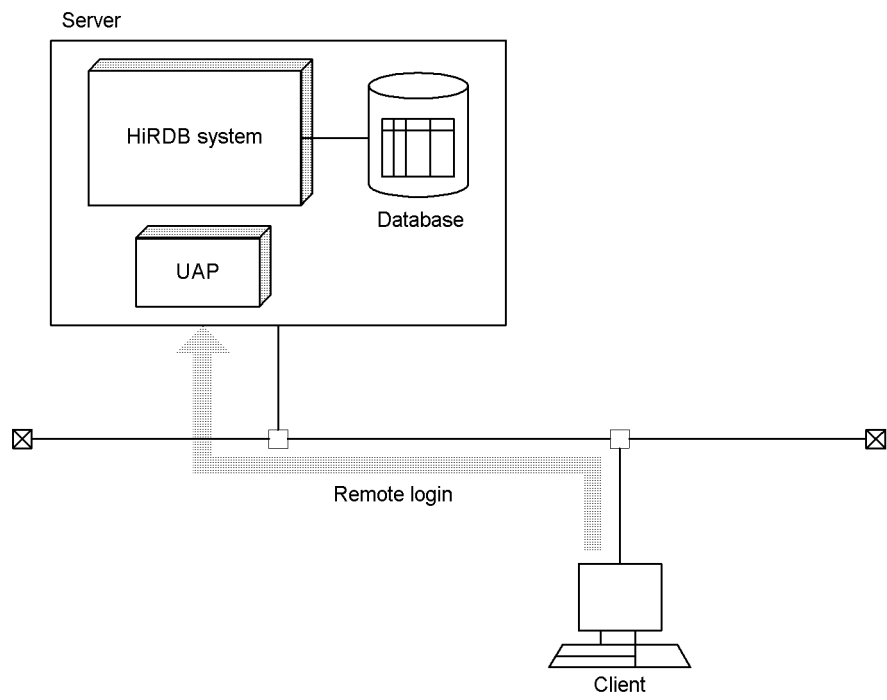


Figure 1-6: Operating mode using a UAP under OLTP as a client

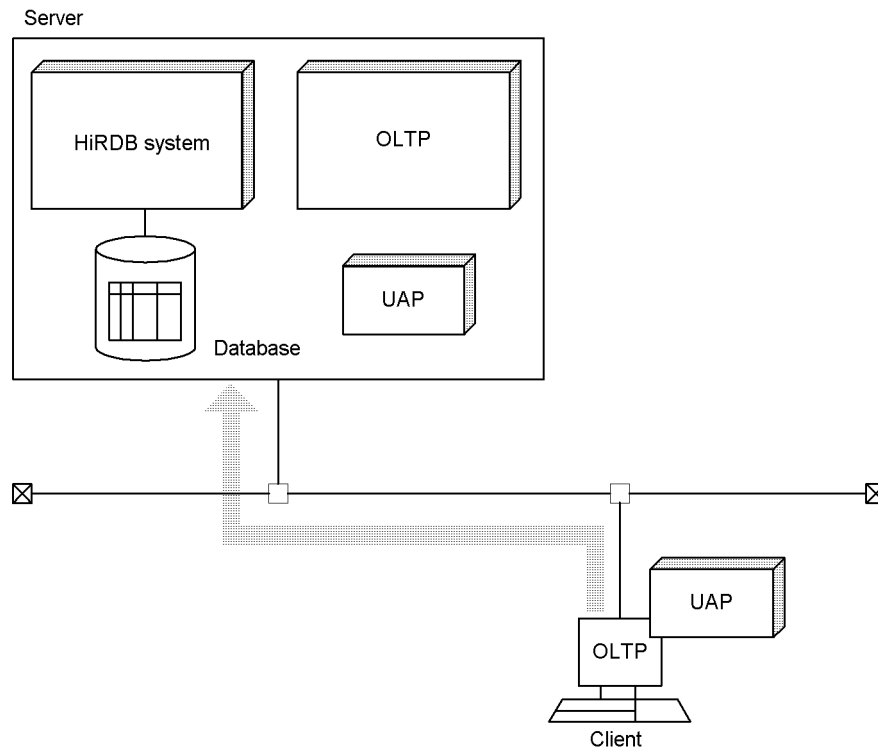
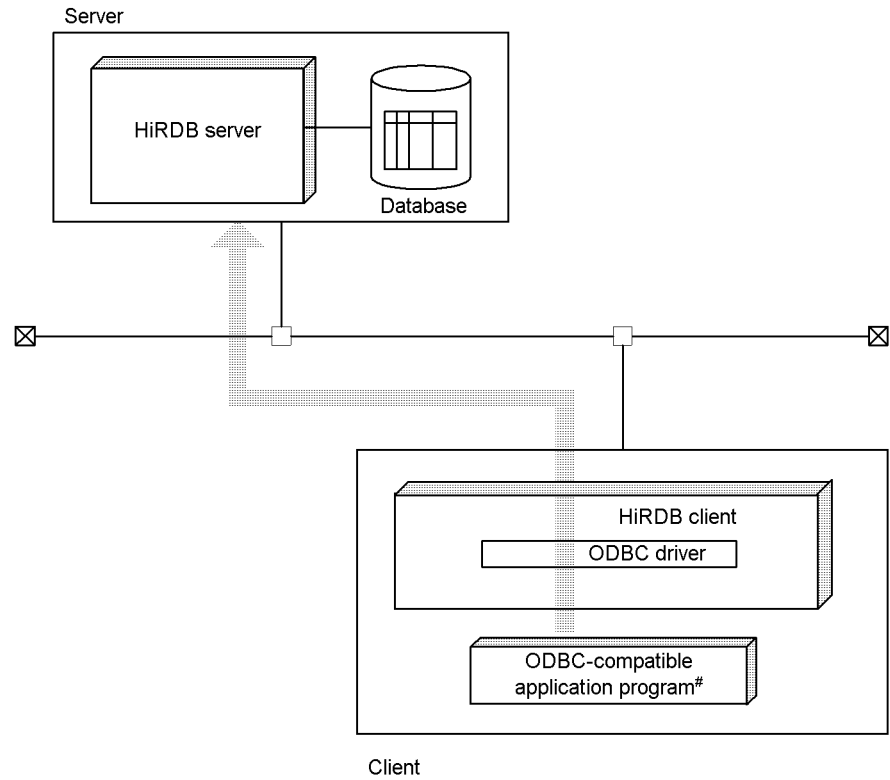
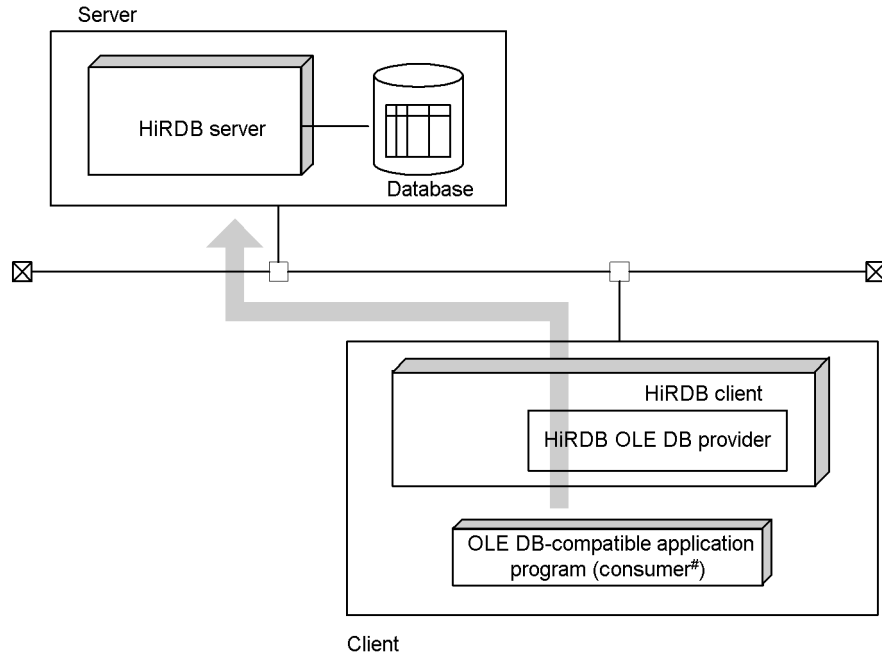


Figure 1-7: Operating mode using an ODBC-compatible UAP as a client



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

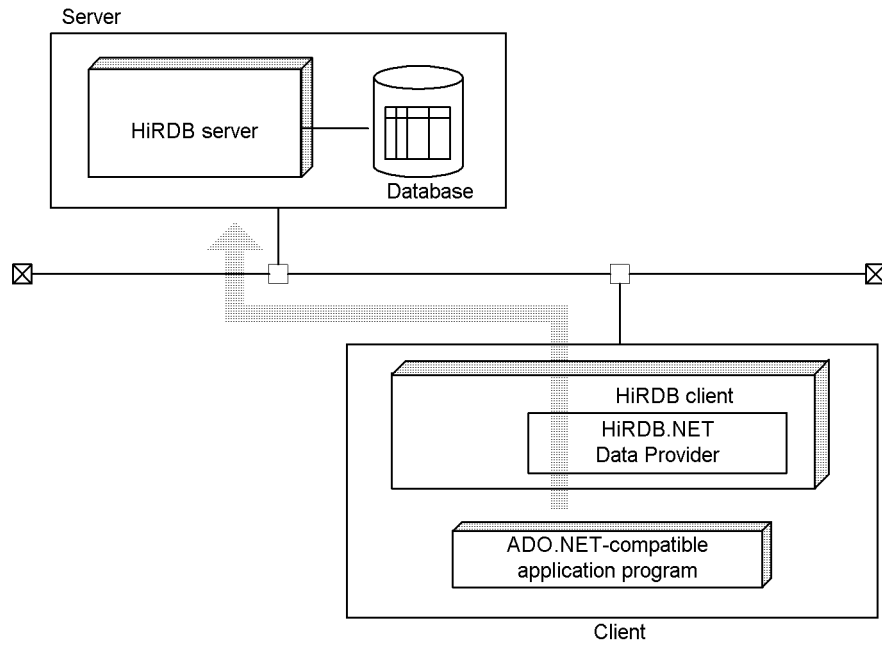


Figure 1-10: Operating mode using a Java (JDBC-compatible) application program as a client

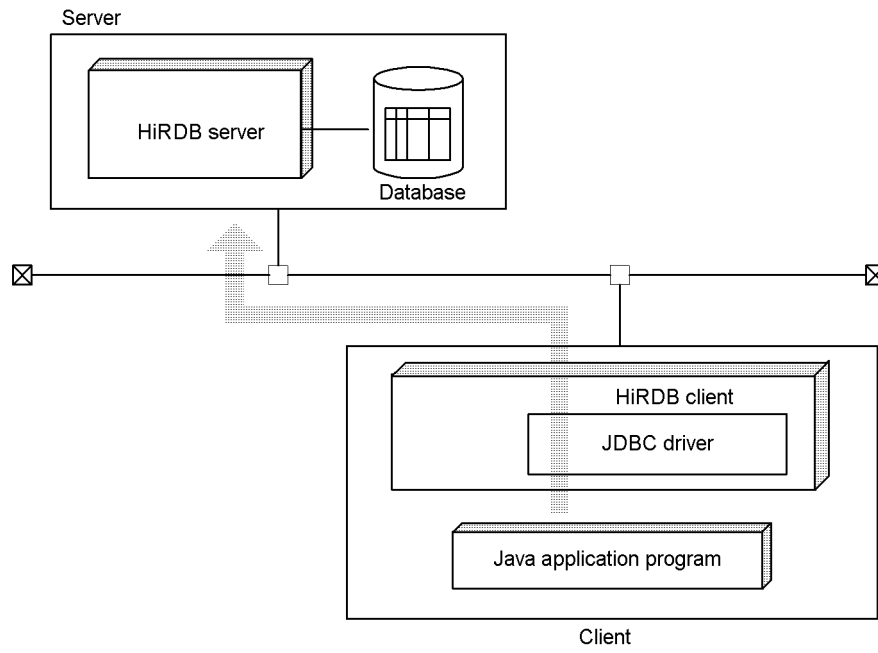
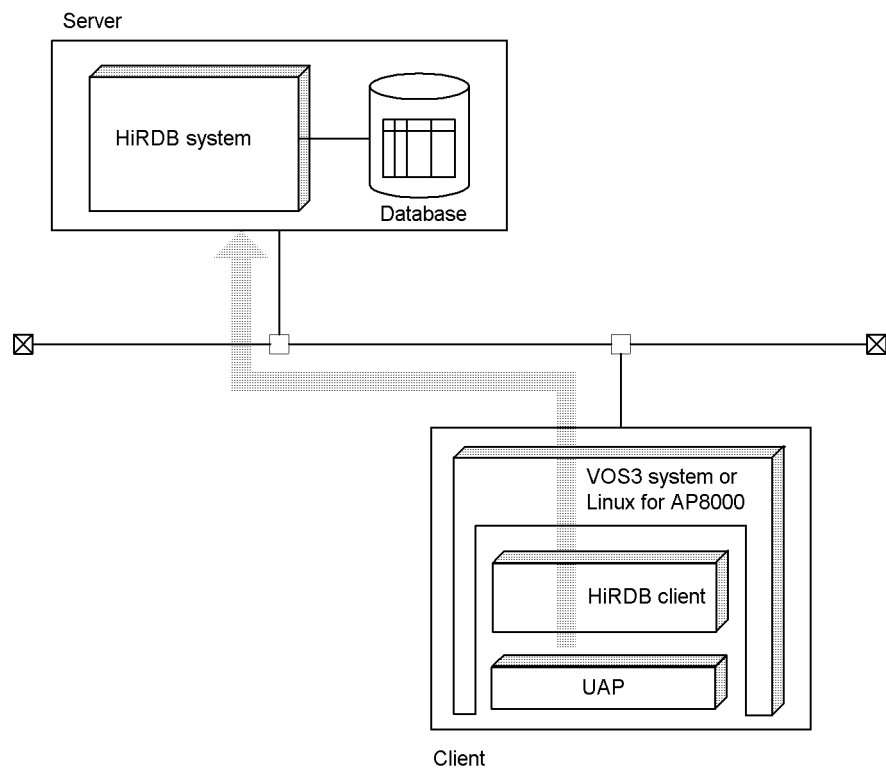


Figure 1-11: Operating mode using a VOS3 system or Linux for AP8000 UAP as a client



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 8 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.

Figure 2-1 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	← Data type
Product code	Product name	Color	Price	Stock quantity	
PCODE	PNAME	COLOR	PRICE	SQUANTITY	← Column name
101L	BLOUSE	BLUE	35.00	62	
101M	BLOUSE	WHITE	35.00	85	← Row
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	

↑ Column

(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.

Figure 2-2 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		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			

← Elements of repetition column

← Row

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.

Figure 2-3 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

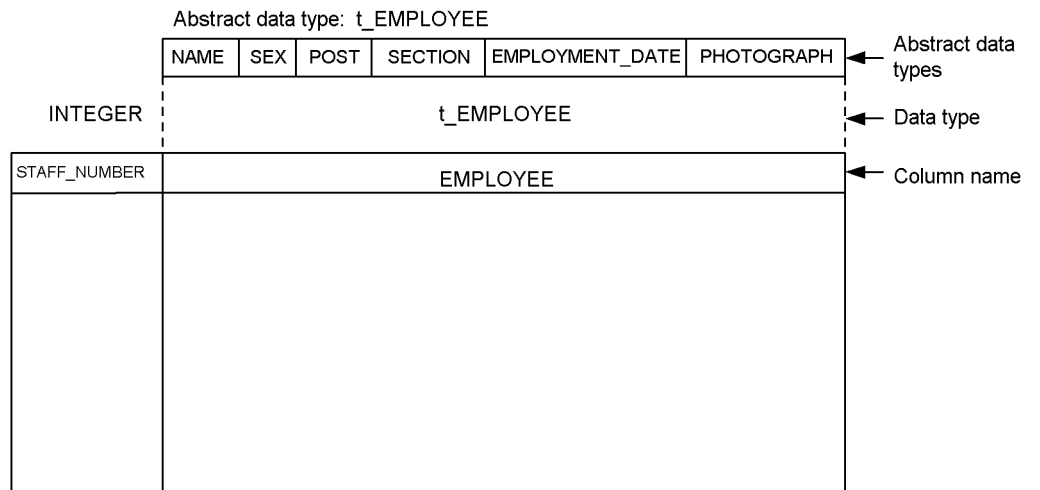
2.1.2 Object relational database tables

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.

Figure 2-4 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

Table: STAFF_TABLE



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

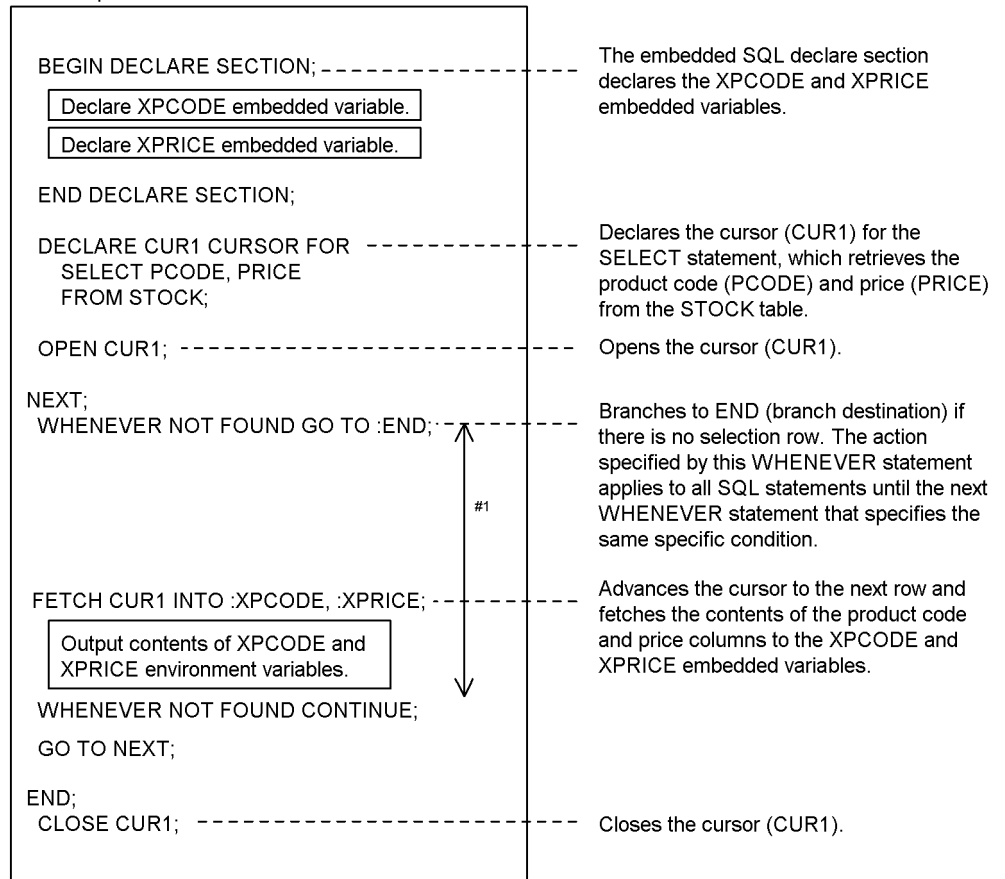
When table retrieval results consist of multiple rows or when retrieving data dynamically after preprocessing the SQL statement with the `PREPARE` statement, a cursor is used to retrieve the individual rows.

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 8 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



Note

The SQL prefix and the SQL terminator have been omitted.

#1 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

<pre>BEGIN DECLARE SECTION; ----- Declare XPRICE embedded variable. Declare PPRICE embedded variable. END DECLARE SECTION; DECLARE CUR1 CURSOR FOR ----- SELECT PRICE FROM STOCK FOR UPDATE OF PRICE; OPEN CUR1;----- NEXT; WHENEVER NOT FOUND GO TO :END; ----- FETCH CUR1 INTO :XPRICE; ----- WHENEVER NOT FOUND CONTINUE; ----- Multiply price of XPRICE embedded variable by 0.9 and set result to PPRICE embedded variable. UPDATE STOCK SET PRICE=:PPRICE ----- WHERE CURRENT OF CUR1; GOTO NEXT; END; CLOSE CUR1; -----</pre>	<p>The embedded SQL declare section declares the XPRICE and PPRICE embedded variables.</p> <p>Declares the cursor (CUR1) for fetching the price column retrieval results one row at a time from the STOCK table.</p> <p>Opens the cursor (CUR1).</p> <p>Branches to END (branch destination) if there is no selection row.</p> <p>Advances the cursor to the next row and reads the value of the price column to the XPRICE embedded variable.</p> <p>Executes the next instruction if there is no row to process.</p> <p>Updates the price column value in the row where the cursor is positioned with the value of the PPRICE embedded variable.</p> <p>Closes the cursor (CUR1).</p>
--	---

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

<pre>SELECT COUNT(*) INTO :PCOUNT ----- FROM STOCK</pre>	<p>Fetches the count from the stock table (STOCK) into the PCOUNT embedded variable.</p>
--	--

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

As an example of a retrieval from a single table, Figure 2-5 shows a case in which a `SELECT` statement is used to retrieve from a stock table those rows containing `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='SKIRT'
```

Retrieval results

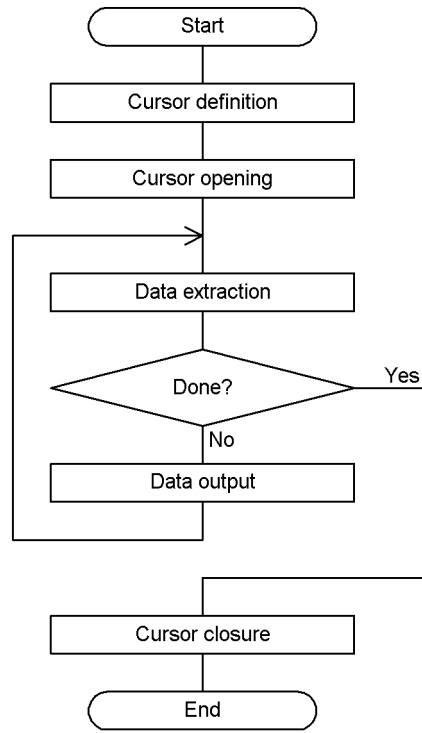
PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

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.

Figure 2-6 shows the 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

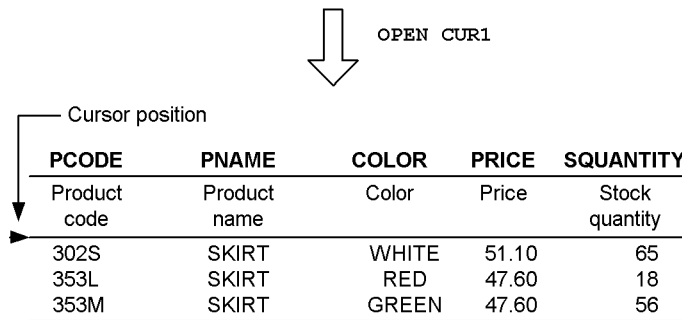
When a cursor is opened, retrieval results can be extracted in accordance with the defined conditions. The retrieval results are stored in a table format in the system and remain valid until the cursor is closed.

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. Figure 2-7 shows a cursor immediately after it has been opened; in this example, the cursor name is `CUR1` and `SKIRT` is the retrieval condition for the stock table.

Figure 2-7: Cursor position immediately following cursor opening



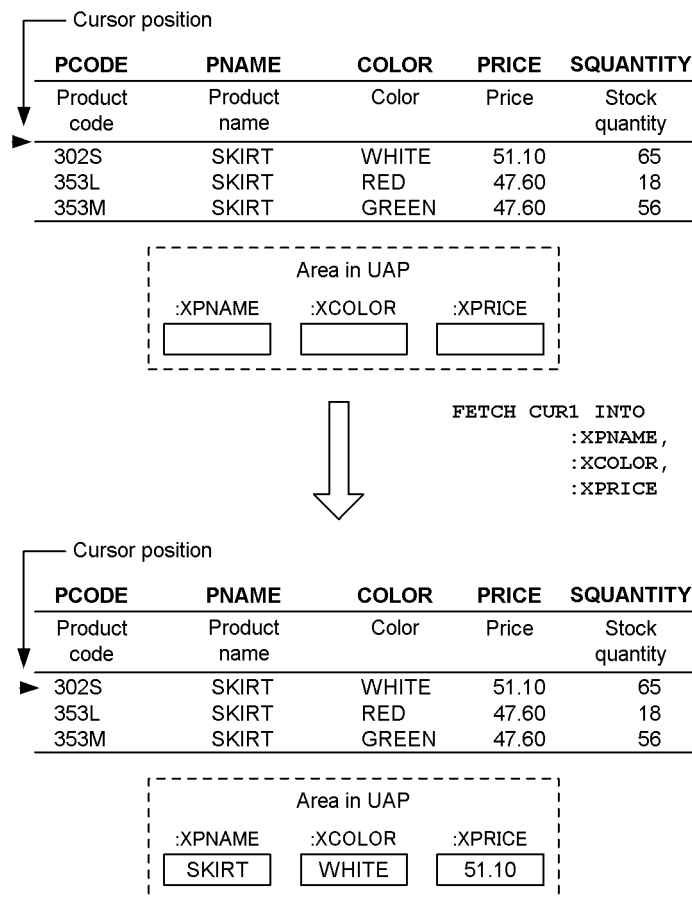
PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56

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 example in Figure 2-8 shows the cursor immediately after it has been opened, and shows how the retrieved contents are stored in the UAP.

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



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. In the example in Figure 2-9, a single set of results is obtained from two tables. The common `PCODE` column, indicating product code, is used to join the tables `STOCK` and `ORDER`. Here, a table consisting of form numbers and product names is created 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
```

Retrieval results

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.

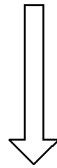
Retrieval on a row basis reduces retrieving overhead for each column, so it enhances access performance.

Figure 2-10 shows an example of retrieval on a row basis. Here, a cursor (`CUR1`) is used, and product name `POLO SHIRT` only is retrieved from the stock table and set into the embedded variable (`:XROW`). For details about an embedded variable, see the *HiRDB Version 6 SQL Reference manual*.

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:

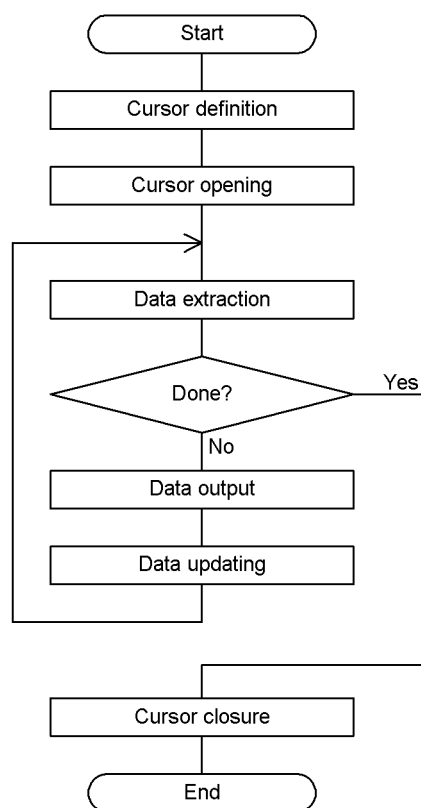
- Updating the row indicated by the cursor
- Updating only those rows that satisfy a condition
- Updating on a row basis (for table with `FIX` attribute only)

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 extract one row at a time. Figure 2-11 shows how to use the cursor to update a table.

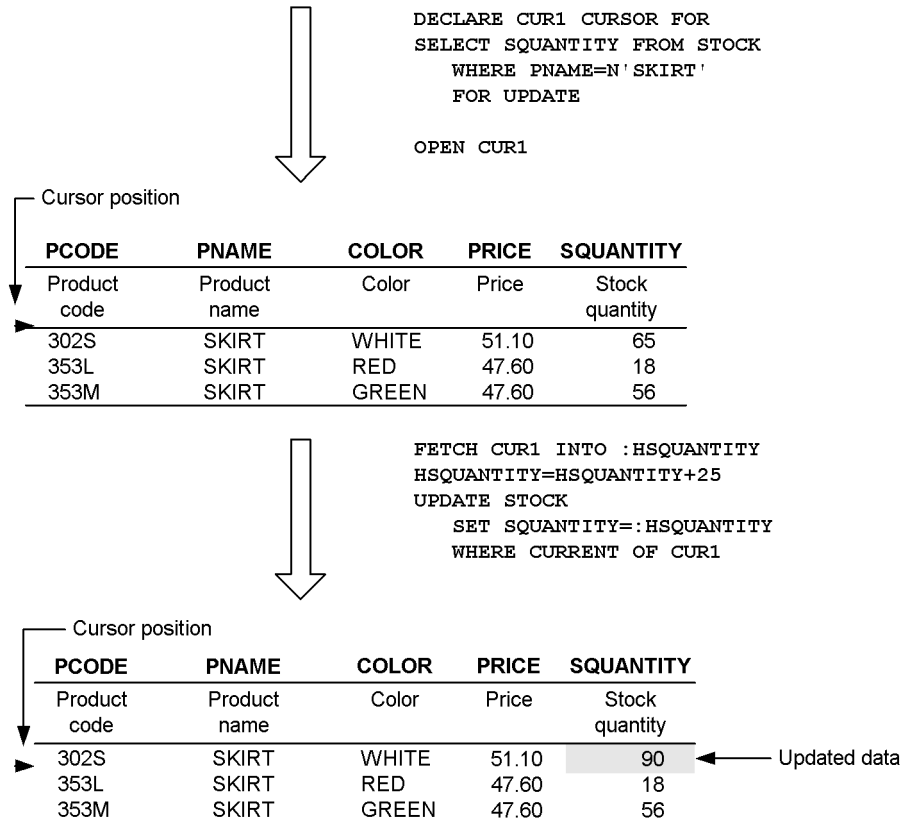
Figure 2-11: Procedure for updating a table



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.

Figure 2-12 shows an example of using the cursor to update a table. It is assumed that the steps up to data fetching have been completed.

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



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.

Figure 2-13 shows an example of updating with a condition specified. Here, the quantity of each item whose product code is 411M is updated to 20.

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
:	:	:	:	:

← Updated data

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.

Figure 2-14 shows an example of updating on a row basis. Here, the quantity of each item in the stock table whose product code is `411M` is updated from 12 to 20; the new value is specified in the 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.

Figure 2-15 shows an example of updating a table that has repetition columns. This example adds the element `DATABASE` to the qualifications of the employee named `SMITH, BOB` in a staff table.

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

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			



```
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	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
	DATABASE				
DAVIS, JIM	SYSTEM ADMINISTRATOR	MALE	CHERYL	MOTHER	1
BOYD, SCOTT		MALE			

Updated data
(added element)

2.5 Data deletion

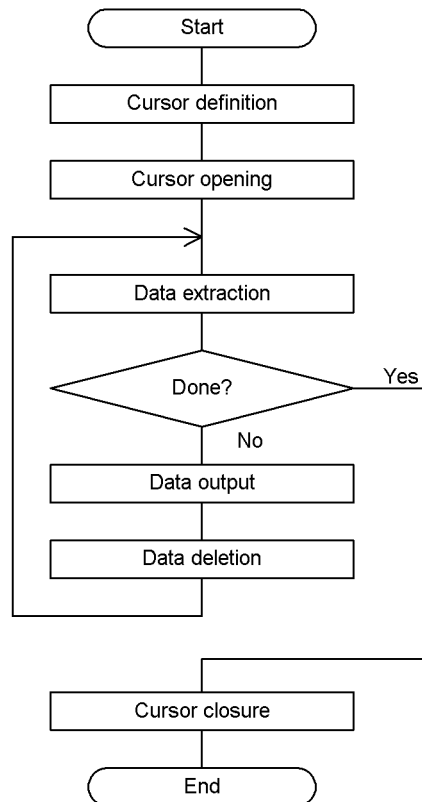
The following three methods are provided for deleting information in a table:

- 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. Figure 2-16 shows the procedure for using a cursor to delete rows in a table.

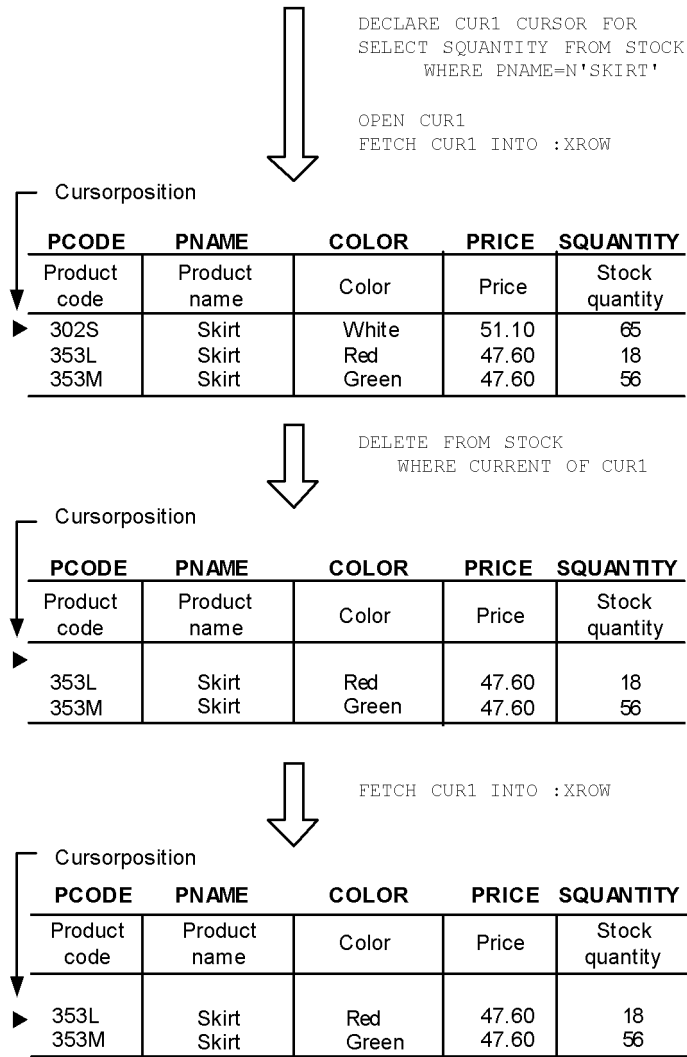
Figure 2-16: Procedure for deleting a table



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.

Figure 2-17 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



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.

Figure 2-18 shows an example of deletion with a condition specified. Here, only the items whose product name is `SKIRT` are deleted from the stock table.

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='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.

Figure 2-19 shows an example of deleting all rows in 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.

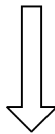
Figure 2-20 shows an example of insertion on a column basis. Here, values set in embedded variables (`:ZPCODE` to `:ZSQANTITY`) are inserted in the columns of the stock table.

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.

Figure 2-21 shows an example of inserting a row on a row basis. Here, the values set in the embedded variable (`:ZROW`) are inserted on a row basis into the stock table.

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

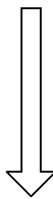
2.6.3 Inserting rows into a table with repetition columns

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]...]`. Figure 2-22 shows an example of inserting rows into a table that has repetition columns. This example inserts a row into a 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	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			



```
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	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			
HUTTON, SARAH	INFORMATION PROCESSING 1	FEMALE	ERIC	HUSBAND	1
	INFORMATION PROCESSING 2		RICHARD	SON	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.

Figure 2-23 shows a data search example using a comparison predicate. Here, the product codes and product names of products with 50 or fewer units in stock are searched from the stock table.

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	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, 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.

Figure 2-24 shows a data search example using a BETWEEN predicate. Here, the product codes and product names of products with between 200 and 300 units in stock are searched from the stock table.

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.

Figure 2-25 shows a data search example using an `IN` predicate. Here, the product codes and product names of products whose unit price is either 36.40 or 47.60 are searched from the stock table.

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.

Figure 2-26 shows a data search example using a `LIKE` predicate. Here, form numbers, product codes, and the ordered quantities are searched from the order table for those customer codes whose second character is `T`.

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.

Figure 2-27 shows a data search example using a `NULL` predicate with `NOT`. Here, form numbers, product codes, and the ordered quantities are searched from the order table for those in which customer codes 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	Orderreceived date	Orderreceived 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

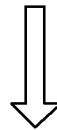
The logical operators AND, OR, and NOT manipulate rows containing data that satisfies multiple conditions.

Figure 2-28 shows a data search example involving multiple conditions. Here, product codes and ordered quantities are searched from the stock table for products whose name is either BLOUSE or POLO SHIRT with 50 or more units in a stock.

Figure 2-28: Data search example involving multiple conditions

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



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

Searchresults

PCODE	SQUANTITY
Product code	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 *2.12.1 Abstract data types provided by the HiRDB Text Search Plug-in*.

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.

Figure 2-29 shows an example of a data search using a structured repetition predicate. In this example, employees who support their father are retrieved from a staff table.

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.

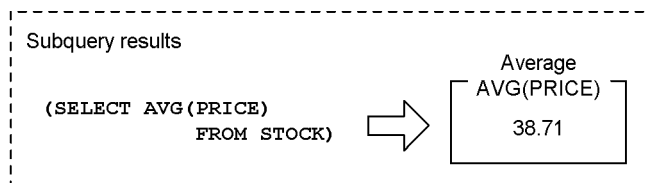
Figure 2-30 shows an example of a data search using a subquery. This example retrieves from a stock table the product codes of products whose prices equal or exceed the average price.

Figure 2-30: Data search example using a subquery

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



Search results

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

(1) Subquery using a quantified predicate

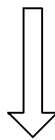
A *quantified predicate* can be used to determine whether or not the results of a subquery satisfy a specified set of comparison conditions and to further narrow the retrieval results. Figure 2-31 shows an example of a subquery using a *quantified predicate*. The example retrieves from the stock table the product codes and names of

products with a greater stock quantity than any quantity of blouse (regardless of the product code).

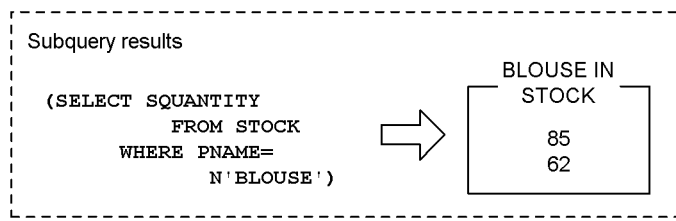
Figure 2-31: Data search example using a subquery and a quantified predicate

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 ')
```



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.

Figure 2-32 shows an example of a subquery using the EXISTS predicate. The example retrieves from the stock table and the order table the names of products for which no orders have been placed.

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

STOCK (Stocktable)

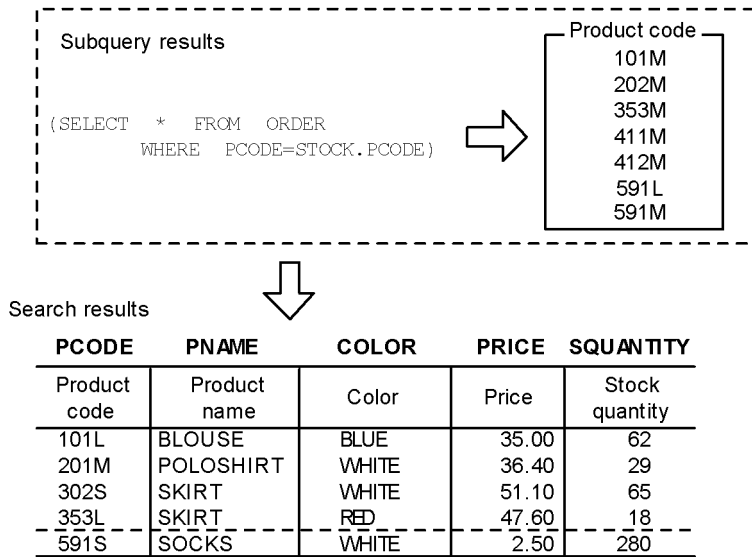
PCODE	PNAME	COLOR	PRICE	SQUANTITY
Product code	Product name	Color	Price	Stock quantity
101L	BLOUSE	BLUE	35.00	62
101M	BLOUSE	WHITE	35.00	85
201M	POLOSHIRT	WHITE	36.40	29
202M	POLOSHIRT	RED	36.40	67
302S	SKIRT	WHITE	51.10	65
353L	SKIRT	RED	47.60	18
353M	SKIRT	GREEN	47.60	56
411M	SWEATER	BLUE	84.00	12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

2. Database Operations



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.

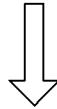
The four types of arithmetic operations are addition, subtraction, multiplication, and division.

Figure 2-33 shows an example of performing arithmetic operations on numeric data. Here, projected revenue figures are calculated on the basis of the unit prices and stock quantities of the `FASTBACK` model; the product codes and calculation results are extracted in million dollar units.

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.

Figure 2-34 shows an example of a time data operation. Here, the form numbers, the product codes, and the ordered quantities for orders received before noon (12:00:00) are extracted from the sales orders table.

Figure 2-34: Example of time data operation

ORDER (Ordertable)

FNO	CCODE	PCODE	OQUANTITY	ODATE	OTIME
Form number	Customer code	Product code	Ordered quantity	Orderreceived date	Orderreceived 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.

Figure 2-35 shows an example of data grouping in which product codes are grouped and the total of each group's ordered quantities is extracted from the stock table.

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.

Figure 2-36 shows an example of data sorting. Here, the form numbers, product codes, and ordered quantities are retrieved from the stock table, and the retrieved items are sorted by the values of the product codes in ascending order.

Figure 2-36: Data sorting example

ORDER (Order table)

FNO	C CODE	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.

Figure 2-37 shows an example of duplicated data elimination. Here, the product codes of products (with at least 10 units ordered) are retrieved from two order tables, and duplicated data is eliminated.

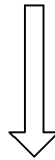
Figure 2-37: Duplicated data elimination example

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.

Figure 2-38 shows an example of outer joining. In this example, a stock table and an order table are outer joined to retrieve the products with a stock quantity of less than 100; the retrieved information includes the product codes, product names, colors, and form numbers of products.

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	QQUANTITY	...
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

```

2. Database Operations

Retrieval results

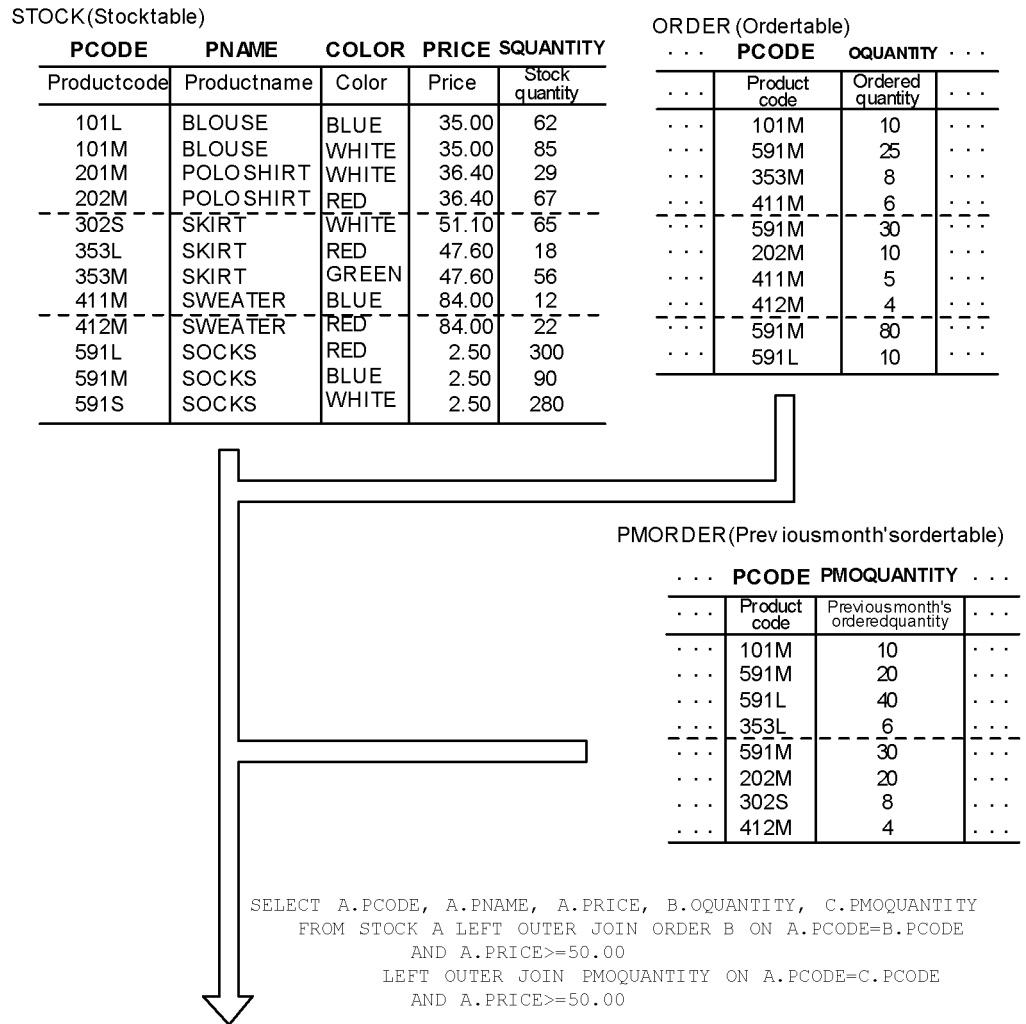
PCODE	PNAME	COLOR	FNO
Productcode	Productname	Color	Formnumber
101L	BLOUSE	BLUE	
101M	BLOUSE	WHITE	026551
201M	POLOSHIRT	WHITE	
202M	POLOSHIRT	RED	026556
302S	SKIRT	WHITE	
353L	SKIRT	RED	
353M	SKIRT	GREEN	026553
411M	SWEATER	BLUE	026554
411M	SWEATER	BLUE	026557
412M	SWEATER	RED	026558
591M	SOCKS	BLUE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

Note

The form number of products with no orders becomes the null value.

Figure 2-39 shows an example of outer joining with three or more tables. In this example, a stock table, an order table, and the previous month's order table are outer joined. For products that have a price of \$50.00 or more, the product name, price, and this and the previous month's ordered quantity are retrieved.

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



2. Database Operations

Retrieval results

PCODE	PNAME	PRICE	OQUANTITY	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 provides examples of defining and manipulating view tables. The examples are based on the stock table and the sales table shown in Figure 2-40.

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

2. Database Operations

- 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

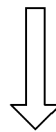
Figure 2-41 shows an example of defining a view table to limit the columns to be searched. In this example, view table `v1` is derived from a stock table in a way that the columns that can be searched will not include the color column.

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

Figure 2-42 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 a stock table and a sales table in order to determine which products at each branch have sold less than 10 items (quantity sold).

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

Figure 2-43 shows an example of defining a read-only view table. In this example, a query is used to create a read-only view table V3 from a stock table to determine the 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(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

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

V3

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

Figure 2-44 shows an example defining a view table from which duplications are eliminated. In this example, view table V4 is created from a 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

Figure 2-45 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) of Section 2.11(1) *Defining view tables*, 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	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



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

V5 (Viewed table 5)

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

(2) Manipulating a view table

Figure 2-46 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.

2.12.1 Abstract data types provided by the HiRDB Text Search Plug-in

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 8 Installation and Design Guide*.

(1) Retrieving data

Figure 2-47 shows an example of data retrieval with a plug-in. 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 a plug-in (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

Figure 2-48 shows another example of retrieval with a plug-in. 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
    
```

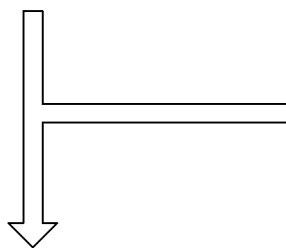
In this example, a medicine management table and a stock table are outer joined and searched. The example uses the `contains` abstract data type function to retrieve

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 a plug-in (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 a plug-in. 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 a plug-in

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 a plug-in. The example deletes the

2. Database Operations

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 a plug-in

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 a plug-in. This example inserts a row for `MEDICINE 25` into the medicine management table. 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 a plug-in

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 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 8 Installation and Design Guide*.

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

Figure 2-52 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 2-52: Example of retrieval from a table with abstract data types

STAFF NUMBER	EMPLOYEE	NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
650056		BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
670027		SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
900123		THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
920100		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

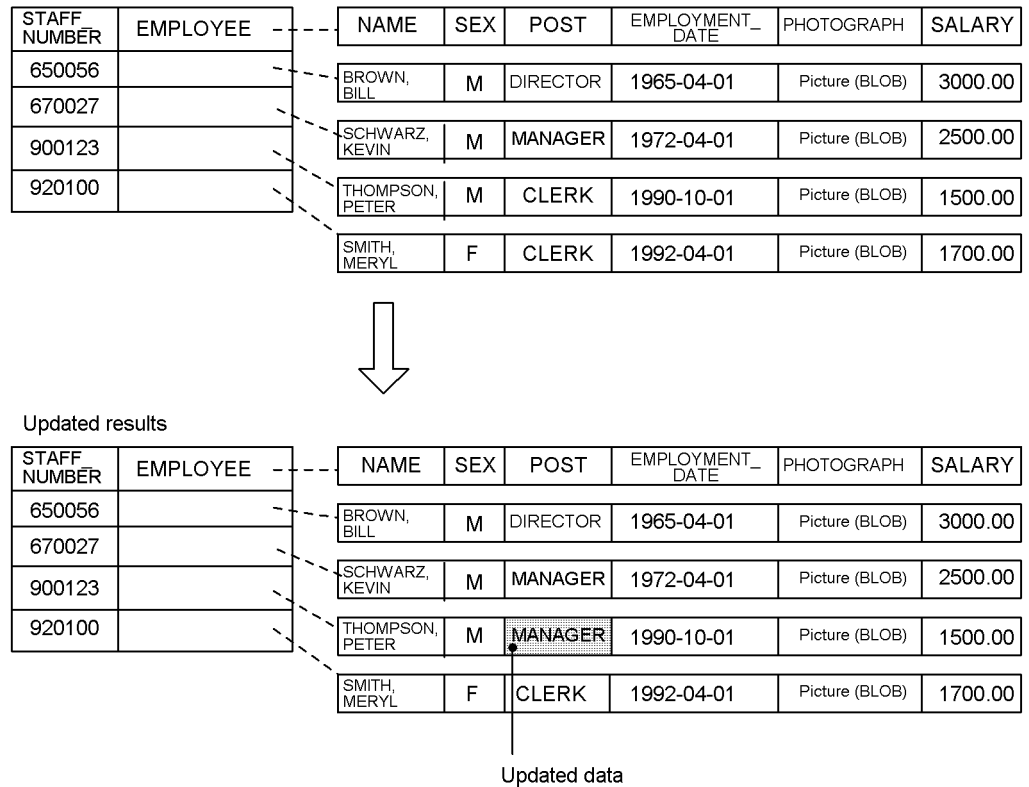
Figure 2-53 shows an example of updating a table that has abstract data types. This example updates the post of the employee with staff number 900123 to `MANAGER`. The SQL statement for updating the table can be specified as follows:

```
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 2-53: Example of updating a table with abstract data types



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

Figure 2-54 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 2-54: Example of deleting rows from a table with abstract data types

STAFF NUMBER	EMPLOYEE	NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
650056		BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
670027		SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
900123		THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
920100		SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00



Deletion results

STAFF NUMBER	EMPLOYEE	NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
650056		BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
670027		SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00

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

Figure 2-55 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 2-55: Example of inserting rows into a table with abstract data types

STAFF NUMBER	EMPLOYEE	NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
650056		BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
670027		SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
900123		THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
920100		SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00



Insertion results

STAFF NUMBER	EMPLOYEE	NAME	SEX	POST	EMPLOYMENT_DATE	PHOTOGRAPH	SALARY
650056		BROWN, BILL	M	DIRECTOR	1965-04-01	Picture (BLOB)	3000.00
670027		SCHWARZ, KEVIN	M	MANAGER	1972-04-01	Picture (BLOB)	2500.00
900123		THOMPSON, PETER	M	CLERK	1990-10-01	Picture (BLOB)	1500.00
920100		SMITH, MERYL	F	CLERK	1992-04-01	Picture (BLOB)	1700.00
950070		STONE, JANE	F	CLERK	1995-04-01	Picture (BLOB)	1400.00

Inserted row

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

Figure 3-1 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 *E. 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 8 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 8 SQL Reference* manual.

(5) Retrieval and update SQL (execution statements)

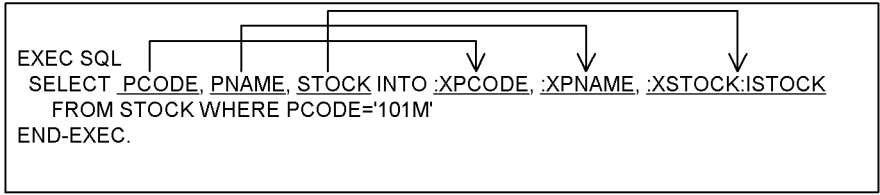
In this section, specify SQL statements for retrieving, inserting, or deleting data. For details about how to specify the individual SQL statements, see 2. *Database Operations*.

This section explains how to use embedded and indicator variables.

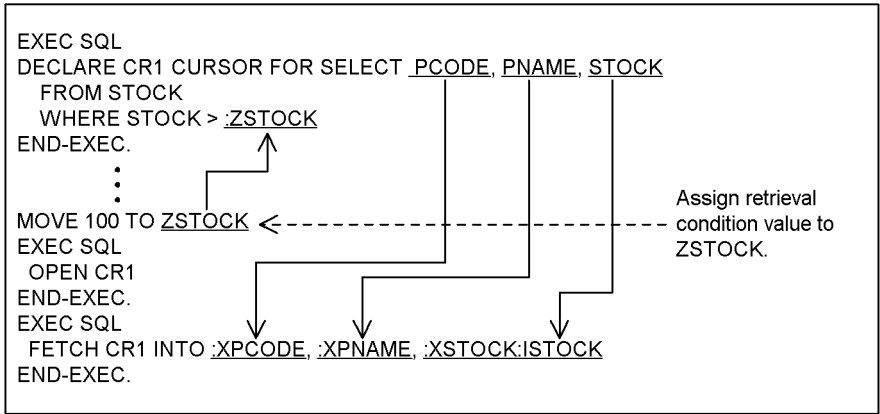
(a) Specifying embedded and indicator variables in a 1-row SELECT or FETCH statement

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>



<FETCH statement>



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).

Table 3-1 lists the UAP descriptive languages that can be used in HiRDB.

Table 3-1: UAP descriptive languages

Operating environment	Descriptive languages
HP-UX	<ul style="list-style-type: none"> • C language (Microsoft Visual C++) • C++ language (Optimizing C++) • COBOL language (COBOL85 and COBOL2002) • OOCOBOL language (OOCOBOL) *
Solaris	<ul style="list-style-type: none"> • C language • COBOL language * COBOL85, COBOL2002, and COBOL language products of other companies (MicroFocusCOBOL and SUN Japanese COBOL)
AIX 5L	<ul style="list-style-type: none"> • C language • C++ language • COBOL language (COBOL85 and COBOL2002)
Linux	<ul style="list-style-type: none"> • C language (gcc) • C++ language (GCC) • COBOL language (COBOL85 and COBOL2002) • OOCOBOL language (OOCOBOL) *
Windows	<ul style="list-style-type: none"> • 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. Table 3-2 lists 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	<ul style="list-style-type: none"> 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
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 8 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 8 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 8 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. Table 3-3 shows the UAP retrieval methods when an SQL statement is used.

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

Retrieval method		SQL statement for specifying query
Embedded UAP	Static SQL	Single-row <code>SELECT</code> statement
		Cursor declaration
	Dynamic SQL	Single-row <code>SELECT</code> statement
		Dynamic <code>SELECT</code> 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*

Table 3-4 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

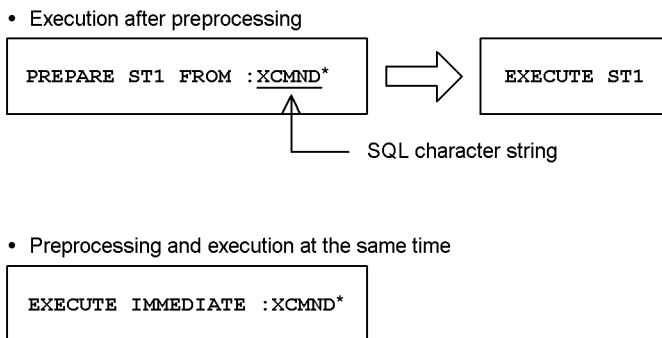
A dynamic SQL provides more flexibility in changing search conditions than a static SQL. However, dynamic SQL statements must be executed each time a condition is changed. For this reason, execution efficiency (processing efficiency) must be considered when deciding whether or not to use a dynamic SQL.

(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 8 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



*:XCMND declares any embedded variables in the embedded variable SQL declaration section. For details about embedded variables, see the *HiRDB Version 7 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	—

3. UAP Design

Type	SQL statement	PREPARE	EXECUTE IMMEDIATE
Control SQL	COMMIT	—	—
	CONNECT	—	—
	DISCONNECT	—	—
	LOCK TABLE	U ³	U
	CONNECT statement with RD-node specification ⁶	—	—
	DISCONNECT statement with RD-node specification ⁶	—	—
	ROLLBACK	—	—
	SET CONNECTION statement ⁶	—	—
	SET SESSION AUTHORIZATION statement	—	—
Definition SQL	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 ALIAS ⁶	U ³	U
	CREATE AUDIT	U ³	U
	CREATE CONNECTION SECURITY	U ³	U
	CREATE FOREIGN INDEX ⁵	U	U
	CREATE FOREIGN TABLE ⁵	U	U
	CREATE FUNCTION	U ³	U
	CREATE INDEX	U ³	U
	CREATE PROCEDURE	U ³	U
	CREATE SCHEMA	U ³	U

Type	SQL statement	PREPARE	EXECUTE IMMEDIATE
	CREATE SERVER ⁵	U	U
	CREATE TABLE	U ³	U
	CREATE TRIGGER	U ³	U
	CREATE TYPE	U ³	U
	CREATE USER MAPPING ⁵	U	U
	CREATE VIEW	U ³	U
	DROP ALIAS ⁶	U ³	U
	DROP AUDIT	U	U
	DROP CONNECTION SECURITY	U ³	U
	DROP DATA TYPE	U ³	U
	DROP FOREIGN INDEX ⁵	U	U
	DROP FOREIGN TABLE ⁵	U	U
	DROP FUNCTION	U ³	U
	DROP INDEX	U ³	U
	DROP PROCEDURE	U ³	U
	DROP SCHEMA	U ³	U
	DROP SERVER ⁵	U	U
	DROP TABLE	U ³	U
	DROP TRIGGER	U ³	U
	DROP USER MAPPING ⁵	U	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 8 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.

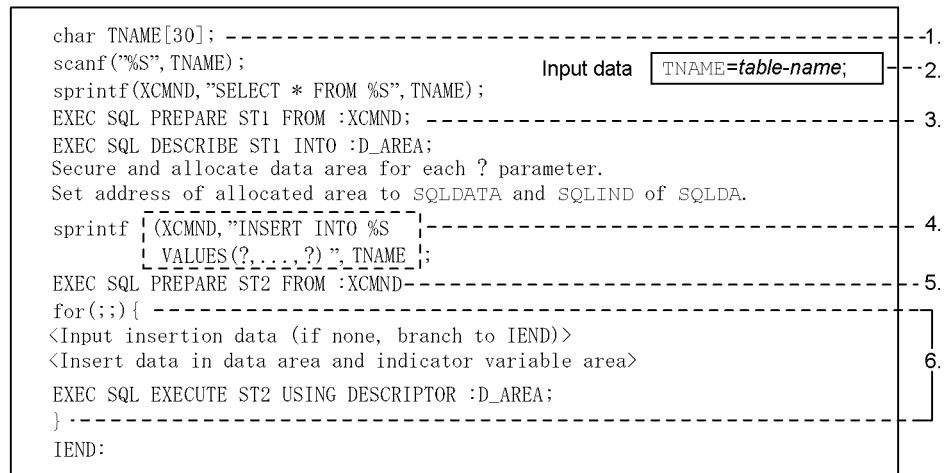
⁴ Executed by the OPEN, FETCH, or CLOSE statement.

⁵ Can be used if HiRDB External Data Access is installed.

⁶ This SQL statement is applicable to the UNIX version only.

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

**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.

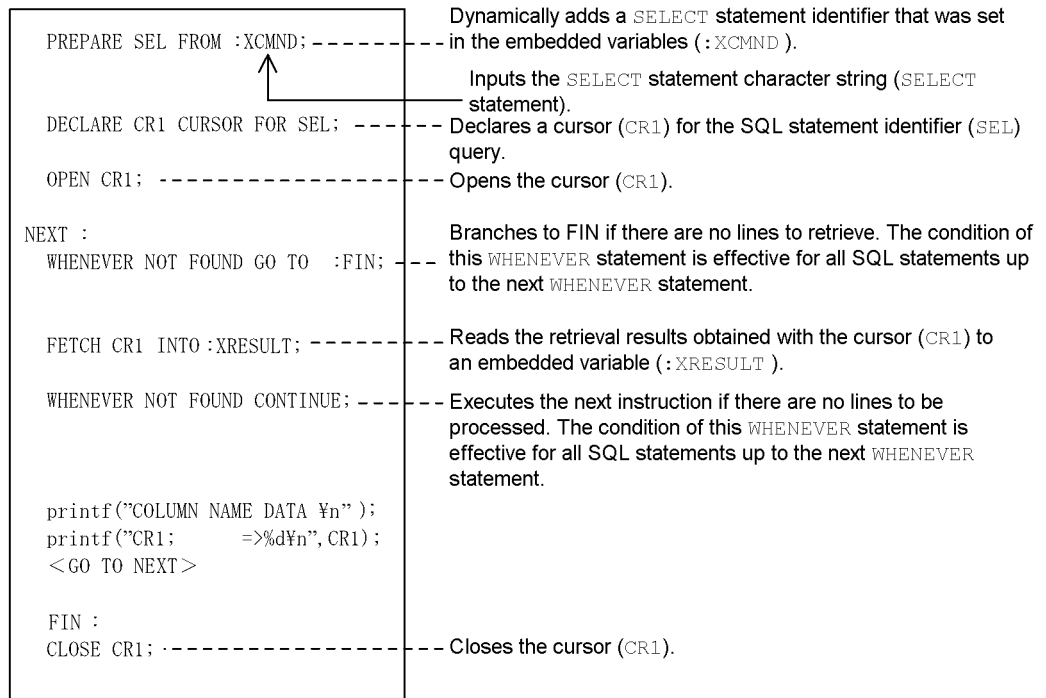
(b) Using the EXECUTE statement and the EXECUTE IMMEDIATE statement

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



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 8 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

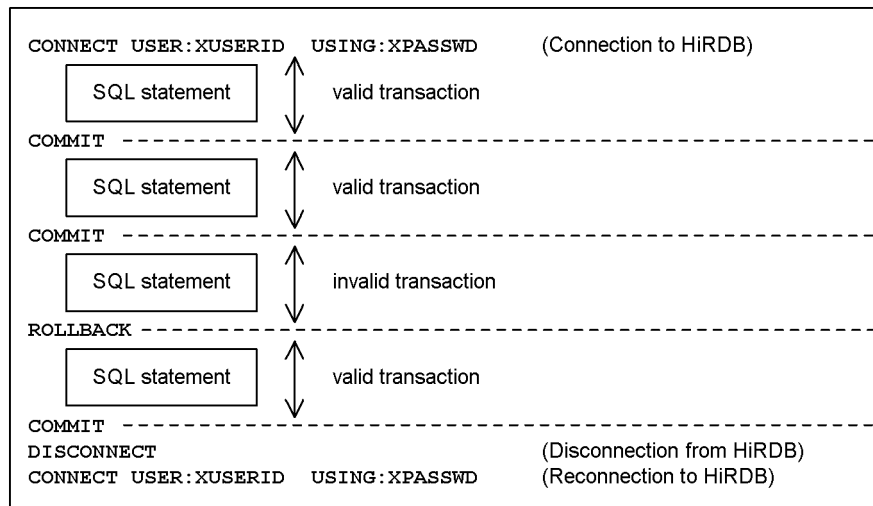
Executing the `CONNECT` statement connects a UAP to a HiRDB system, and executing the `DISCONNECT` statement disconnects them.

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.

Figure 3-6 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

Table 3-6 explains setting synchronization points and handling 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.

Table 3-7 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

PDXAMODE specification	OpenTP1 specification		Scope of LOCK TABLE UNTIL DISCONNECT		
0	trn_rm_open_close_scope=process		Effective until Resource Manager is closed.		
	trn_rm_open_close_scope=transaction		Effective within a global transaction.		
1	trn_rm_open_close_scope=process	-d option specified in trnstring operand		Effective until the Resource Manager is closed.	
		-d option not specified in trnstring operand	A single AP comprises a global transaction in the OpenTP1 system.		
			Multiple APs comprise a global transaction in the OpenTP1 system.		A single AP links to the HiRDB XA library.
		Multiple APs link to the HiRDB XA library.	Effective within the global transaction.		
trn_rm_open_close_scope=transaction					

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

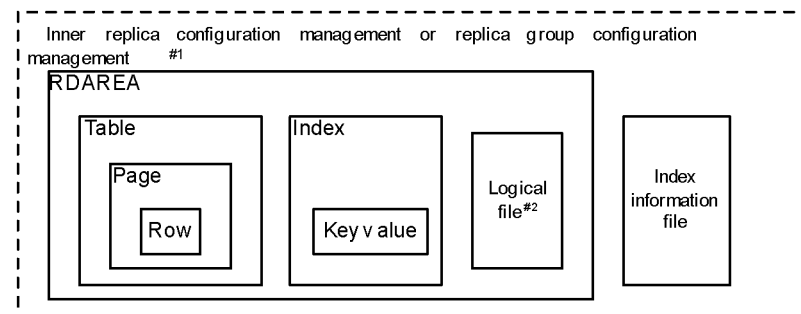
(1) Locked resources and inclusive relationships

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. Figure 3-7 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.

When updatable online reorganization is executed, the highest locked resource is the inner replica configuration management or the replica group configuration management. If the inner replica configuration management cannot be locked, the replica group configuration management is locked.

When an RDAREA that is not defined as a replica RDAREA is accessed, that RDAREA is locked.

#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 8 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 8 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. Table 3-8 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

Mode	SR	PR	SU	PU	EX
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.

Table 3-9 shows the lock mode transition rules.

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					Table	NO WAIT table	
		Inner replica config ⁶	Replica group config ⁸	RDAREA					
For tables	For IX			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}		SR	SR	SU		EX	SU	—	
Addition ¹		SR	SR	SU		EX	SU	—	
Deletion ¹		SR	SR	SU		—	SU	—	
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 ^{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}		SR	SR	SU		—	EX	EX	
Table definition change ¹³		SR ⁹	SR ⁹	SU ⁷		—	EX	EX	

— : 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 8 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.

⁷ If an RDAREA is added or is altered with the free space reuse facility, the RDAREA is locked.

⁸ When updatable online reorganization is executed, the replica group containing the RDAREA to be processed is locked.

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.

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	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}		—	EX	—	EX	EX	EX
Addition ¹		—	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}		—	—	—	—	—	—
Table definition change ⁷		—	—	—	—	—	—

— : 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 8 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.

Table 3-12: Typical lock mode combinations (page locking) (1/2)

SQL statement and execution environment		Resource							
		Higher level ----- Lower level					Table	NO WAIT table	
		Inner replica config ⁶	Replica group config ⁸	RDAREA					
For tables	For IX			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}		SR	SR	SU		EX	SU	—	
Addition ¹		SR	SR	SU		EX	SU	—	
Deletion ¹		SR	SR	SU		—	SU	—	
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 ^{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}		SR	SR	SU		—	EX	EX	
Table definition change ¹³		SR ⁹	SR ⁹	SU ⁷		—	EX	EX	

— : 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 8 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.

⁷ If an RDAREA is added or is altered with the free space reuse facility, the RDAREA is locked.

⁸ When updatable online reorganization is executed, the replica group containing the RDAREA to be processed is locked.

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.

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	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}		—	EX	EX	—	EX	EX
Addition ¹		—	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}		—	—	—	—	—	—
Table definition change ⁷		—	—	—	—	—	—

— : 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 8 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.

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	NO WAIT table
For tables	For IX			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}		SR	SR	SU			EX	SU

SQL statement and execution environment			Resource						
			Higher level -----			Lower level			
			Inner replica config ⁵	Replica group config ⁷	RDAREA			Table	NO WAIT table
For tables	For IX	Last HiRDB file ⁴							
Addition ¹⁰			SR	SR	SU		EX	SU	—
Deletion ¹⁰			SR	SR	SU		—	SU	—
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}			SR	SR	SU		—	EX	EX
Table definition change ¹³			SR ⁸	SR ⁸	SU ⁶		—	EX	EX

— : 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.

⁶ If an RDAREA is added or is altered with the free space reusage facility, the RDAREA is locked.

⁷ When updatable online reorganization is executed, the replica group containing the RDAREA to be processed is locked.

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 8 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.

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	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}		—	EX	—, EX ⁴ , 5	EX, — 4, 5	—	EX
Addition ⁶		—	EX	—, EX ⁴ , 5	EX, — 4, 5	—	EX
Deletion ⁶		—	—	—, EX ⁴ , 5	EX, — 4, 5	—	EX
LOCK statement	SHARE specified ⁷		—	—	—	—	—
	EXCLUSIVE specified	Unshared table	—	—	—	—	—
		Shared table ⁷	—	—	—	—	—
Table deletion ^{1, 9}		—	—	—	—	—	—
Index	Definition ⁹		—	—	—	—	—
	Deletion ^{2, 9}		EX	—	—	—	—

SQL statement and execution environment	Resource					
	Higher level -----			Lower level		
	Index	Index information file ³	Page	Row	Key value	Logical file
Deletion of all rows ^{1,9}	—	—	—	—	—	—
Table definition change ⁹	—	—	—	—	—	—

— : 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 8 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.

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	NO WAIT table
	For tables ^{#2}	For indexes	For LOB	Last HiRDB file		
Deletion of all rows	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 (pdroorg) ^{#3}	SU	—	—	—	EX	EX
Database structure modification utility (pdmod)	SU	—	—	—	EX	EX
Integrity check utility (pdconstck) ^{#3}	SU	—	—	—	EX	EX
Reflection command for online reorganization (pdorend) ^{#3}	SU	—	—	—	EX	EX

— : 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.

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	—	—	—	—	—	—
Changing a table's definition (changing the storage partitioning conditions)	—	—	—	—	—	—
Database load utility (<code>pdload</code>)#2	—	—	—	—	—	—
Database reorganization utility (<code>pdroorg</code>)#2	—	—	—	—	—	—
Database structure modification utility (<code>pdmod</code>)	—	—	—	—	—	—
Integrity check utility (<code>pdconstck</code>)#2	—	—	—	—	—	—
Reflection command for online reorganization (<code>pdorend</code>)#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.

(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					Table	NO WAIT table
		Inner replica config *	Replica group config *	RDAREA				
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	—
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	NO WAIT table
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		—	—	—	—	—	—

SQL statement		Resource					
		Higher level ----- Lower level					
		Index	Index information file	Page	Row	Key value	Logical file
Index	Definition	—	—	—	—	—	—
	Deletion	—	—	—	—	—	—
Deletion of all rows		—	—	—	—	—	—
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.

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.

(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 8 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

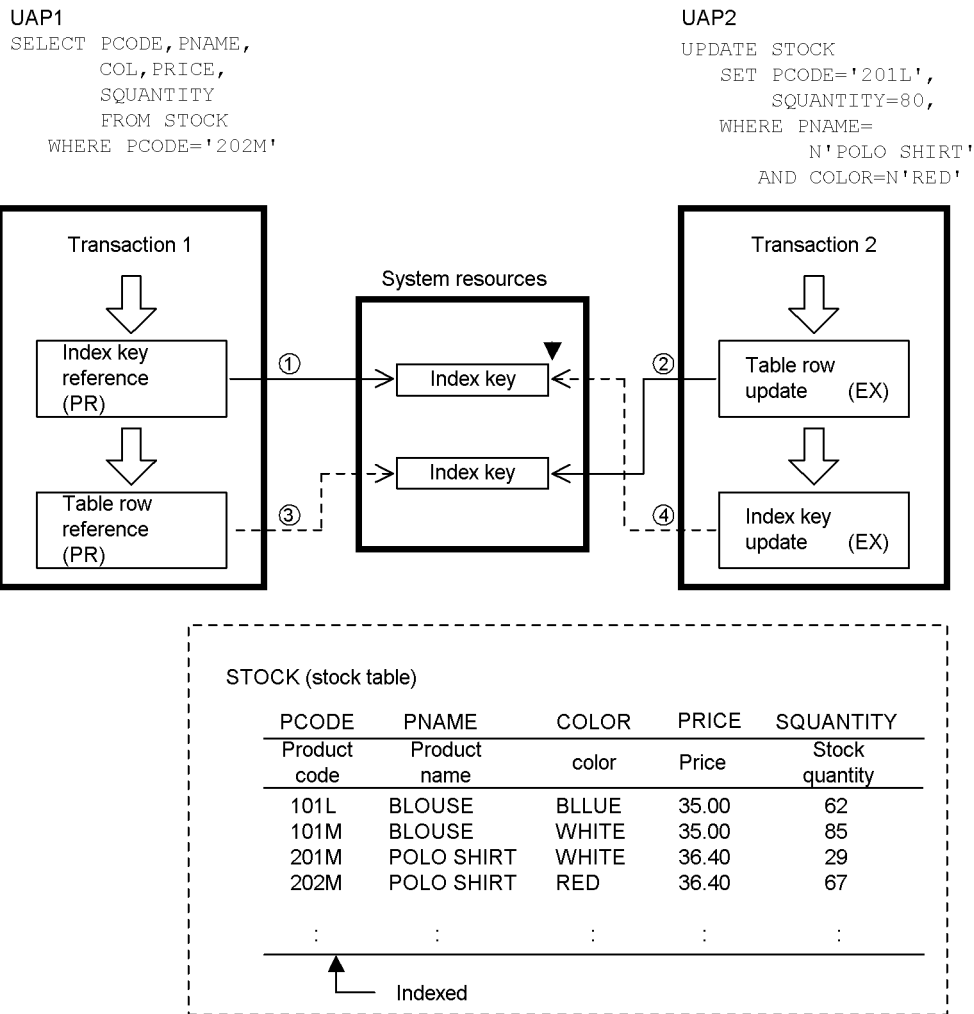
(1) Causes of deadlock

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.

Figure 3-8 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

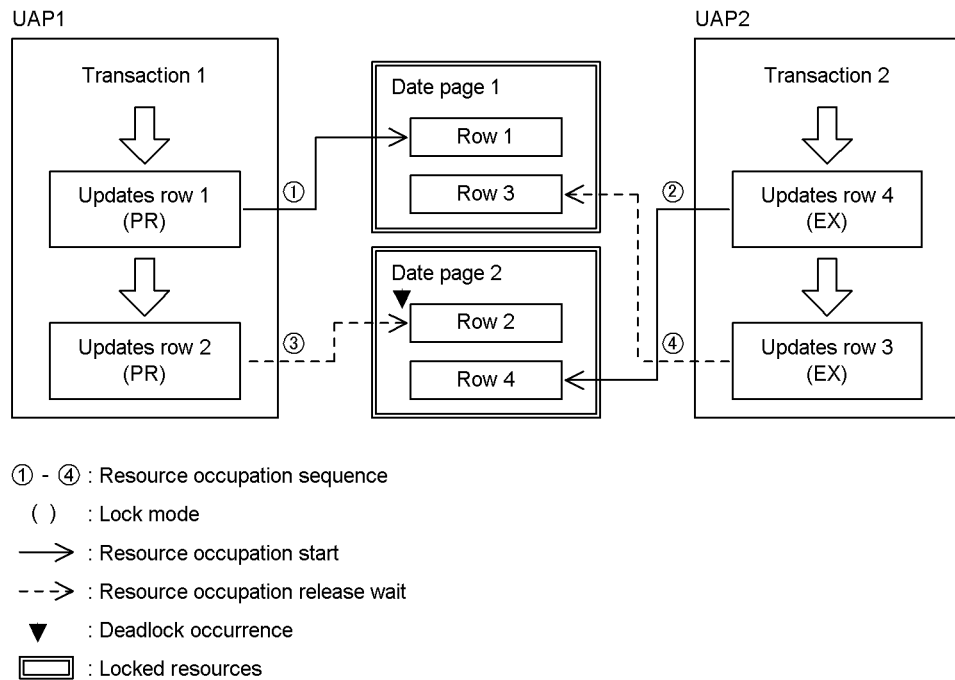


- ① -④: Resource occupation sequence
- () : Lock mode
- : Resource occupation start
- > : Resource occupation release wait
- ▼ : Deadlock occurrence

With page-locking, there are situations in which deadlock cannot be prevented even though UAP access procedures are standardized.

Figure 3-9 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.

Figure 3-10 shows an example of global deadlock, in which two transactions try to execute retrieval and updating in reverse order on tables stored in 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

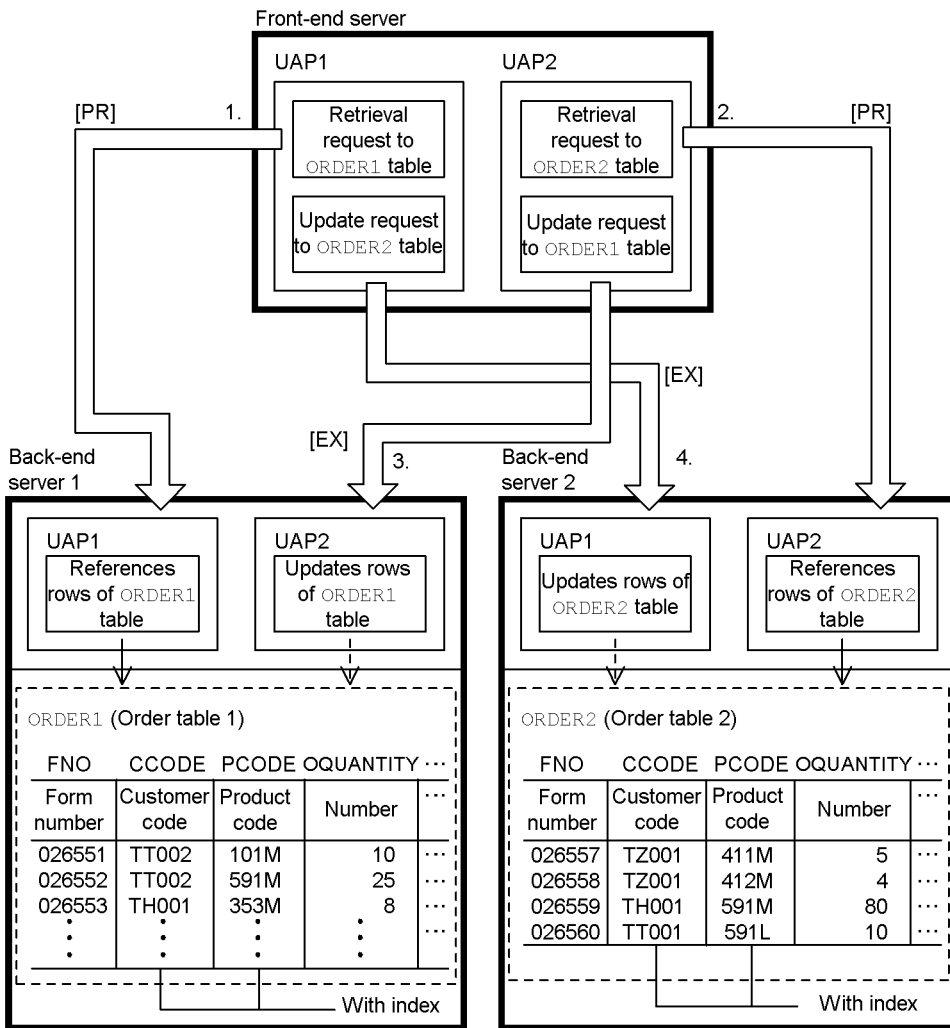
```

[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'
    
```



1. to 4.: Processing request sequence → : Processing request sequence
 [] : Lock mode -> : 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) **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. Table 3-22 shows the major types of deadlocks and their countermeasures.

Table 3-22: Deadlocks and their countermeasures

Deadlocked resources	Cause	Countermeasure
Row and row	UAP access sequence (shown in Figure 3-10)	<ul style="list-style-type: none"> • Standardize UAP access sequence. • Use <code>LOCK TABLE</code> 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)	<ul style="list-style-type: none"> • Do not update the retrieved row. • Do not update the values in a column to the same value. • Minimize index definition. • Use <code>LOCK TABLE</code> to lock the table. • Perform <code>NOWAIT</code> retrieval. • Re-execute UAP after deadlock occurs. • Apply non-locking of index key values
Index key and index key	UAP access sequence	<ul style="list-style-type: none"> • Standardize UAP access sequence. • Minimize index definition. • Use <code>LOCK TABLE</code> to lock the table. • Perform <code>NOWAIT</code> 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)	<ul style="list-style-type: none"> • Use <code>ALTER TABLE</code> to change page locking to row locking.

(4) **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 8 System Definition manual*.

(5) 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 8 System Definition* manual.
- Apply non-locking of index key values.

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 *3.4.8 Lock sequence based on SQL statement and index types*.

(6) 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 8 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.

3.4.5 Unlocked conditional search

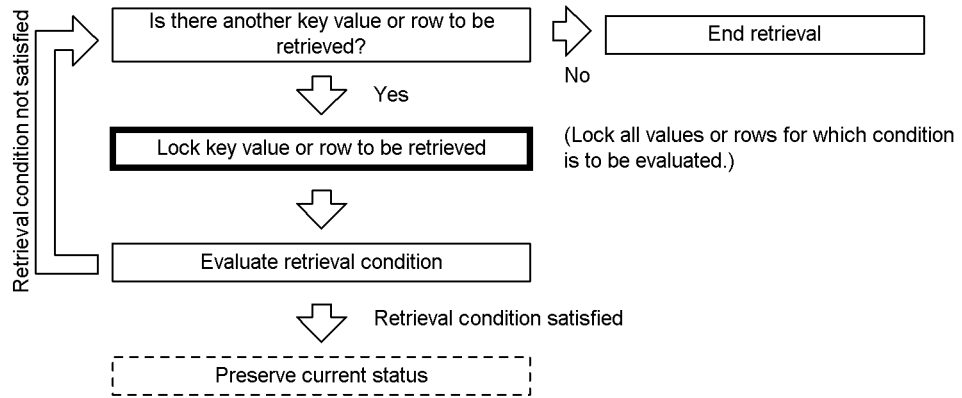
Unlocked conditional search does not lock items whenever retrieval processing is underway, but instead locks only those rows and key values that satisfy the retrieval condition. Unlocked conditional search can reduce the retrieval time compared to ordinary retrieval processing, because rows and key values 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.

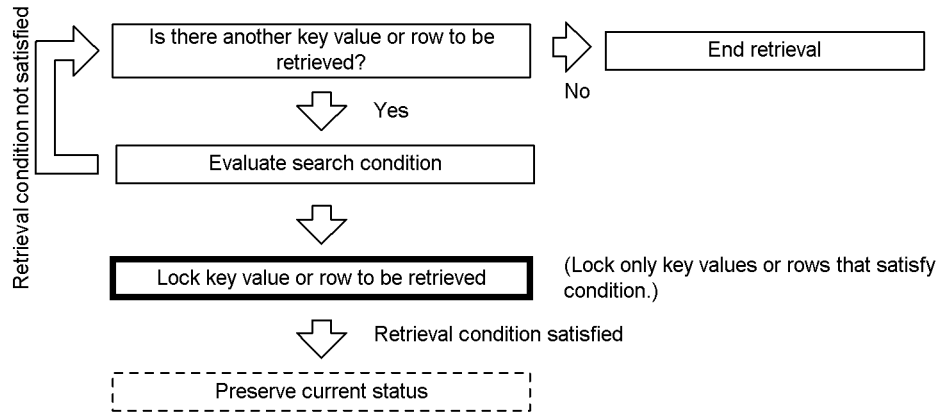
Figure 3-11 shows the processing flows of an ordinary retrieval and of a retrieval using unlocked conditional search.

Figure 3-11: Processing flows of an ordinary retrieval and of a retrieval using an unlocked conditional search

• Ordinary retrieval



• Retrieval using an unlocked conditional search



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 a lock (such as `WITHOUT LOCK NOWAIT`)
- Retrieval that uses the direct product of two tables or a `quantified` predicate as the retrieval condition
- Retrieval that uses an index when non-locking of index keys is applied

Because unlocked conditional search performs a conditional search without locking items, if it is executed simultaneously with an updating transaction, the result may not be the same as would have been the case with an ordinary retrieval.

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 data update, see the *HiRDB Version 8 Installation and Design Guide*.

(2) Specifying non-locking of index key values

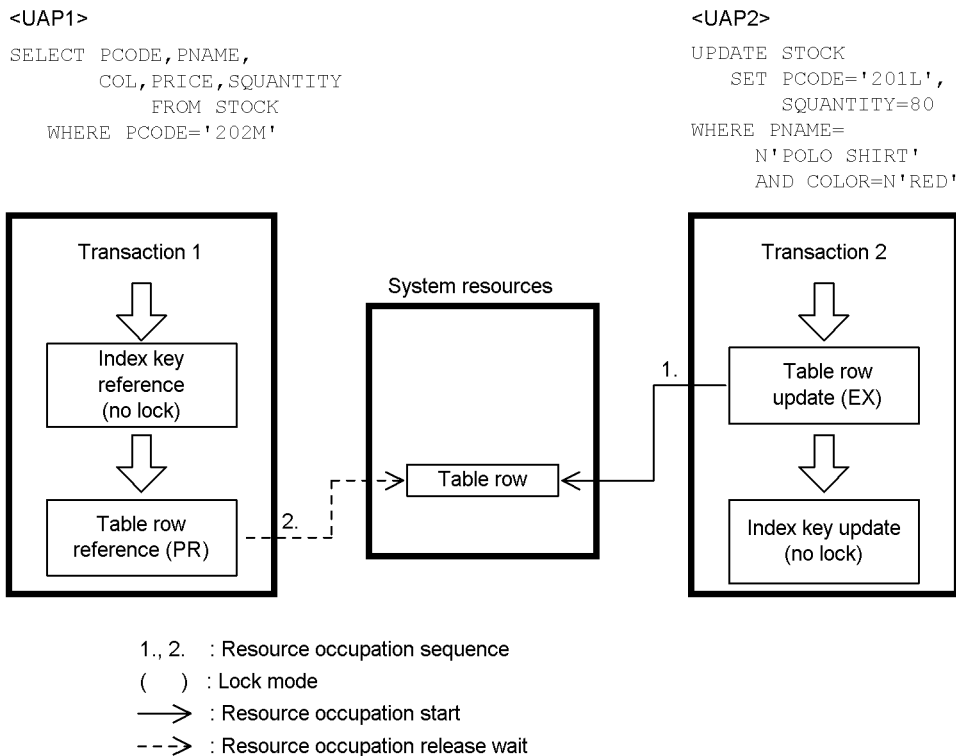
To apply non-locking of index key values, specify `NONE` in the system definition for the `pd_indexlock_mode` operand. For details about the `pd_indexlock_mode` operand, see the *HiRDB Version 8 System Definition* manual.

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. Figure 3-12 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



(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 during insertion or update processing of table data for which the uniqueness constraint was specified, apply non-locking of index key values. Also apply non-locking of index key values to give priority to attempting insertion or update processing even if waiting is involved.

(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 Lock and suppression implementable with a UAP

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 if `WITHOUT LOCK NOWAIT` is specified, a table undergoing data processing by the database load utility (except when `nowait=yes` is specified in the `option` statement) or the database reorganization utility (except when `-k unld` is specified) cannot be searched.

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.

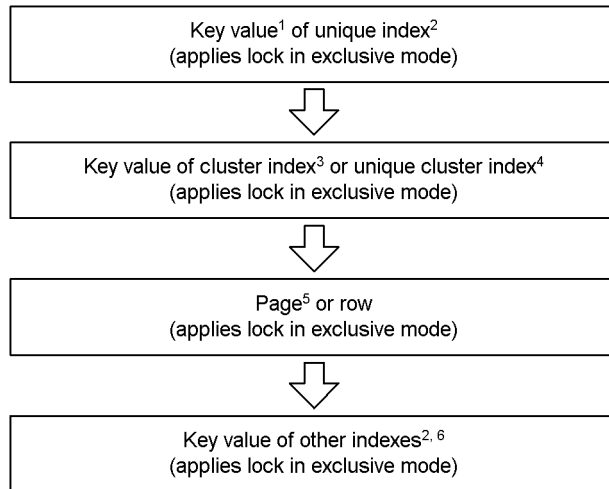
(4) Notes

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.

3.4.8 Lock sequence based on SQL statement and index types

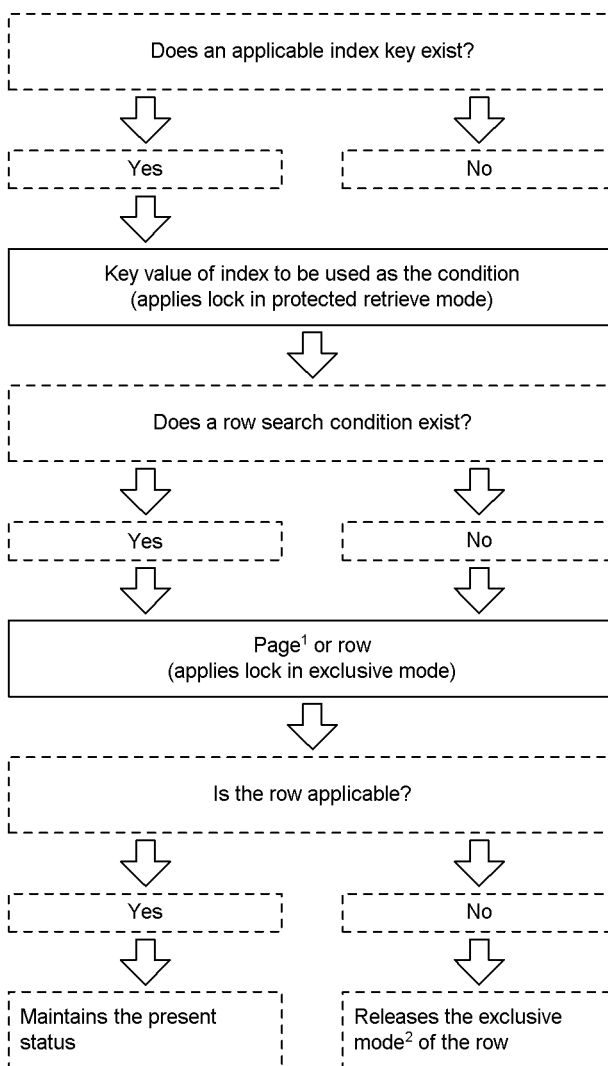
(1) Lock sequence of data manipulation SQL statements for index key values and data page rows

(a) INSERT statement



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.

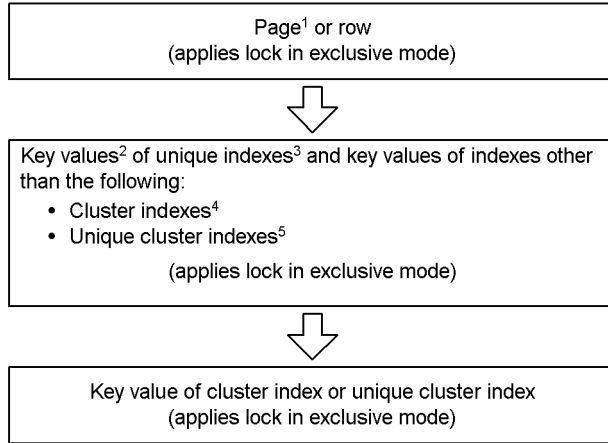
(b) DELETE statement that does not use a cursor or UPDATE statement to search for data matching a condition



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



¹ Equivalent to page-locked table.

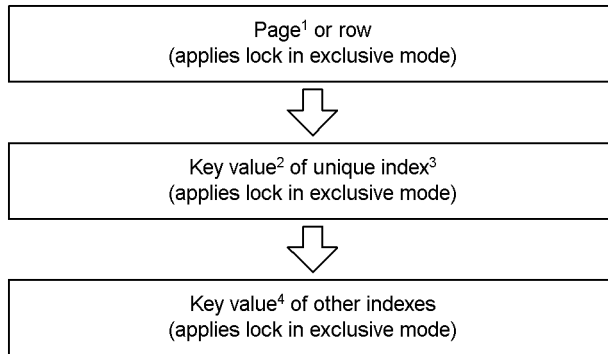
² If multiple key values are defined, they will be processed in the order opposite from the definition order.

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



¹ Equivalent to page-locked table.

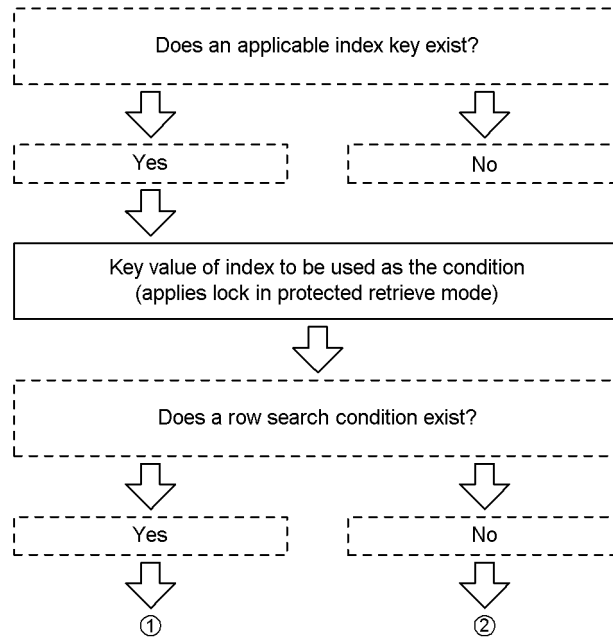
² If multiple key values are defined, they will be processed in the order opposite from the definition order.

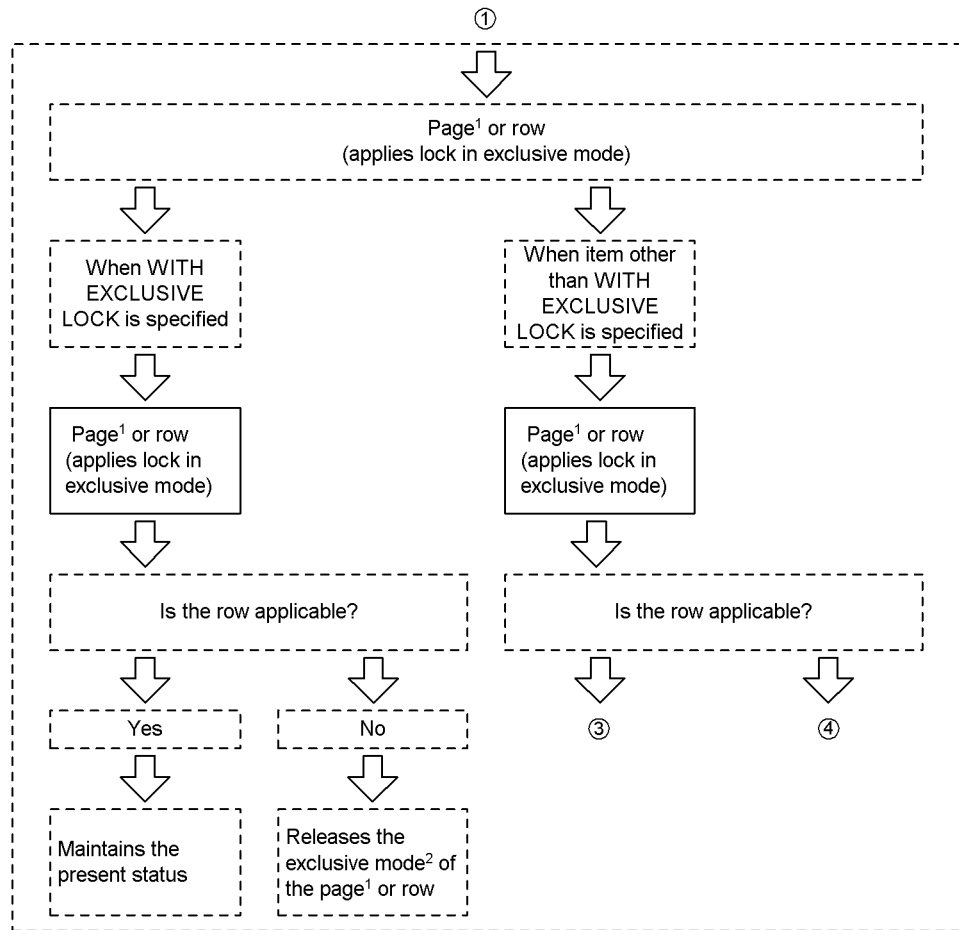
³ Index created for a column for which the uniqueness constraint (UNIQUE) is specified.

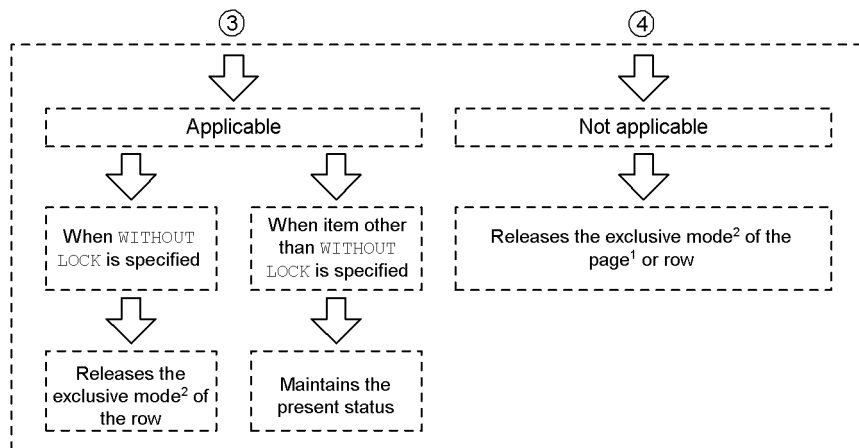
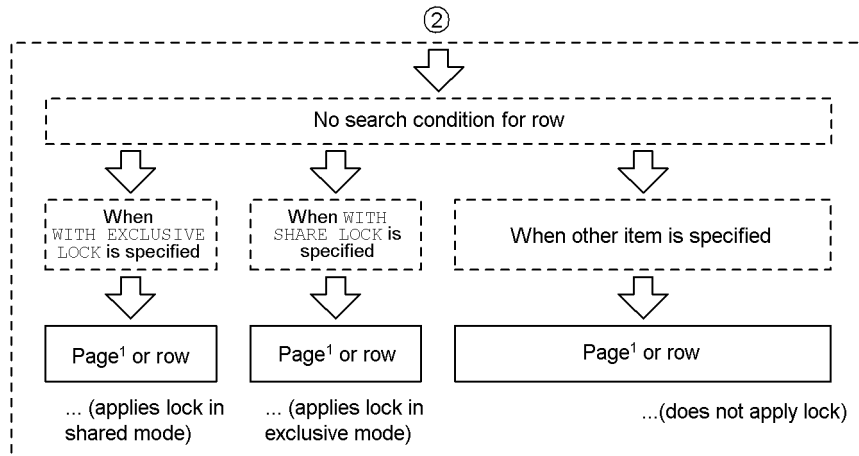
⁴ Index other than a unique index, cluster index⁵, or unique cluster index⁶.

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



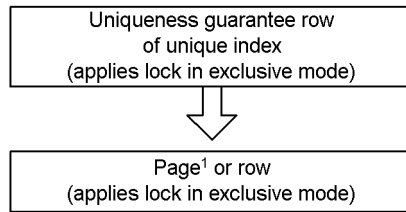


¹ Equivalent to page-locked table.

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.

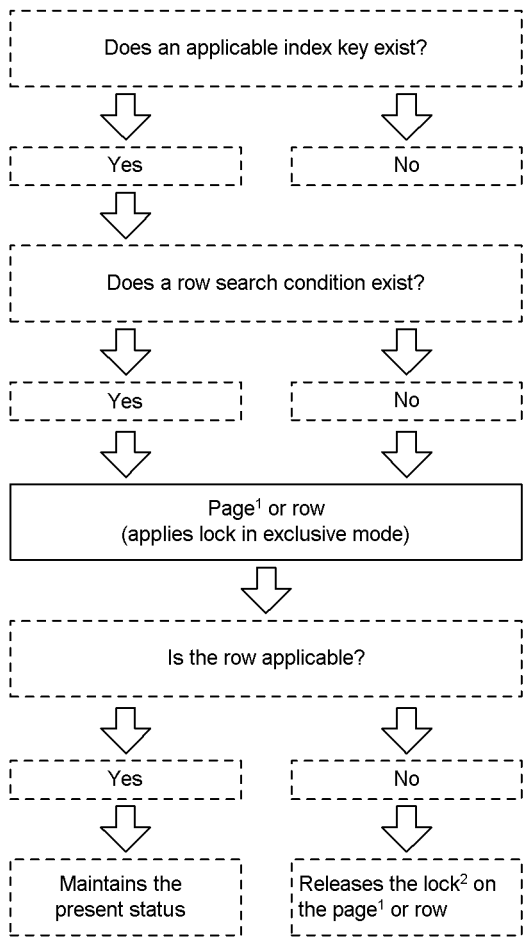
(2) Lock sequence of data manipulation SQL statements when non-locking of index key values is used

(a) INSERT statement



¹ Equivalent to page-locked table.

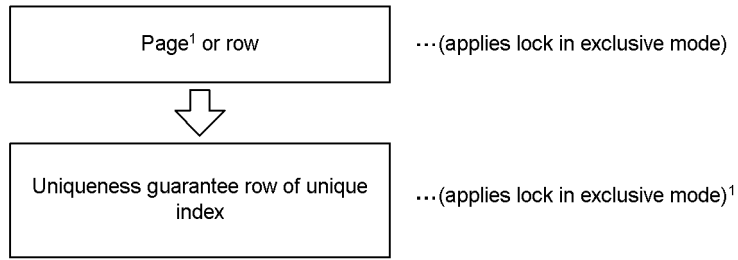
(b) DELETE statement that does not use a cursor or UPDATE statement to search for data matching a condition



¹ Equivalent to page-locked table.

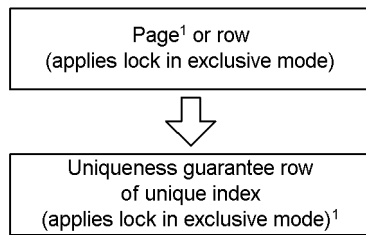
² When the exclusive mode lock is applied to a row that has already been locked in the protected retrieve mode, the exclusive mode lock is not released, even when the processing in the exclusive mode terminates first. In this case, the lock is released only after both the processing in the protected retrieve mode and the processing in the exclusive mode have terminated.

(c) DELETE statement that uses a cursor



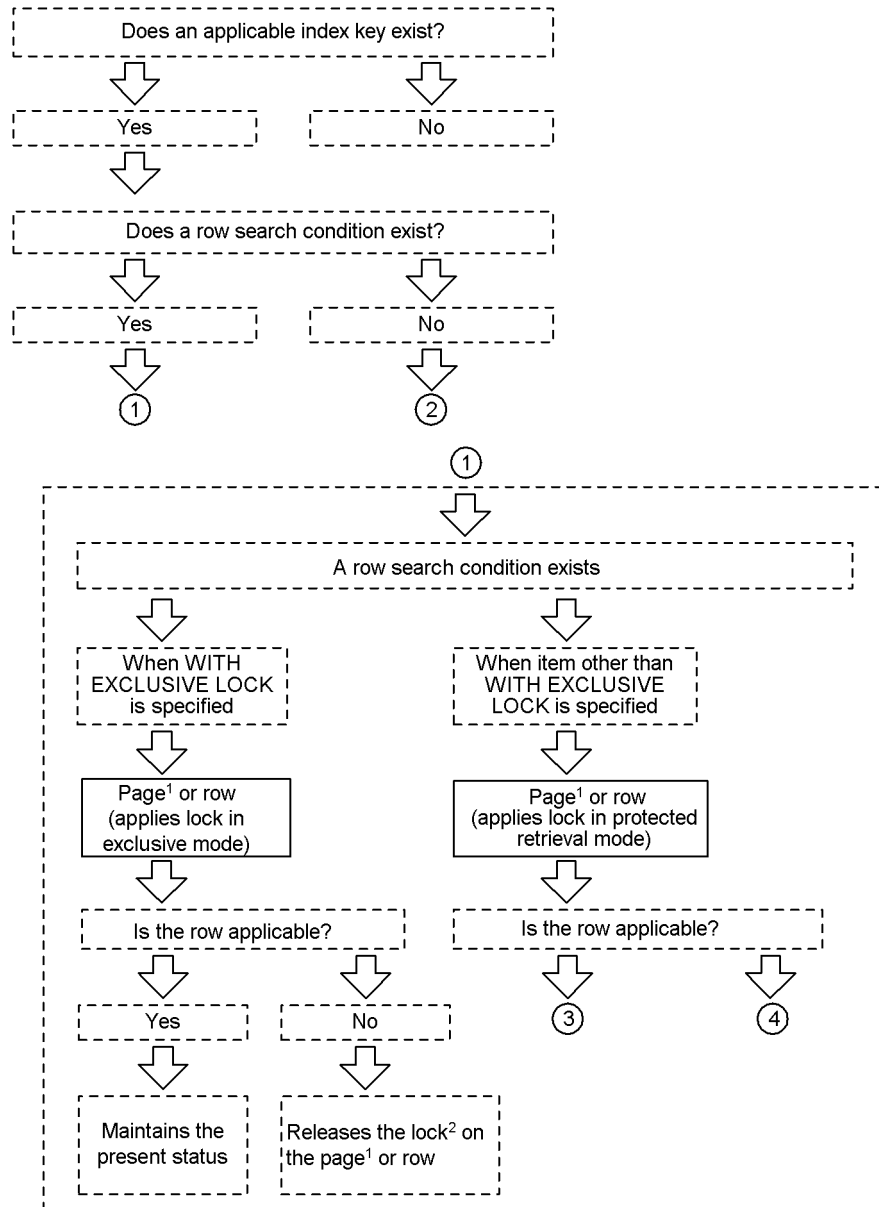
¹ Equivalent to page-locked table.

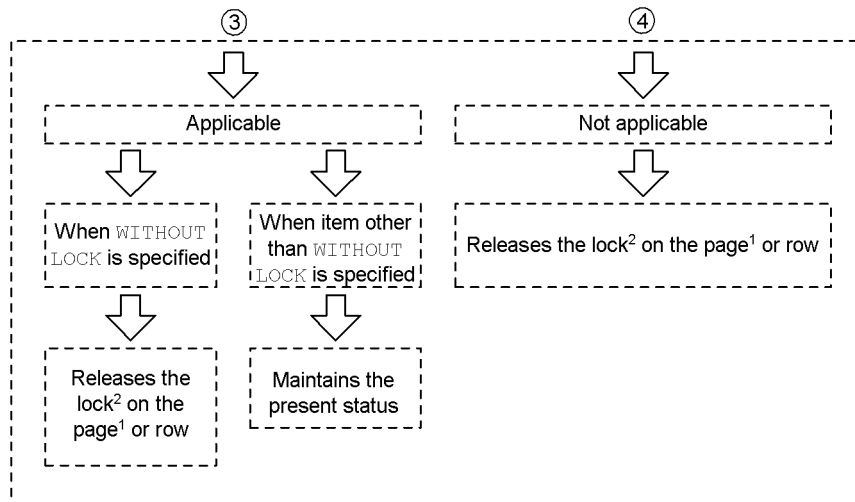
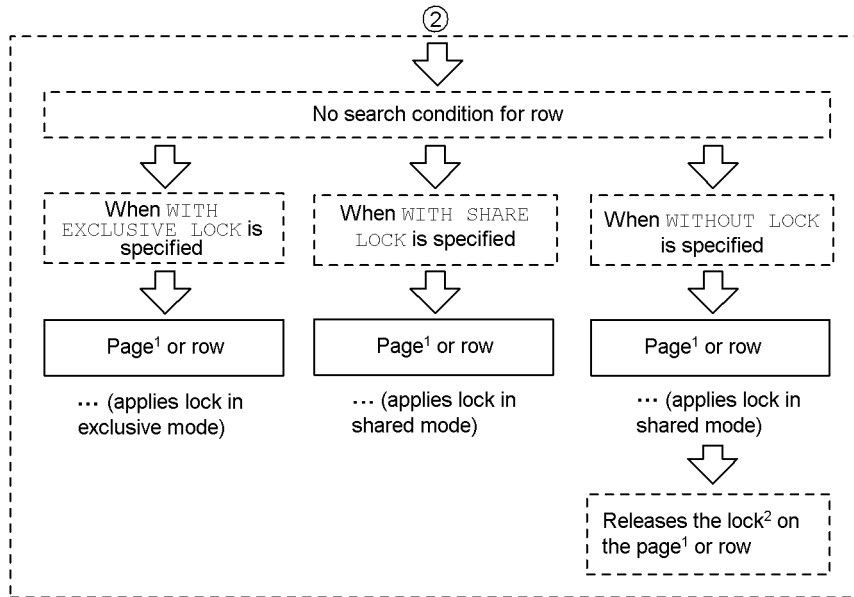
(d) UPDATE statement that uses a cursor



¹ Equivalent to page-locked table

(e) **SELECT or FETCH statement**





¹ Equivalent to page-locked table.

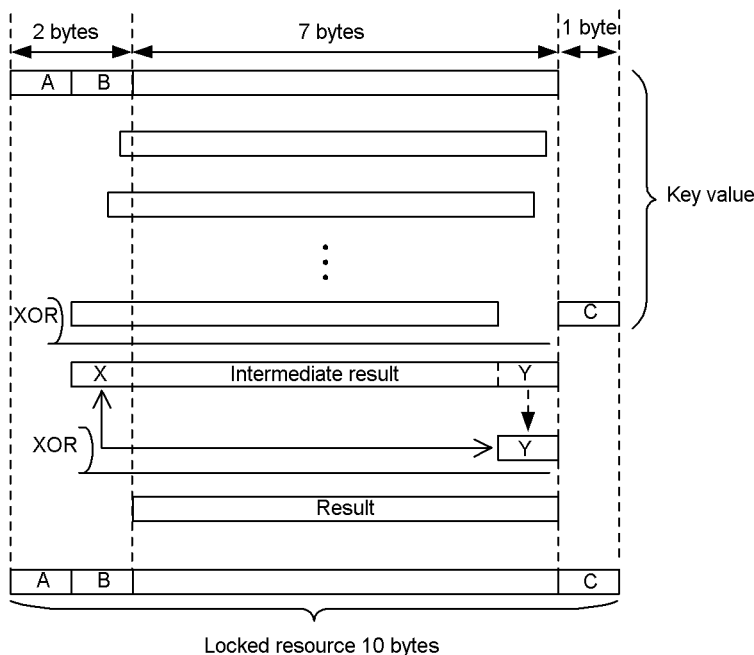
² When the exclusive mode lock is applied to a row that has already been locked in shared mode, the exclusive mode is not released, even when the exclusive mode terminates first. In this case, the exclusive mode is released only after both the processing in the shared mode and the processing in the exclusive mode have terminated.

3.4.9 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 8 System Definition* manual.

Figure 3-13 shows how a key value locked resource is created when `TYPE1` is specified in the `pd_key_resource_type` operand. Figure 3-14 shows how a key value locked resource is created when `TYPE2` is specified.

Figure 3-13: 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.

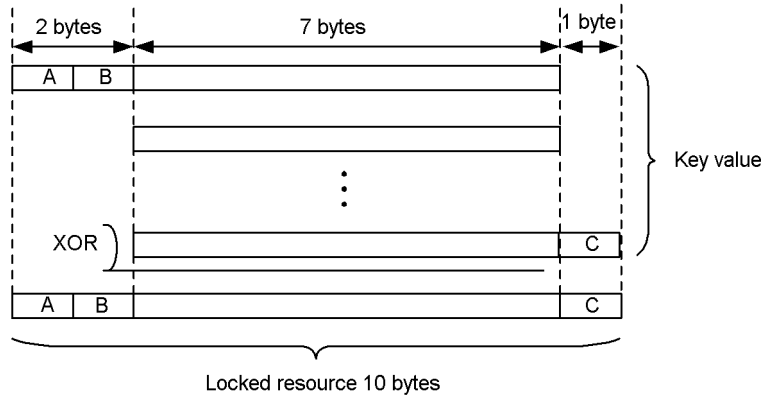
If the key value length exceeds 10 bytes, the system removes the first two bytes and the last byte of the key value, extracts the remaining data in 7-byte units, and applies `exclusive-OR` while bit-shifting the units. The bit shift operation logically shifts each unit by the remainder after the extraction 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 3-14: 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.

Table 3-23 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 3-23: 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	<code>FOR READ ONLY</code> clause specified	NP	NP	Y	Y	Y	Y	Y	Y	Y	Y

3. UAP Design

Condition		Operation that does not use a cursor										
		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				
Specification of cursor updatability		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	Y
		NP	P	Y	Y	Y	Y	Y	Y	Y	Y	Y
		P	NP	Y	Y	Y	Y	Y	Y	Y	Y	Y
		P	P	Y	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		

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.

Table 3-24 lists the issues to be considered when specifying the `FOR UPDATE` (or `FOR UPDATE OF`) clause and the `FOR READ ONLY` clause.

Table 3-24: Specifying FOR UPDATE and FOR READ ONLY clauses

Application		Consideration
<code>FOR UPDATE</code> 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 <code>FETCH</code> ; creation of this work table involves overhead during retrieval.
<code>FOR UPDATE OF</code> 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 <code>FETCH</code> ; creation of this work table involves overhead during retrieval.
<code>FOR READ ONLY</code> 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 <code>FETCH</code> 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 `FETCH` is executed, the lock mode that has priority is the lock option specified in the cursor declaration. 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.

Table 3-25 shows the relationships between the lock option specified during cursor declaration or dynamic SELECT statement preprocessing and the lock option specified during table operations.

Table 3-25: 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
	WITHOUT LOCK NOWAIT			No	WITHOUT LOCK NOWAIT
				Yes	Error

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
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.

Because HiRDB fetches all rows targeted for retrieval during the first `FETCH` statement when a cursor is used, a lock buffer shortage may occur. This problem is caused by an increase in the overhead associated with creation of the internal table. Therefore, before a cursor is used, the target rows should be narrowed using a search condition.

If the target rows cannot be narrowed, consider other measures to suppress locking, such as changing the unit of locking. For details about lock suppression, see *3.4.7 Lock and suppression implementable with a UAP*.

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. However, before the cursor is opened, a `LOCK` statement with `UNTIL DISCONNECT` specified must be issued to lock the table. 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

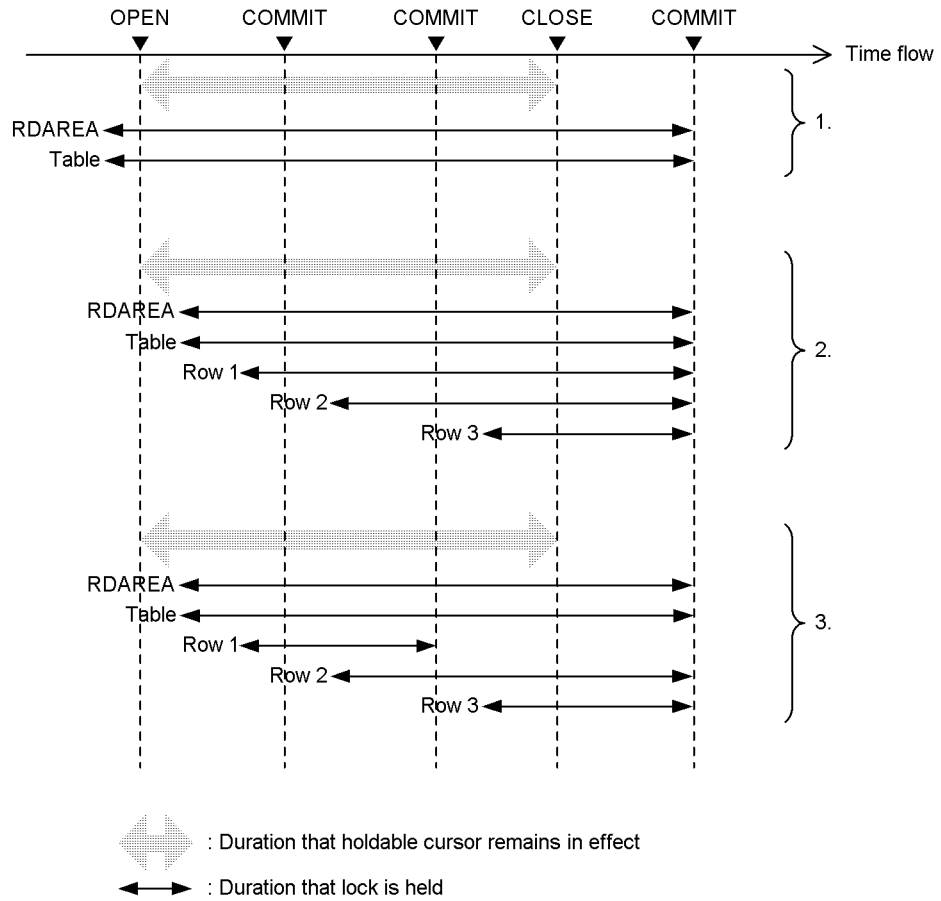
`COMMIT` statement can be executed while retrieving or updating a large amount of data. Moreover, because a `COMMIT` statement can be executed while keeping the cursor open, a synchronization point can be activated even when a large amount of data is being retrieved or updated (i.e., when a transaction 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 the work table file and freeing of the work buffer for the work table take place during commit processing after the holdable cursor for which the work table file was created is 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_holder` operand of the HiRDB system definition.

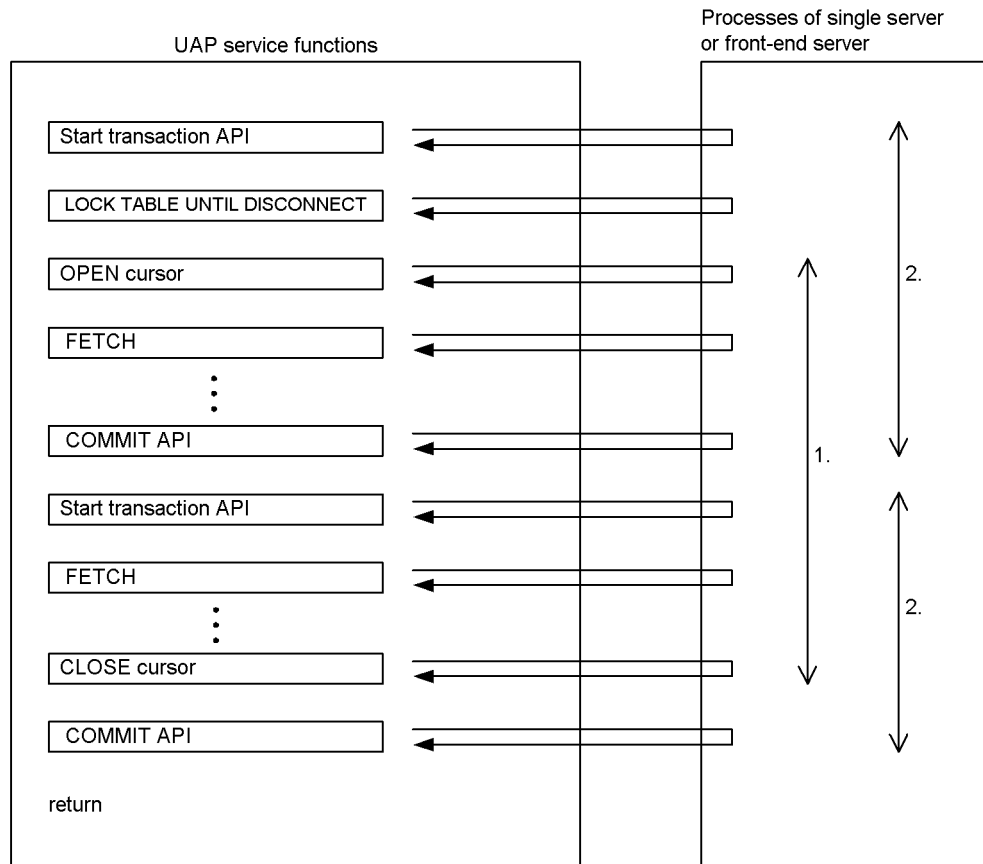
For a UAP executing in an OLTP environment to use a holdable cursor, the following conditions must be satisfied:

- 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 process.

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] ; .....I
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;
:

```

3. UAP Design

```
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.

Table 3-26 shows the SQL statement execution status indicated by the values set in the variables.

Table 3-26: 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	'Δ'	—	'00000'
	With warning ⁴	0	'w'	—	'01nnn' ¹ (<i>nnn</i> ≠ R00)
		>0 (≠ 100, 110)	—	—	'R01R00'
	Without data ³	110	—	—	'R2000'
No data		100	—	—	'02000'
Termination with error	Without implicit rollback	<0	—	'Δ'	'mmnnn' ²
	With implicit rollback	<0	'w'	'w'	'40nnn' ¹

mm: Class

nnn: Subclass

—: No value is set.

¹ 000 if *nnn* is not set.

² R0000 if *mm* or *nnn* is not set.

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`. Table 3-27 shows the relationships among the `SQLSTATE`, `SQLCODE`, and `SQLWARN0` values if normal termination with warning occurs.

Table 3-27: Relationship among SQLSTATE, SQLCODE, and SQLWARN0 values when normal termination with a warning occurs

SQLSTATE value	SQLCODE value	SQLWARN0 value
01nnn (<i>nnn</i> ≠ R00)	0	'w'
01R00	Positive number other than 100	Blank 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='40nnn'`, 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 *10.1.1 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='01nnn'`, 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 3-27*.

(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.

Table 3-28 shows the additional return code information and the items that are referred to by the information.

Table 3-28: Additional return code information and items referred to by the information

Additional information	Referenced item
Message concerning <code>SQLCODE</code> *	<code>SQLERRML</code> field in SQL Communications Areas and contents of <code>SQLERRMC</code> field
When a distributed system is used, information that identifies the source of the error in terms of the local RD node vs. other nodes	Contents of <code>SQLCAIDE</code> field of SQL Communications Areas

* 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 8 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 with the FIX attribute
- 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 Using tables for managing numbers
- 4.12 Narrowed search
- 4.13 File output facility for BLOB data
- 4.14 Addition update and partial extraction facility for BLOB and BINARY data
- 4.15 Retrieve first n records facility
- 4.16 Automatic reconnect facility
- 4.17 Locator facility
- 4.18 Facility for returning the total number of hits

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 8 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.

4.2 Manipulation of 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 (e.g., 0 for numeric data and blanks for character data).

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 8 System Operation Guide*.

Stored procedures and stored functions can code processing procedures in SQL or Java. Those coded in SQL are called *SQL stored procedures* and *SQL stored functions*, while those coded in Java are called *Java stored procedures* and *Java stored functions*.

For details about Java stored procedures and Java stored functions, see 9. *Java Stored Procedures and Java 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 program 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

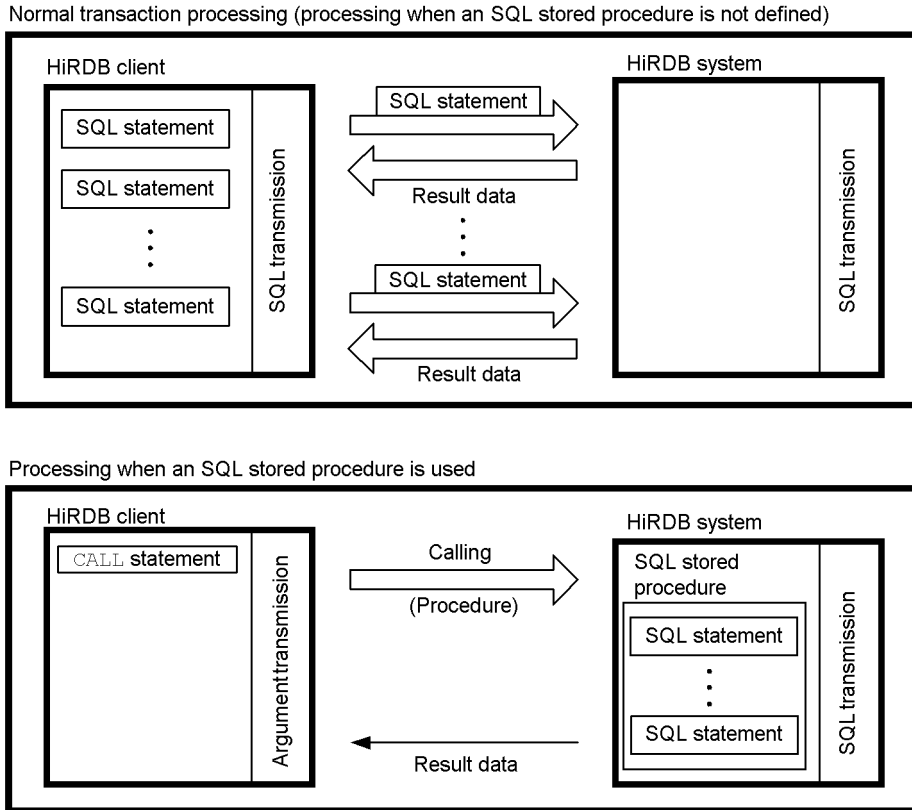
A stored procedure is a facility that registers an SQL-coded database processing procedure to a database as a procedure.

(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.

Figure 4-1 shows the benefits of using an SQL stored procedure.

Figure 4-1: Benefits of using an SQL stored procedure



(2) Defining and executing an SQL stored procedure

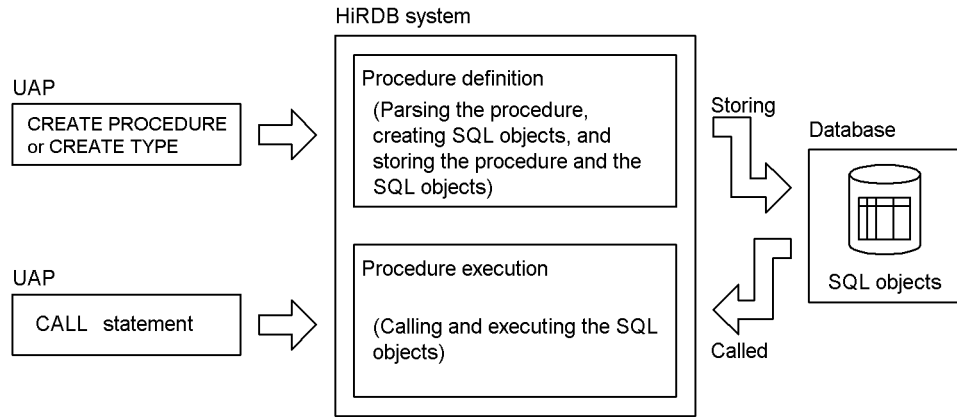
CREATE PROCEDURE or CREATE TYPE stores a defined procedure in a database as an SQL stored procedure; DROP PROCEDURE deletes an SQL stored procedure from the database. Once stored in a database, an SQL stored procedure can be executed by calling it with a CALL statement.

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 pdddefrev 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 pdddefrev command, see the *HiRDB Version 8 Command Reference* manual.

Definition and execution of an SQL stored procedure are illustrated in Figure 4-2.

Figure 4-2: Defining and executing an SQL stored procedure



(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 4-3: 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 ----- 5.
      LEAVE while_loop;
    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 ----- 7.
      SET x_total_quantity = x_total_quantity + x_quantity_1;
      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 8 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 8 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

4. UAP Design for Improving Performance and Handling

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 version of HiRDB client)

```
setenv PDWRTLNFILSZ 4096
```

PDWRTLNFILSZ setup example (Windows version of 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:

- 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) Note on transaction completion

When creating a stored procedure that completes a transaction, you must consider the note described below.

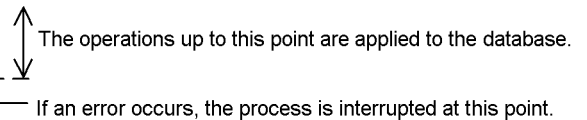
(c) Re-executing a stored procedure

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.

```

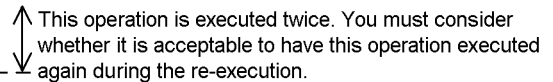
Procedure PROC1
INSERT INTO T1 VALUES1(100)
COMMIT -----
UPDATE T2 SET .....
    
```



The cause of the error is removed and procedure PROC1 is re-executed.

```

Procedure PROC1
INSERT INTO T1 VALUES1(100)
COMMIT -----
UPDATE T2 SET .....
    
```



(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.

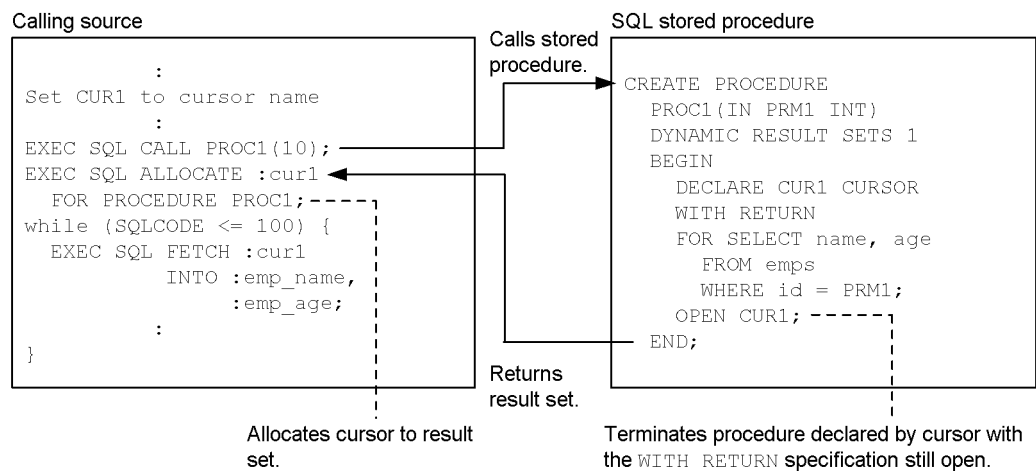
(a) What is the results-set return facility?

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.

Figure 4-4 shows an overview of the results-set return facility.

Figure 4-4: Overview of results-set return facility (for SQL stored procedures)



(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
  OPN 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 {
    long len;
    char str[31];
  } curl;
  int emp_id;
  char emp_name[13];
  int emp_age;
  EXEC SQL END DECLARE SECTION;

```

4. UAP Design for Improving Performance and Handling

```
----- (CONNECT process to HiRDB (omitted)) -----  
  
curl.len = sprintf(curl.str, "cursor1"); ..... 1  
EXEC SQL CALL PROC(10); ..... 2  
If (SQLCODE == 120) { ..... 3  
    EXEC SQL ALLOCATE GLOBAL :curl  
        FOR PROCEDURE PROC2; ..... 4  
  
    printf("*** emps_1 ***\n"); ..... 5  
    while (1) { ..... 5  
        EXEC SQL FETCH GLOBAL :curl ..... 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 :curl; ..... 6  
  
    if (SQLCODE == 121) { ..... 7  
        printf("*** emps_2 ***\n"); ..... 8  
        while (1) { ..... 8  
            EXEC SQL FETCH GLOBAL :curl ..... 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 :curl; ..... 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`.

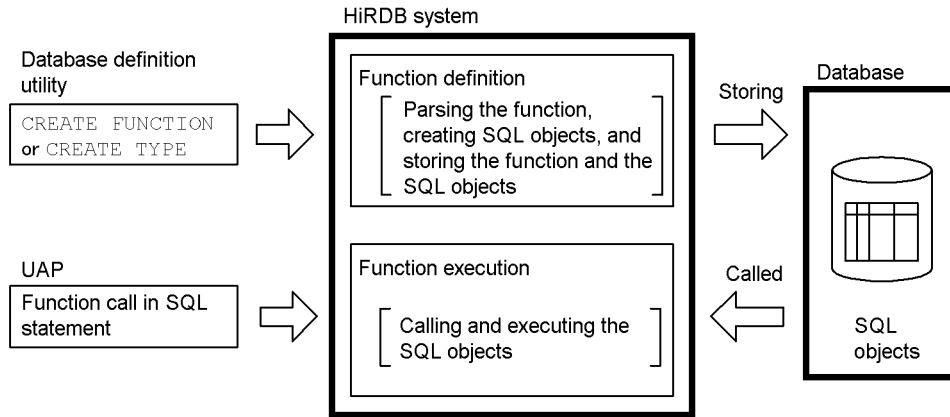
4.3.2 Defining a stored function

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. Figure 4-5 shows the definition and execution of an SQL stored function.

Figure 4-5: Defining and executing an SQL stored function



(2) Example of an SQL stored function

Figure 4-6 shows an example of how routine control SQL statements are combined and defined as a user-defined function and how the function is called and executed.

Figure 4-6: 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 8 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;
```

4. UAP Design for Improving Performance and Handling

```
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;
```

4. UAP Design for Improving Performance and Handling

```
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. Table 4-1 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 4-1: Priorities of pre-defined data types

Argument data type	Priority
Numeric data	SMALLINT → INTEGER → DECIMAL → SMALLFLT → FLOAT
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 4-1. If a data type is an abstract data type, the priority is determined according to Table 4-2 as follows:

Table 4-2: 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 8 SQL Reference* manual.

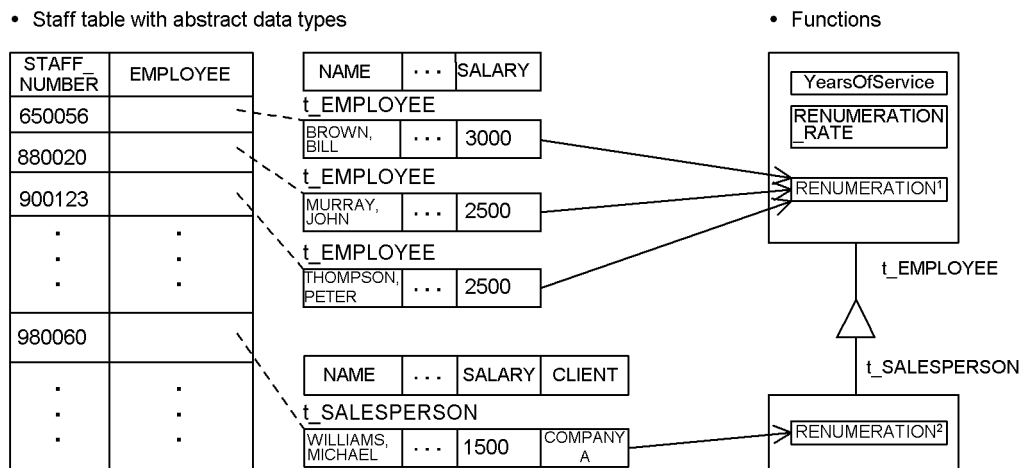
However, if the data type of the function result is an abstract data type, the abstract data type of the RETURNS clause for the basic function is used.

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. Figure 4-7 shows the correspondences between a table with abstract data types and the called function.

Figure 4-7: 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 RENUMERATION to retrieve data from the staff table:

```
SELECT STAFF_NUMBER FROM STAFF_TABLE WHERE
    RENUMERATION (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 8 Installation and Design Guide*.

¹ REMUNERATION = SALARY × REMUNERATION_RATE ()

² REMUNERATION = TOTAL_NUMBER_OF_CLIENTS × 1000 + SALARY × 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 invalid 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:

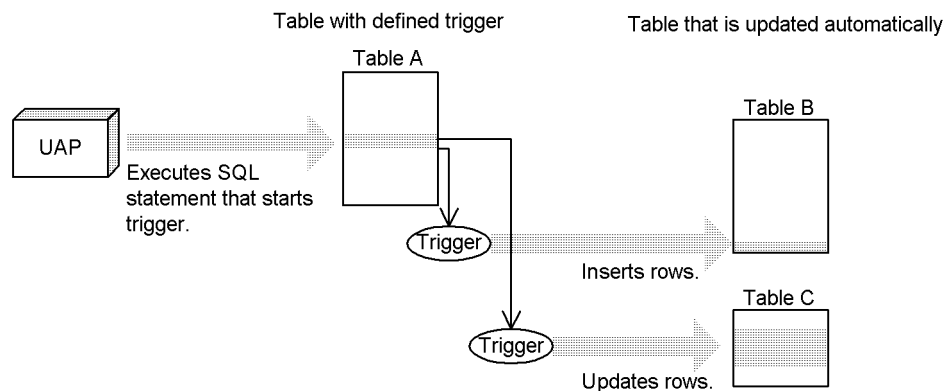
- A UAP has called a stored procedure that has the same name (same authorization identifier and same routine identifier) and the same number of parameters as the stored function to be created.

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. Figure 4-8 shows an overview of triggers.

Figure 4-8: Trigger overview



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 8 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 8 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 8 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

Table 4-3 describes the features of the SQL optimizing modes.

Table 4-3: 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.)	Do not specify the SQL extension optimizing option, or clear the specification for application of optimizing mode 2 based on cost 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 application of optimizing mode 2 based on cost in 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
- Addition update and partial extraction facility for BLOB data and BINARY data
- HiRDB External Data Access facility
- 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

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) Addition update and partial extraction facility for 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) HiRDB External Data Access facility

- When a foreign table is included in a query

Example

```
SELECT T1.C1,FT2.C2 FROM T1 LEFT OUTER JOIN FT2  
ON T1.C1=FT2.C1
```

Note

The underlined section is the applicable location. FT2 is a foreign table.

(h) 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.

(i) 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.

(j) 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.

(k) 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.

(l) 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.

(m) 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.

(n) 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.

(o) 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.

(p) 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.

(q) 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.

(r) 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.

(s) 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.

(t) 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.

(u) 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.

(v) 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.

(w) 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.

(x) 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.

(y) 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.

(z) 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.

(aa) 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.

(ab) 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.

(3) Valid scope of the SQL optimization option and SQL extension optimizing option

Table 4-4 shows the SQL optimizing modes in which the SQL optimization option and SQL extension optimizing option are valid.

Table 4-4: 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 8 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, the system generator, `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 8 Command Reference* and verify the performance.
 - With data stored in the table, set the optimizing information collection level to `lv11` (specify `lv11` in the `-c` option) and execute the optimizing information collection utility. When `lv11` 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 `lv11`.

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

(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. Suppressing foreign server execution of SQL statements that contain join operations
4. Forcing foreign server execution of SQL statements that contain direct products

5. Suppressing derivation of unconditionally created high-speed search conditions for foreign server execution

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:

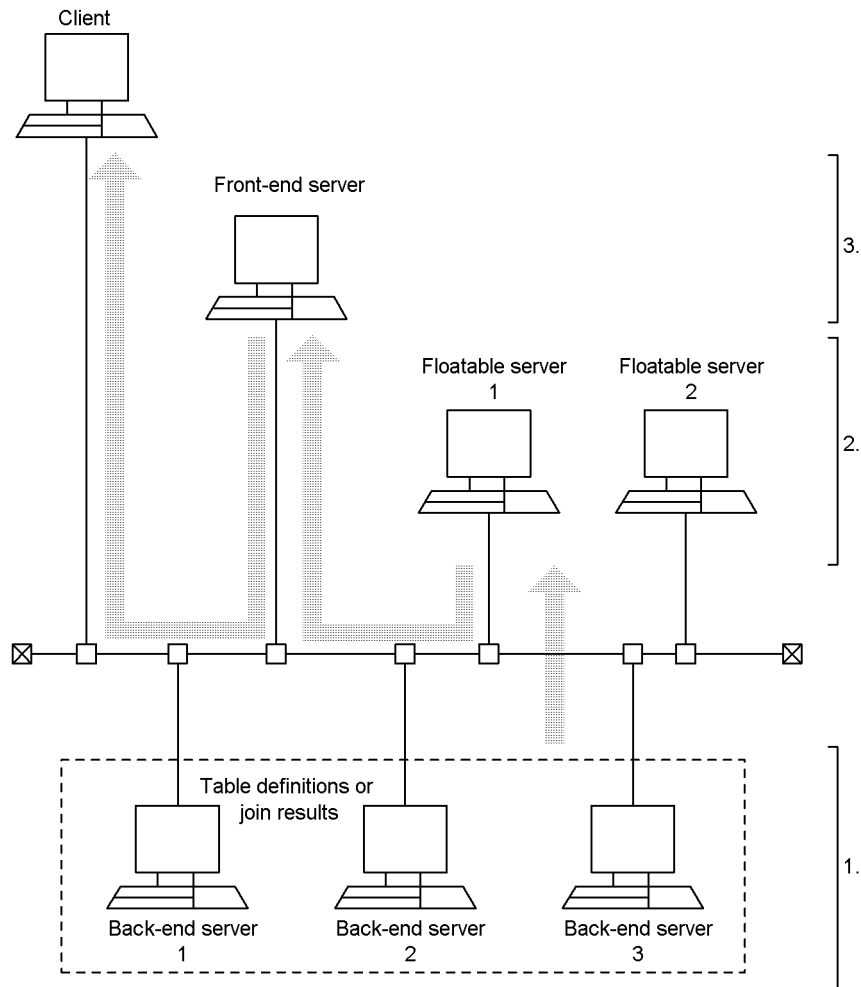
1. SQL command options (procedure body of ALTER PROCEDURE, ALTER ROUTINE, ALTER TRIGGER, CREATE PROCEDURE, CREATE TRIGGER and CREATE TYPE)
2. pd_optimize_level and pd_additional_optimize_level operands of the front-end server definitions
3. pd_optimize_level and pd_additional_optimize_level operands 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. Figure 4-9 shows the query processing method of SQL statements in the HiRDB/Parallel Server.

Figure 4-9: 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

Table 4-5 shows the optimization features related to floatable server allocation.

Table 4-5: 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.
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.

Optimization method	Advantages	Disadvantages
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

Table 4-6 shows the optimization features related to the number of floatable server allocation candidates.

Table 4-6: Optimization features related to number of floatable server allocation candidates

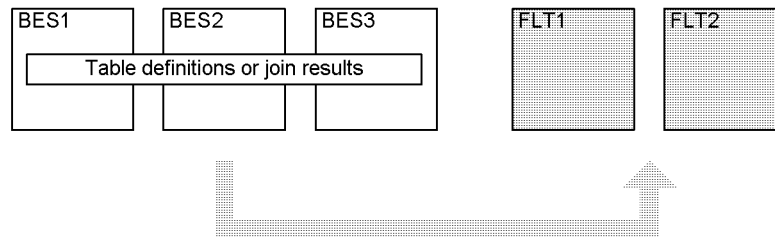
Optimization method	Advantages	Disadvantages
Omitted	In searches involving many data items, more servers are allocated as floatable servers. In searches involving few data items, fewer servers are allocated as floatable servers.	If a narrowing predicate such as = or BETWEEN is specified in the search 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

Figure 4-10 shows floatable server allocation when the optimization method is omitted.

Figure 4-10: Floatable server allocation when the optimization method is omitted



BES: Back-end server
 FLT : Floatable server
 [shaded square] : 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.

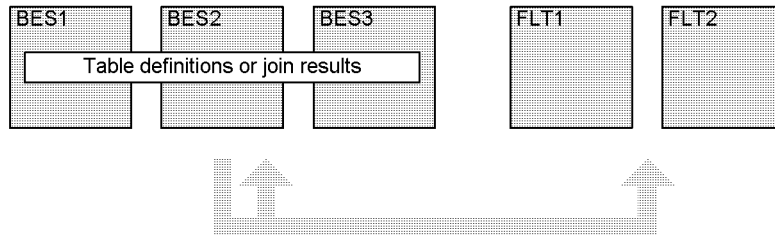
If increasing the number of floatable server candidates is specified, the HiRDB system allocates both FLT1 and FLT2 as floatable servers.


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

Figure 4-11 shows floatable server allocation when increasing the target floatable servers (back-end servers for fetching data).

Figure 4-11: 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.

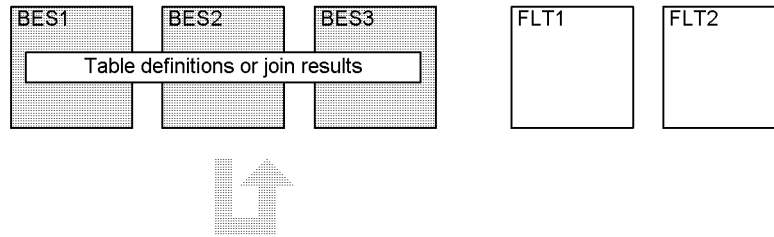
If increasing the number of floatable server candidates is specified, the HiRDB system allocates BES1, BES2, BES3, FLT1, and FLT2 as floatable servers.


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

Figure 4-12 shows floatable server allocation when limiting the target floatable servers (back-end servers for fetching data) is applied.

Figure 4-12: Floatable server allocation when limiting 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, and BES3. However, all of these servers are not necessarily allocated.

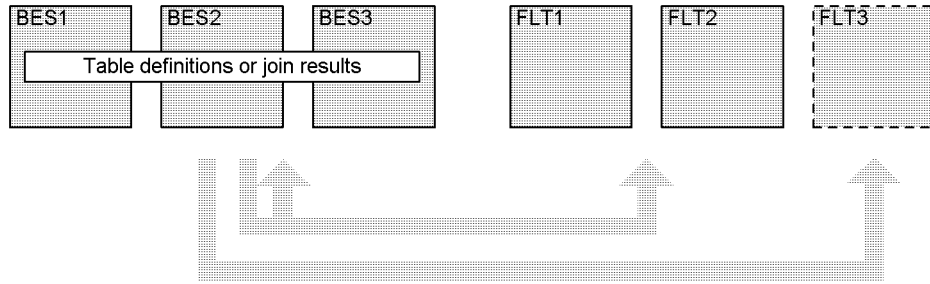
If increasing the number of floatable server candidates is specified, the HiRDB system allocates BES1, BES2, and BES3 as floatable servers.



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

Figure 4-13 shows floatable server allocation when separating data collecting servers is used.

Figure 4-13: Floatable server allocation when separating data collecting servers is applied



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 than 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 8 Command Reference* manual.

Table 4-7 describes the optimization features related to grouping processing methods.

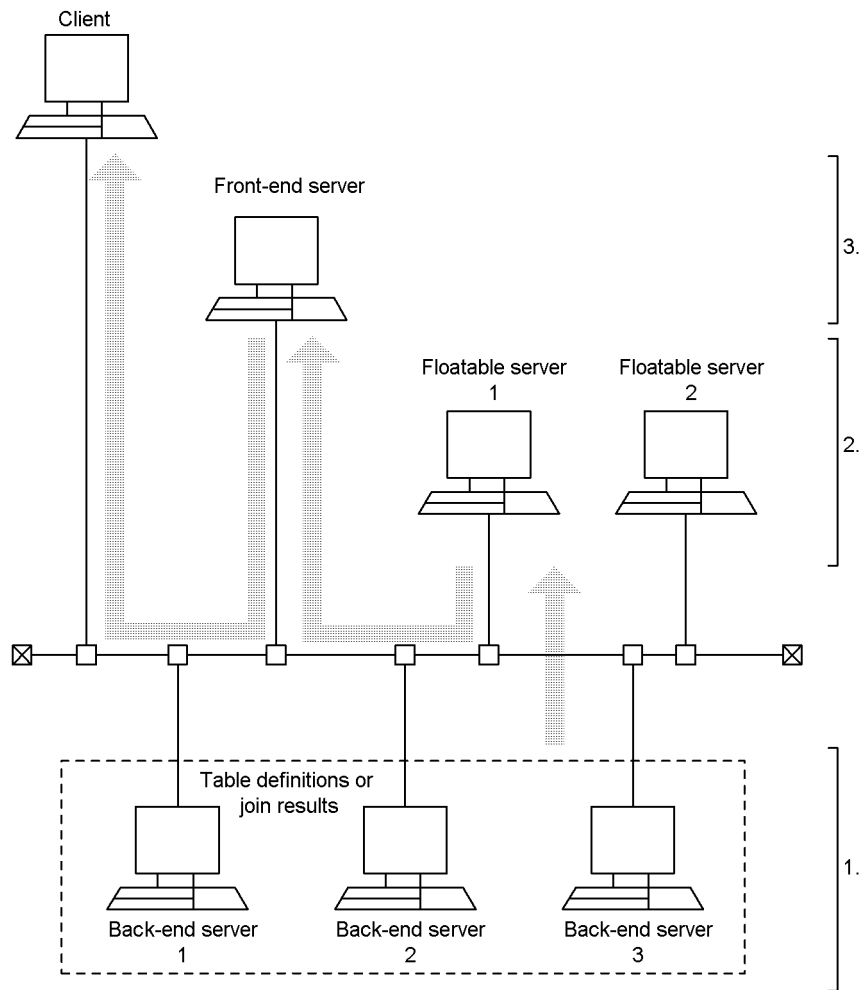
Table 4-7: Optimization features related to grouping processing methods

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

Figure 4-14 shows grouping processing when the optimization method is omitted.

Figure 4-14: Grouping processing method when the optimization method is omitted



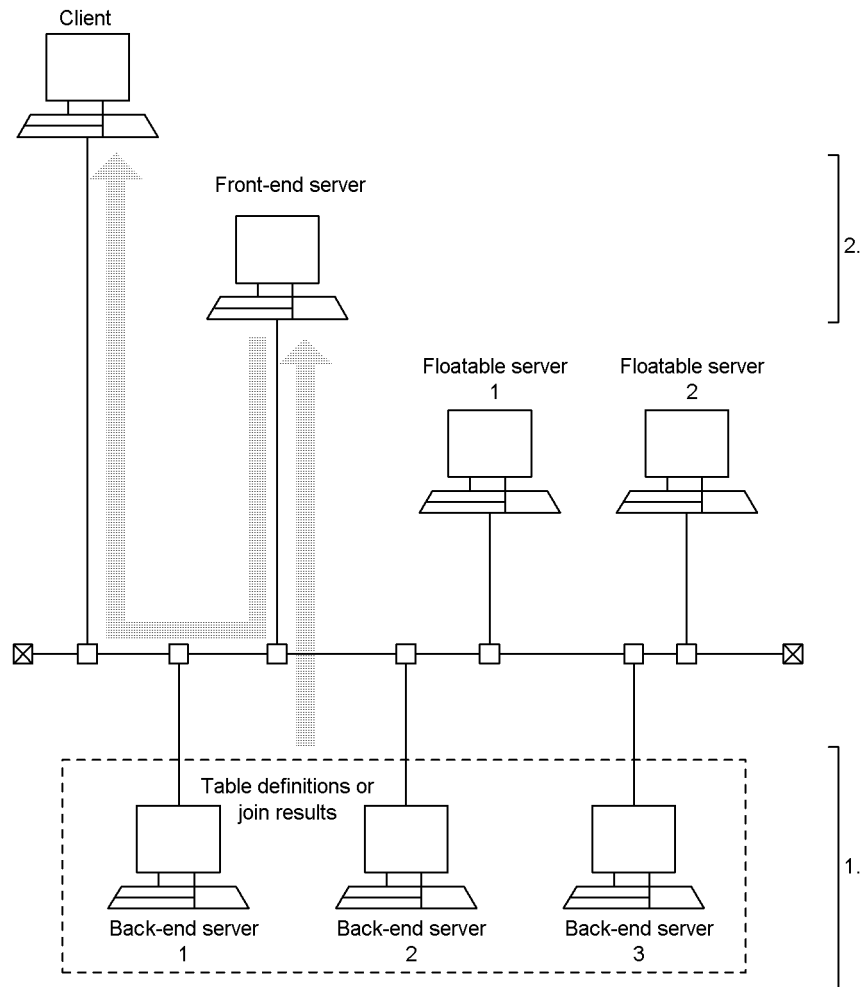
Explanation

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

Figure 4-15 shows grouping processing when rapid grouping processing is applied.

Figure 4-15: Grouping processing when rapid grouping processing is applied

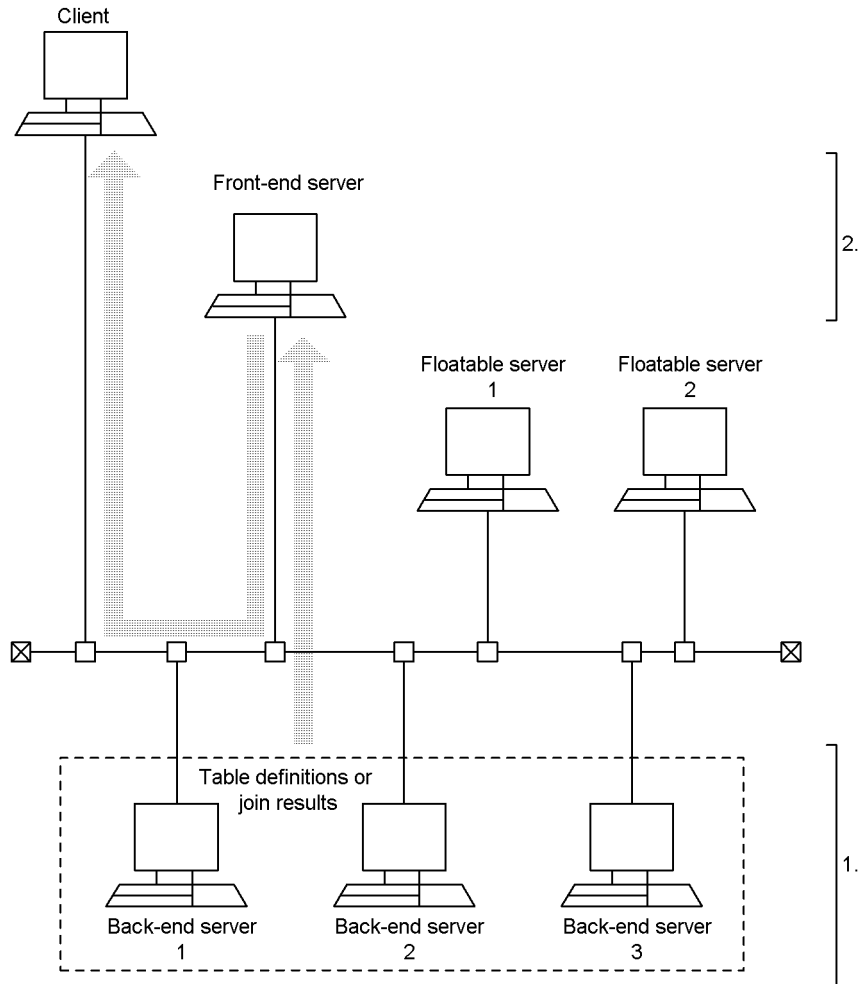
**Explanation**

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

Figure 4-16 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 4-16: 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

Table 4-8 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 4-8: 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.
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.

Join method	Processing method	Initial data fetching	Advantages	Disadvantages
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 (<code>pdgetcst</code>) must be executed.* Also, the SQL object size becomes large because several join methods are prepared.
Distributed nested loops join	This method uses join column values of the outer table to match up the values in a foreign table, which is the inner table. (This method is applied when HiRDB External Data Access is installed in the system.)	Fast	Data can be retrieved quickly if the number of outer table hits is small and the number of inner table hits is large.	Because the foreign server containing the foreign table is searched each time a row is fetched from the outer table, the performance worsens as the number of outer table hits increases.

* 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 8 Command Reference* and verify the performance.

(2) Processing methods

(a) Merge join

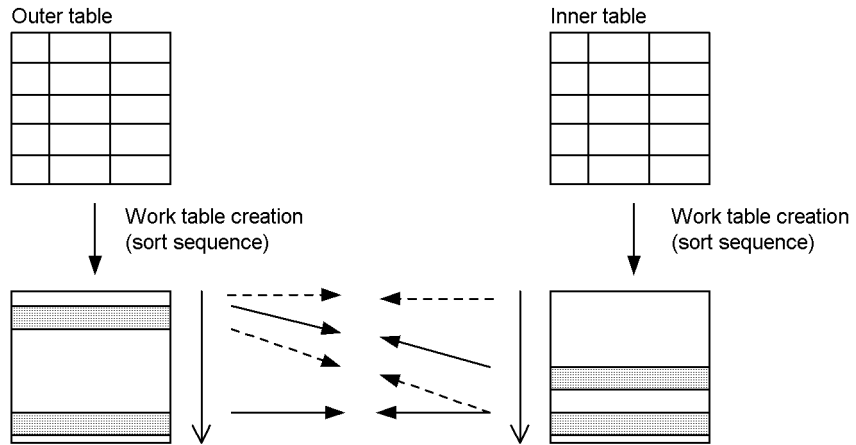
Merge join is effective when the outer table cannot be narrowed very much.

`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.

Figure 4-17 shows the processing of `SORT MERGE JOIN`.

Figure 4-17: Processing of SORT MERGE JOIN

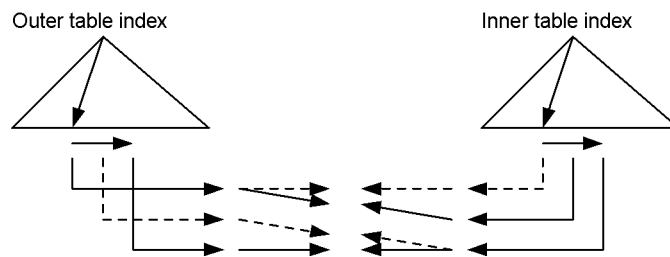


KEY SCAN MERGE JOIN

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.

Figure 4-18 shows the processing of KEY SCAN MERGE JOIN.

Figure 4-18: Processing of KEY SCAN MERGE JOIN

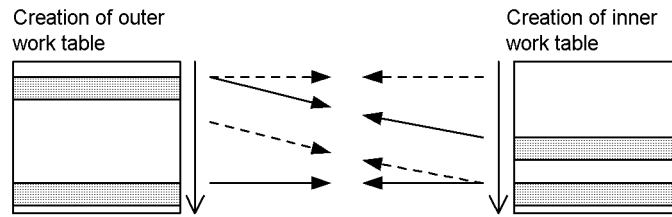


LIST SCAN MERGE JOIN

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.

Figure 4-19 shows the processing of LIST SCAN MERGE JOIN.

Figure 4-19: 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

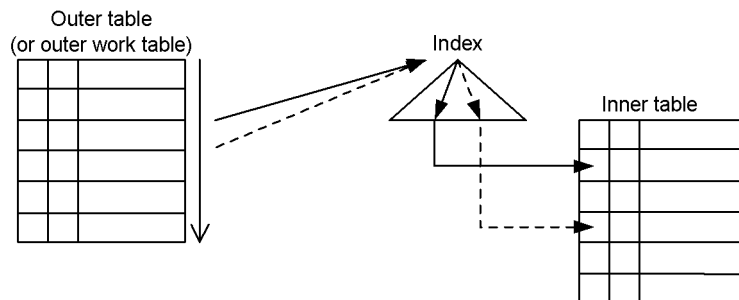
Nested-loops-join is effective if an index is defined in the inner table, and the outer table can be narrowed significantly

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.

Figure 4-20 shows the processing of NESTED LOOPS JOIN.

Figure 4-20: Processing of NESTED LOOPS JOIN

**Note**

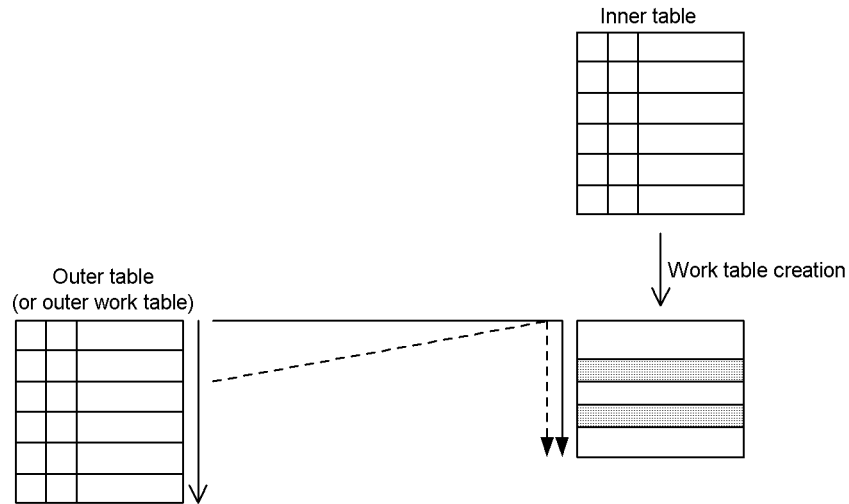
In some cases, an index is used when the outer table is searched.

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.

Figure 4-21 shows the processing of R-LIST NESTED LOOPS JOIN.

Figure 4-21: Processing of R-LIST NESTED LOOPS JOIN



Note

In some cases, an index is used when the outer table is searched.

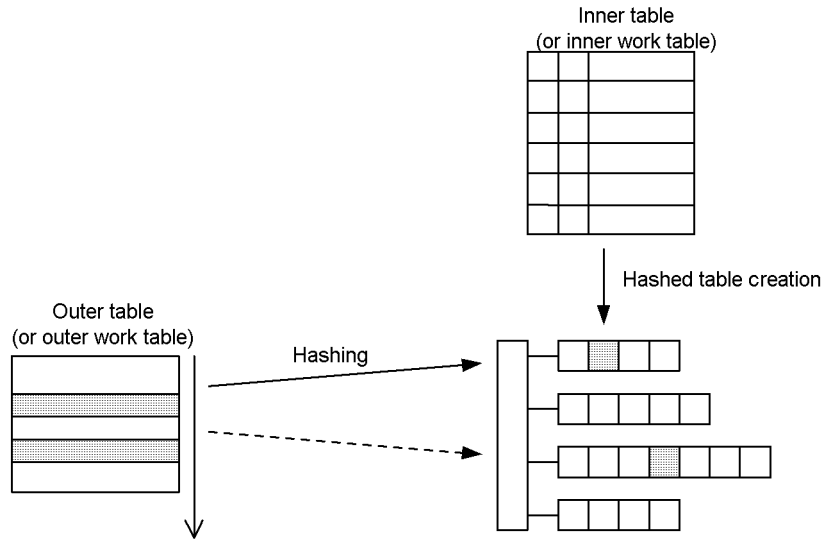
(c) Hash join

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.

Figure 4-22 shows the processing of **HASH JOIN**.

Figure 4-22: Processing of HASH JOIN



There are four hash join processing methods. Table 4-9 describes the hash join processing methods and features.

Table 4-9: 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.*

4. UAP Design for Improving Performance and Handling

Processing method	Description	Advantages	Disadvantages	Selection method
Bucket partitioning hash join	<p>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.</p> <p>The amount of memory used becomes small, and the processing performance drops slightly.</p>	<p>Hash join can be executed in environments that have a small work table buffer area.</p>	<p>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.</p>	<p>Change the hash table size.*</p>
Continuous hash join	<p>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.</p> <p>The amount of memory used becomes large, and the processing performance improves.</p>	<p>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.</p>	<p>When the number of tables to be executed becomes large, the work table buffer area that is used becomes large.</p>	<p>This method cannot be selected. The HiRDB system automatically selects the optimal method based on the number of table rows.</p>

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.

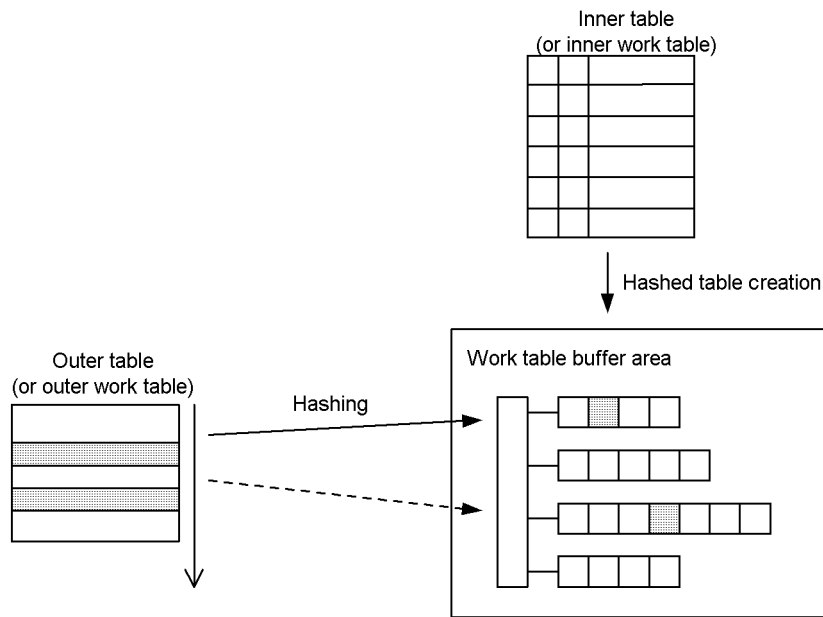
* For details about how to change the hash table size, see 4.5.10 *Preparing for application of hash join and subquery hash execution.*

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. Figure 4-23 shows the processing method of batch hash join.

Figure 4-23: Processing method of batch hash join



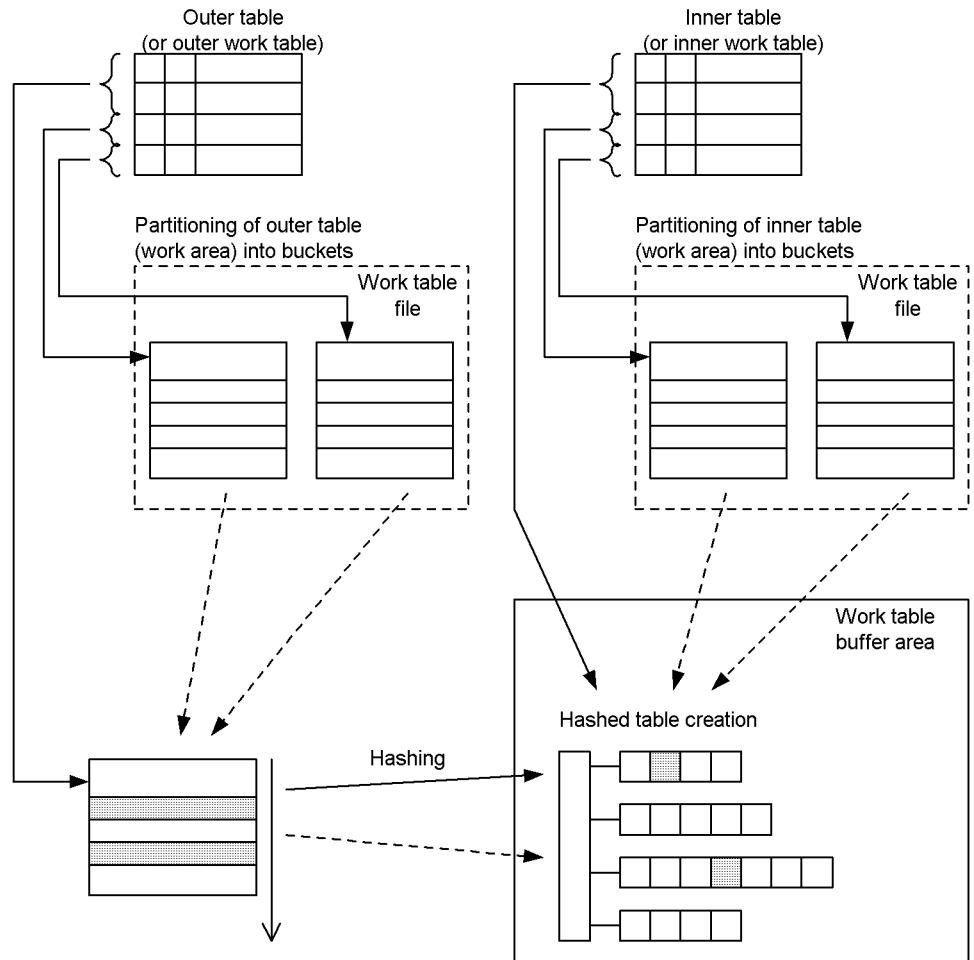
■ **Bucket partitioning hash join**

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.

Figure 4-24 shows the processing method of bucket partitioning hash join.

Figure 4-24: Processing method of bucket partitioning hash join



■ Continuous hash join

This processing method is applied to searches involving three or more tables.

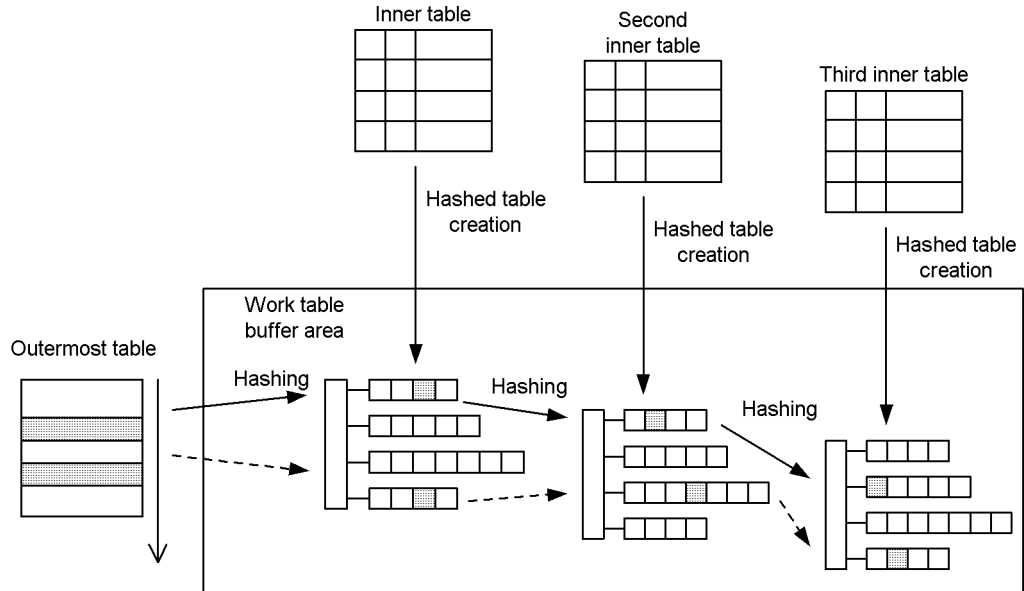
First, hash tables are created from all target tables except the outermost table and expanded into the work table buffer area. Next, a row is fetched from the outer table, hashed, and then matched and joined with the hash table that was created from the inner table. If the join condition is satisfied, 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.

Figure 4-25 shows the processing method of a continuous hash join.

Figure 4-25: 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.

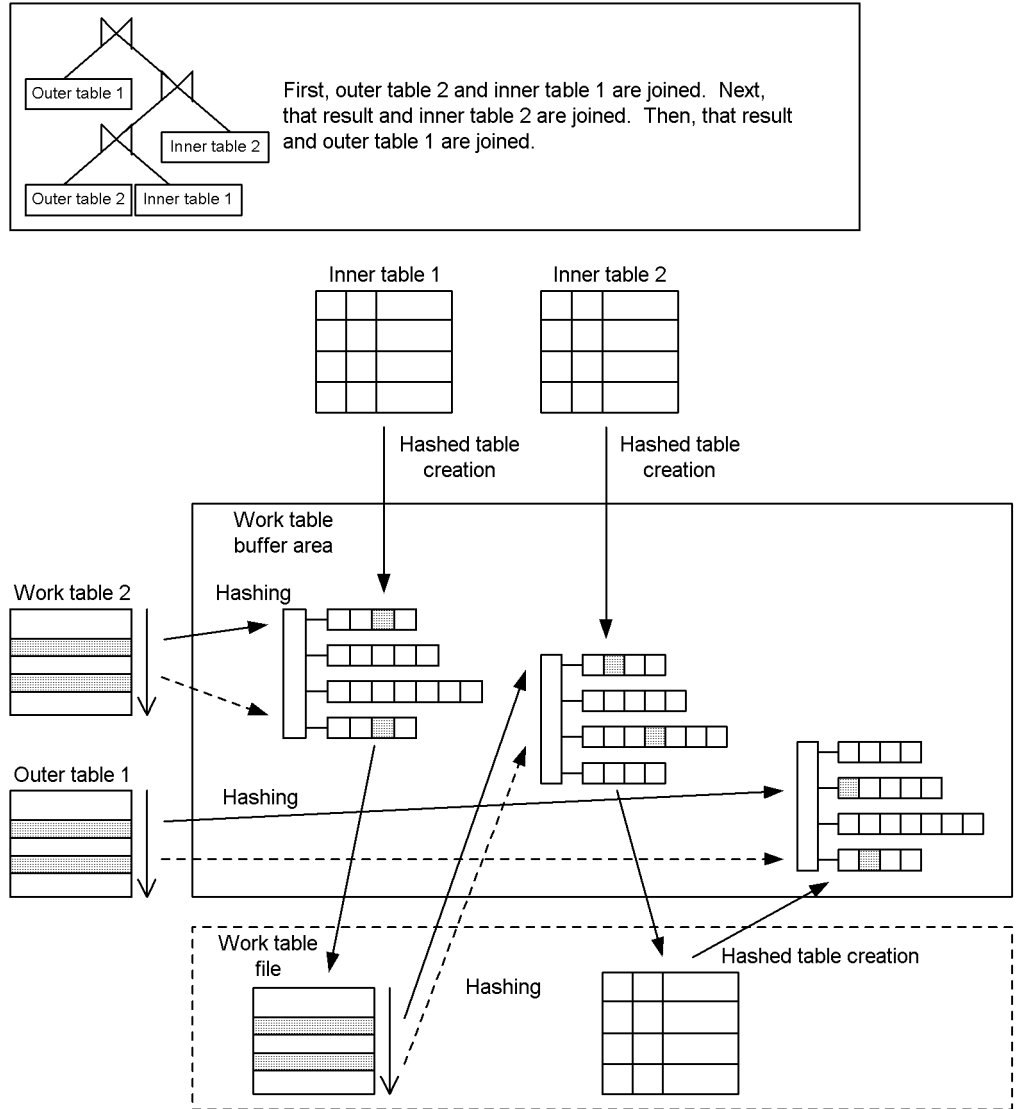
The processing changes depending on whether the join result becomes the outer table or the inner table.

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.

Figure 4-26 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 4-26: Processing method of intermittent hash join



(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.

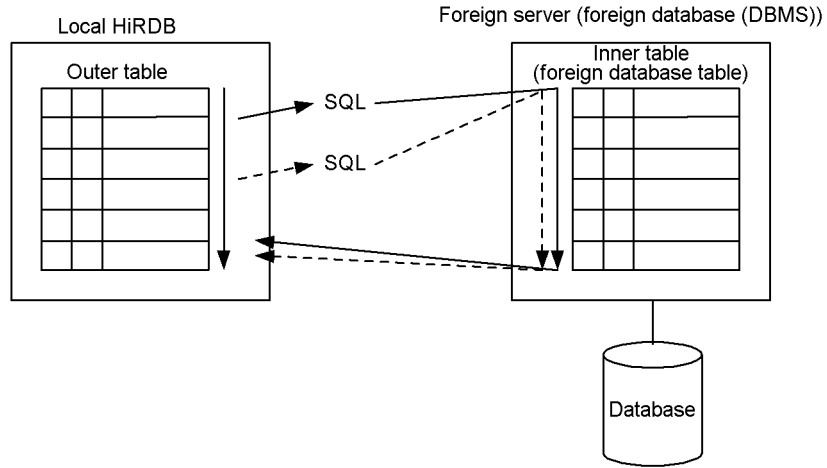
(e) Distributed nested loops join

DISTRIBUTED NESTED LOOPS JOIN (DNL JOIN)

The DISTRIBUTED NESTED LOOPS JOIN method performs a nested-type loop process that fetches rows that satisfy the join condition. To do this, the local HiRDB fetches rows from the outer table. Then for each row, the local HiRDB executes an SQL statement that uses a variable to pass the values of the outer table row to the foreign server where the inner table (foreign table) is located. The values of the outer table row are then matched with those in the inner table. The local HiRDB can get the retrieval search results from the foreign database (DBMS) by sending a foreign table acquisition request to the foreign table.

Figure 4-27 shows the processing method of DISTRIBUTED NESTED LOOPS JOIN.

Figure 4-27: Processing method of DISTRIBUTED NESTED LOOPS JOIN



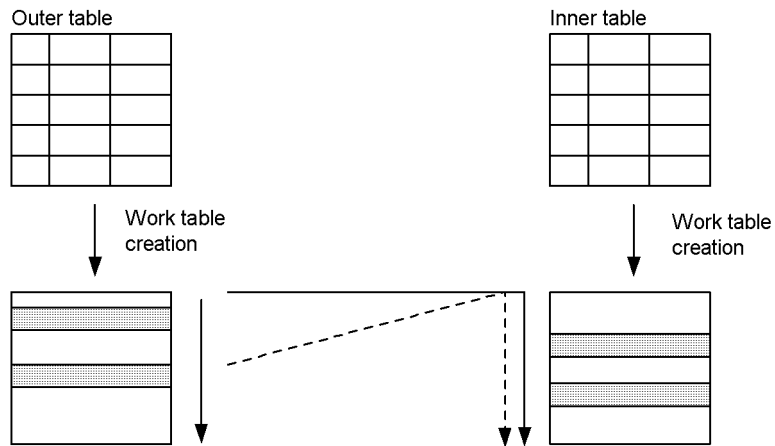
(f) Cross join

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.

Figure 4-28 shows the CROSS JOIN processing method.

Figure 4-28: 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

Table 4-10 describes the search method types (except LIST SCAN, ROWID FETCH, FOREIGN SERVER SCAN and FOREIGN SERVER LIMIT SCAN) 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. FOREIGN SERVER SCAN and FOREIGN SERVER LIMIT SCAN are applied when the local HiRDB gets retrieval results from a foreign server.

Table 4-10: 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.
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 rows can be obtained in order (or reverse order) of the index configuration column values. ¹ The data can be searched rapidly even if a cluster key index is used and the search cannot be narrowed very much.	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.

Search method	Processing method	Advantages	Disadvantages
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 rows can be obtained in order (or reverse order) of the index configuration column values. 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.
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 In processing that requires sorting, the rows can be obtained in order (or reverse order) of the index configuration column values. However, if HiRDB judges that sorting is unnecessary, it may cancel sort processing.

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 8 Command Reference* and verify the performance.

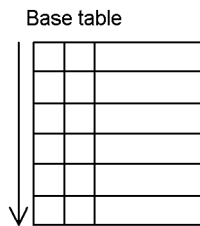
(2) Processing methods

(a) Search using no index

TABLE SCAN

This processing method searches the database pages of a table without using an index. Figure 4-29 shows the TABLE SCAN processing method.

Figure 4-29: TABLE SCAN processing method

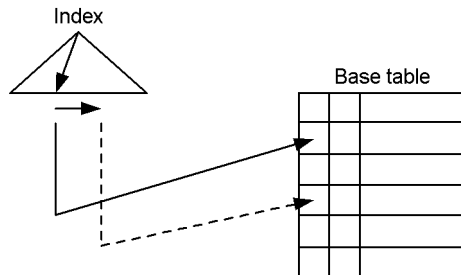


(b) Search using one index

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. Figure 4-30 shows the INDEX SCAN processing method.

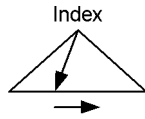
Figure 4-30: INDEX SCAN processing method



KEY SCAN

This processing method searches only the index pages of a single-column index. The data pages are not searched. Figure 4-31 shows the KEY SCAN processing method.

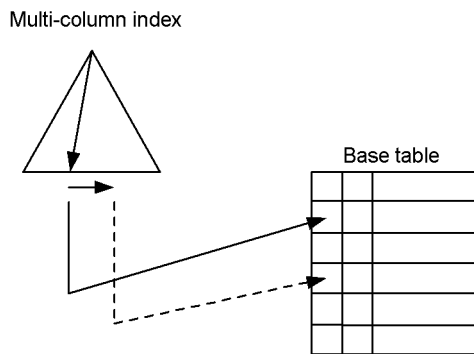
Figure 4-31: KEY SCAN processing method



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. Figure 4-32 shows the MULTI COLUMNS INDEX SCAN processing method.

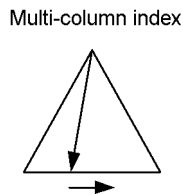
Figure 4-32: MULTI COLUMNS INDEX SCAN processing method



MULTI COLUMNS KEY SCAN

This processing method searches only the index pages of a multi-column index. The data pages are not searched. Figure 4-33 shows the MULTI COLUMNS KEY SCAN processing method.

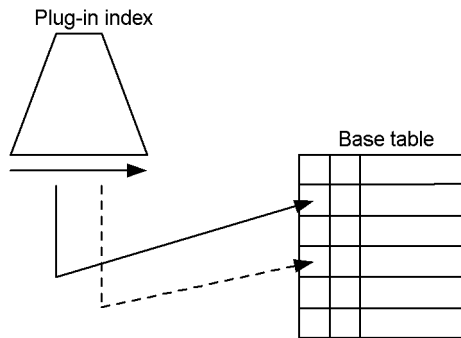
Figure 4-33: MULTI COLUMNS KEY SCAN processing method



PLUGIN INDEX SCAN

This processing method uses a plug-in index to narrow the search and then searches the data pages of the table. Figure 4-34 shows the `PLUGIN INDEX SCAN` processing method.

Figure 4-34: `PLUGIN INDEX SCAN` processing method



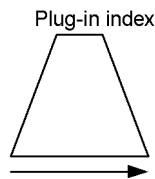
Note

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. Figure 4-35 shows the `PLUGIN KEY SCAN` processing method.

Figure 4-35: `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

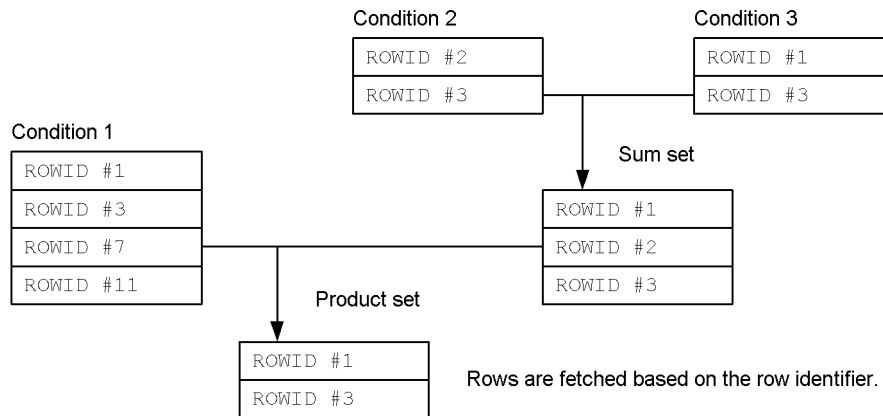
For search conditions that are combined with the AND or OR operator, the respective indexes are used to conduct the search, and the row identifiers (ROWID) are stored in the respective work tables. These work tables are consolidated into a single work table by forming a product set when the AND operator 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.

Figure 4-36 shows the AND PLURAL INDEXES SCAN processing method.

Figure 4-36: AND PLURAL INDEXES SCAN processing method

WHERE *condition-1* AND (*condition-2* OR *condition-3*)



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.

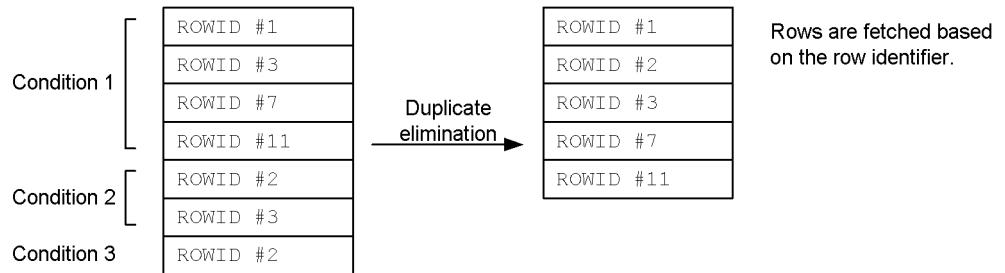
When creating a work table of row identifiers from each condition, HiRDB sometimes uses TABLE SCAN to create the work table, even if the condition

column does not have an index.

Figure 4-37 shows the OR PLURAL INDEXES SCAN processing method.

Figure 4-37: OR PLURAL INDEXES SCAN processing method

WHERE *condition-1* OR *condition-2* OR *condition-3*



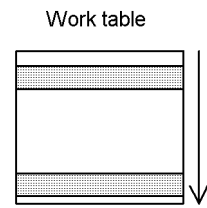
(5) Search of internally created work table

LIST SCAN

This processing method searches a work table that was created internally.

Figure 4-38 shows the LIST SCAN processing method.

Figure 4-38: LIST SCAN processing method



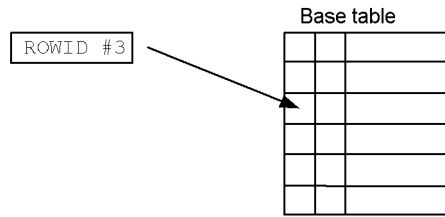
(6) Search using a row identifier

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.

Figure 4-39 shows the ROWID FETCH processing method.

Figure 4-39: ROWID FETCH processing method



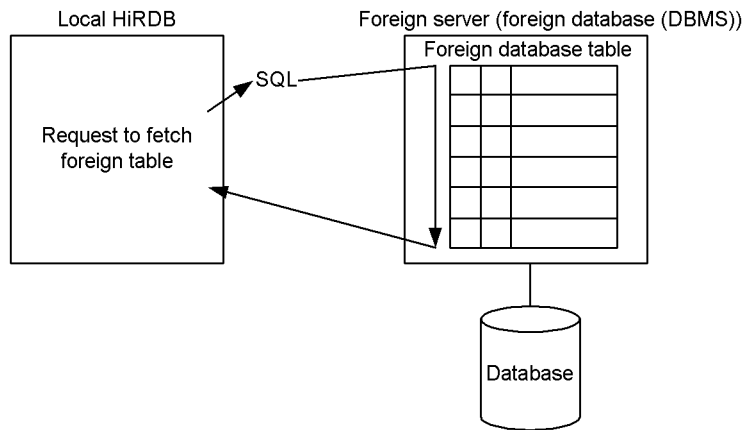
(7) Retrieving query results from a foreign server

FOREIGN SERVER SCAN

The local HiRDB executes an SQL statement to the foreign server where the foreign table is located. The local HiRDB then receives the retrieval results for the query.

Figure 4-40 shows the FOREIGN SERVER SCAN processing method.

Figure 4-40: FOREIGN SERVER SCAN processing method

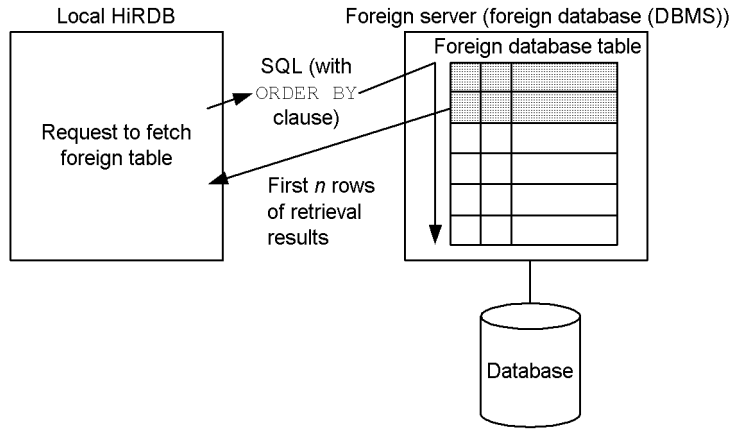


FOREIGN SERVER LIMIT SCAN

When the retrieve first *n* records facility is being used, the local HiRDB executes an SQL statement containing an ORDER BY clause to the foreign server where the foreign table is located. The local HiRDB then receives the first *n* rows of the retrieval results for the query.

Figure 4-41 shows the FOREIGN SERVER LIMIT SCAN processing method.

Figure 4-41: FOREIGN SERVER LIMIT SCAN processing method



4.5.8 Execution of subqueries with no external references

(1) Execution method types

Table 4-11 describes the execution formats and features of inquiries that do not have external references. Table 4-12 describes the optimal execution methods of queries that do not have external references.

Table 4-11: 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.

Execution method	Processing method	Advantages	Disadvantages
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 4-12: 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)			

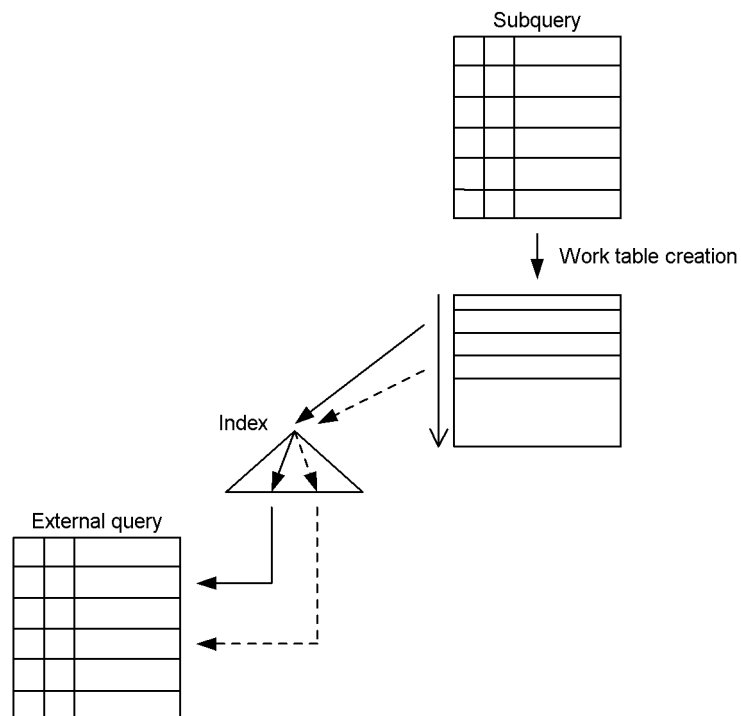
(2) Processing methods**(a) Work table ATS execution****WORK TABLE ATS SUBQ**

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. Figure 4-42 shows the WORK TABLE ATS SUBQ processing method.

Figure 4-42: 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.

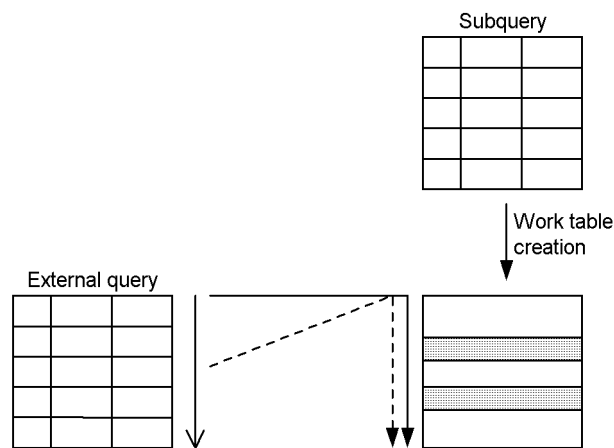
(b) Work table execution

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.

Figure 4-43 shows the WORK TABLE SUBQ processing method

Figure 4-43: 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

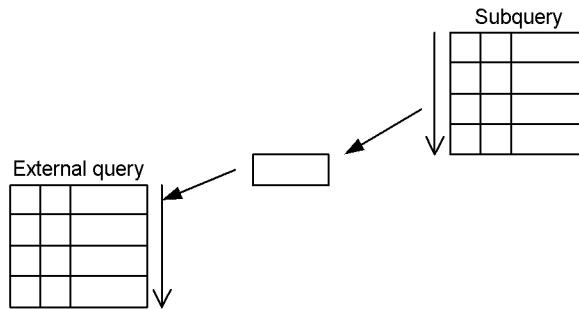
ROW VALUE SUBQ

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.

Figure 4-44 shows the `ROW VALUE SUBQ` processing method.

Figure 4-44: `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

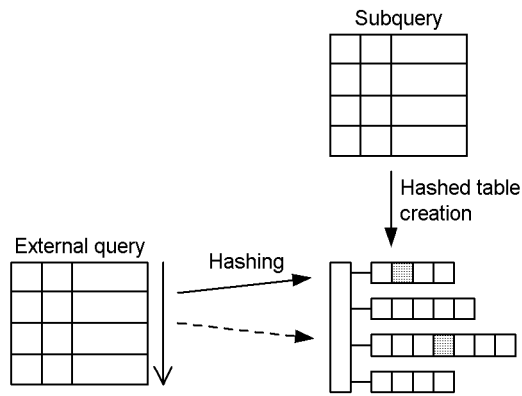
`HASH SUBQ`

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.

Figure 4-45 shows the `HASH SUBQ` processing method.

Figure 4-45: 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

Table 4-13 shows the execution methods and features of subqueries that have external references.

Table 4-13: 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

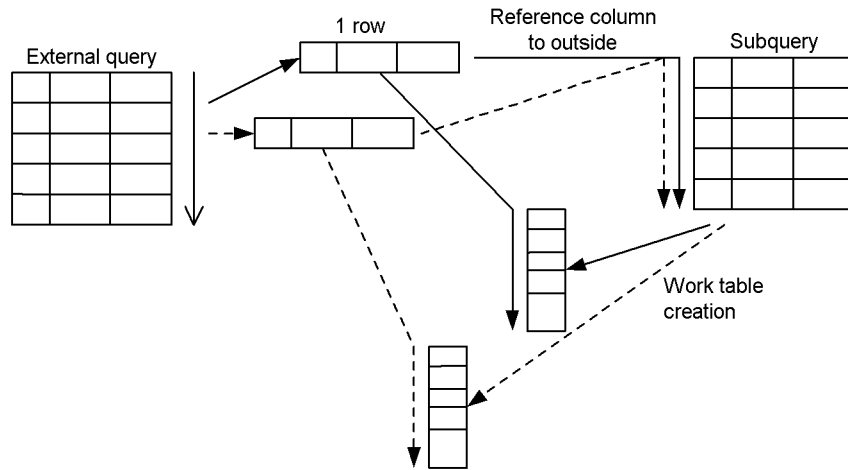
This processing method is applied to table subqueries specified on the right side of quantified predicates and `IN` predicates.

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.

Figure 4-46 shows the NESTED LOOPS WORK TABLE SUBQ processing method.

Figure 4-46: 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

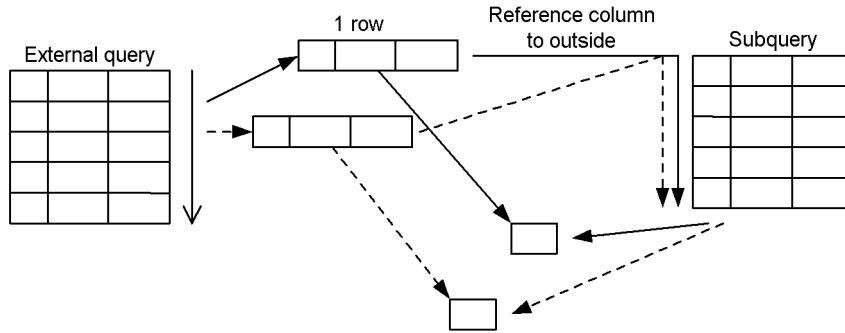
This processing method is applied to row subqueries, scalar subqueries, and EXISTS predicates.

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.

Figure 4-47 shows the NESTED LOOPS ROW VALUE SUBQ processing method.

Figure 4-47: 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

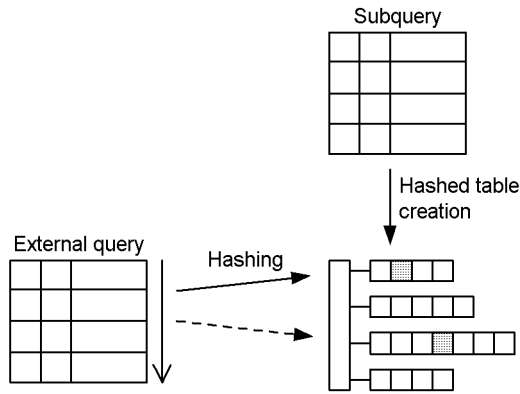
This processing method applies to table subqueries specified in EXISTS predicates and on the right side of comparison predicates, quantified predicates, and IN predicates.

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.)

Figure 4-48 shows the HASH SUBQ processing method.

Figure 4-48: 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:

$$\text{hash-table-size (in kilobytes)} \geq \lceil (\text{maximum-hash-table-row-length (in bytes)} \times 2 + 32) \div 128 \rceil \times 128$$

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:

$$\text{Row length of each table} = \sum_{i=1}^n (a_i) + 2 \times n$$

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:

$$\text{Hashed table row length} = \frac{\sum_{j=1}^{m-1} \text{Row length } j \text{ of each table}}{4} \times 4 + 6$$

(Unit: bytes)

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:

$$\text{Hashed table row length} = \frac{\sum_{i=1}^n (a_i) + 2 \times n}{4} \times 4 + 6$$

(Unit: bytes)

a_i

Data length of the *i*-th data item. For details about data length, see the *HiRDB Version 8 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.

$$\text{hash-table-size-for-batch-hash-join (in kilobytes)} = \uparrow (\text{number-of-hash-table-data-pages} + \text{number-of-hash-table-management-table-pages}) \div \text{number-of-one-segment-pages} \uparrow \times 128$$

$$\text{number-of-hash-table-data-pages} = \uparrow \text{number-of-hash-table-rows} \div \text{MIN} \{ \downarrow (\text{hash-table-page-length} \ 48) \div \text{hash-table-row-length} \downarrow, 255 \} \uparrow + 63$$

$$\text{number-of-hash-table-management-table-pages} = \uparrow (16 \times \text{number-of-hash-table-rows} + (\uparrow (\text{number-of-hash-table-data-pages} \times \text{hash-table-page-length} + 16 \times \text{number-of-hash-table-rows}) \div (\text{number-of-one-segment-pages} \times \text{hash-table-page-length}) \uparrow \times 8) + 8) \div \text{hash-table-page-length} \uparrow \times \text{hash-table-page-length}$$

$$\text{number-of-one-segment-pages} = \downarrow (128 \times 1024) \div \text{hash-table-page-length} \downarrow$$

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
16,361 to 32,720	$\uparrow (\text{hash-table-row-length} + 48) \div 2048 \uparrow \times 2048$ <i>hash-table-row-length:</i> 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:

$$\text{work-table-buffer-size (in kilobytes)} \geq (\text{hash-table-size (in kilobytes)} \times 2 + 256) \times \text{maximum-number-of-hash-joins-in-SELECT-statement} + 128$$

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).

$$\begin{aligned} \text{number-of-hash-joins-in-SELECT-statement} = & \\ & ((\text{number-of-tables-joined-with=} \\ & (\text{number-of-query-specifications-joined-with=}), + \\ & (\text{number-of=} \text{ANY-quantified-predicates}) + \\ & (\text{number-of=} \text{SOME-quantified-predicates}) + \\ & (\text{number-of-IN-(subquery)-specifications}) + \\ & (\text{number-of-other-subqueries-that-specify-external-reference-columns-with=} \text{in} \\ & \text{-search-conditions}) \end{aligned}$$

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:

$$\text{work-table-buffer-size (in kilobytes)} \geq \text{hash-table-size (in kilobytes)} \times \text{maximum-number-of-hash-joins-in-SELECT-statement} + 384$$

(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 *10.1.4 UAP statistical report facility*. For details about the statistics analysis utility, see the *HiRDB Version 8 Command Reference* manual.

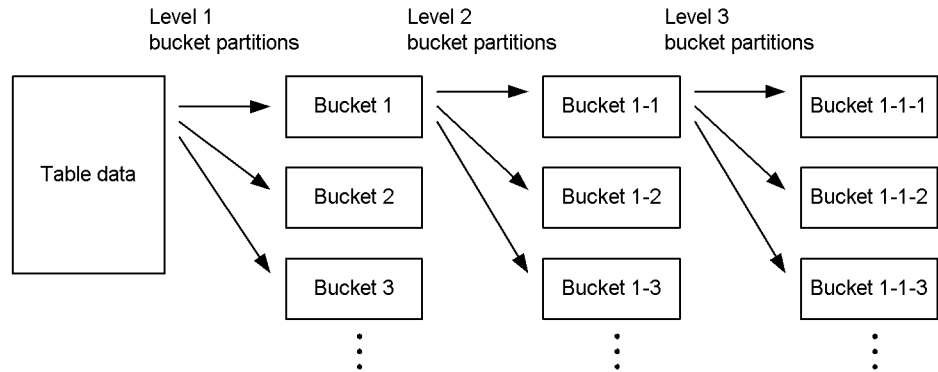
(b) Items derived from tuning information

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:

$$number-of-bucket-partitions = \text{MIN} \{ \lfloor (hash-table-size \div 2) \div hash-table-page-length \rfloor, 64 \}$$

Even with batch hash join, level 1 bucket partitioning is executed for inner tables to be joined.

(c) Tuning methods

Table 4-14 describes the tuning methods for hash table size.

Table 4-14: 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 hash join cannot be used. If this value is 0, hash join or subquery hash execution has not been performed.

Tuning information (unit: kilobytes)	Tuning method
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 to 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 *10.1.4 UAP statistical report facility*. For details about the statistics analysis utility, see the manual *HiRDB Version 8 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

Table 4-15 shows tuning information for the hashing mode.

Table 4-15: 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

(a) UNIX version

Whether or not HiRDB derives high-speed search conditions depends on the specification values of the SQL optimization and SQL extension optimizing options. Table 4-16 shows the relationships between the SQL optimization and SQL extension optimizing options and deriving high-speed search conditions.

Table 4-16: 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 ²	Derive only to foreign servers ³	Derive ⁴
CNF conversion	OR condition for one table	Internal table	One-table condition		—	—	—
		Foreign table ¹	One-table condition		—	—	—
	OR condition extending across two or more tables	Internal table	One-table condition		—	—	G
			Join condition (<i>column=column</i>) is extracted by OR reduction		—	—	G
			Other condition for two or more tables		—	—	—
		Confined to one foreign server ¹	One-table condition		—	—	—
			Join condition (<i>column=column</i>) is extracted by OR reduction		—	—	G
			Other condition for two or more tables		—	—	—
	Extends across multiple foreign servers or across foreign and internal tables ¹	Confined to internal tables	One-table condition	—	—	G	

4. UAP Design for Improving Performance and Handling

Type	Derivation source condition		Derived condition		Specified SQL optimization option or SQL extension optimizing option		
					Do not derive ²	Derive only to foreign servers ³	Derive ⁴
				Join condition (<i>column=column</i>) is extracted by OR reduction	—	—	G
				Other condition for two or more tables	—	—	—
			Confined to one foreign server	One-table condition	—	G	G
				Join condition (<i>column=column</i>) is extracted by OR reduction	—	G	G
				Other condition for two or more tables	—	G	G
			Extends across multiple foreign servers or across foreign and internal tables	Join condition (<i>column=column</i>) is extracted by OR reduction	—	—	G
				Other condition for two or more tables	—	—	—

Type	Derivation source condition		Derived condition	Specified SQL optimization option or SQL extension optimizing option		
				Do not derive ²	Derive only to foreign servers ³	Derive ⁴
Condition shifting	Join condition for tables A and B, and condition for table A	Tables A and B are internal tables	One-table condition for table B	—	—	G
		Tables A and B are confined to one foreign server ¹	One-table condition for foreign table B	—	—	—
		Tables A and B extend across multiple foreign servers or across foreign and internal tables ¹	One-table condition for internal table B	—	—	G
			One-table condition for foreign table B	—	G	G

Type	Derivation source condition		Derived condition	Specified SQL optimization option or SQL extension optimizing option		
				Do not derive ²	Derive only to foreign servers ³	Derive ⁴
	Join condition for tables A and B, and join condition for tables A and C	Tables A, B, and C are internal tables	Join condition for tables B and C	—	—	—
		Tables A, B, and C are confined to one foreign table ¹	Join condition for tables B and C, which are confined to one foreign server	—	—	—
		Tables A, B, and C extend across multiple foreign servers or across foreign and internal tables ¹	Join condition for internal tables B and C	—	—	—
			Join condition for tables B and C, which are confined to one foreign server	—	G	G
			Join condition for tables B and C, which extend across two foreign servers or across foreign and internal tables	—	—	—

Legend:

G: HiRDB generates high-speed search conditions.

—: HiRDB does not generate high-speed search conditions.

Note

An internal table refers to one of the following types of tables:

- Any table when HiRDB External Data Access is not installed
- A non-foreign table when HiRDB External Data Access is installed

¹ This condition applies when HiRDB External Data Access is installed.

² In the SQL extension optimizing options, specify the value for suppressing derivation of unconditionally created high-speed search conditions for foreign server execution.

However, this option specification becomes invalid if the value for deriving high-speed search conditions is specified concurrently in the SQL optimization options.

³ Do not concurrently specify the value for suppressing derivation of unconditionally created high-speed search conditions for foreign server execution in the SQL extension optimizing options and the value for deriving high-speed search conditions in the SQL optimization options.

⁴ Specify the value for deriving high-speed search conditions in the SQL optimization options.

(b) Windows version

High-speed search conditions are generated when you specify the value for deriving high-speed search conditions in the SQL optimization options. Table 4-17 shows the relationships between the SQL optimization options and deriving high-speed search conditions.

Table 4-17: Relationships between the SQL optimization options and deriving high-speed search conditions

Type	Derivation source condition	Derived condition	Specified SQL optimization option (derives high-speed search conditions) [#]
CNF conversion	OR condition for one table	One-table condition	—
	OR condition extending across two or more tables	One-table condition	G
		Join condition (<i>column=column</i>) is extracted by OR reduction	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

Legend:

G: HiRDB generates high-speed search conditions.

— : HiRDB does not generate high-speed search conditions.

#

Specify the value for deriving high-speed search conditions in the SQL optimization options.

(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 HiRDB can generate conditions that are joined with the `OR` operator and are confined to one foreign server when the HiRDB External Data Access facility is being used, HiRDB derives those conditions as a high-speed search conditions. HiRDB can thereby narrow the number of data items to be retrieved from the foreign server.
- 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.
- 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.

(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 version 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 join condition and the one-table condition to be used in deriving the new condition are confined to one foreign server.
- 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.
- Repetition columns are used and joined.
- 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 version 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 join condition contains a foreign table.
- All join conditions from which the new conditions would be derived are confined to one foreign table.
- 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. Table 4-18 shows the relationships between data guarantee level and lock option.

Table 4-18: 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

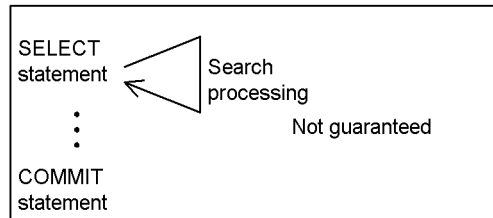
4.6.2 Data guarantee level types

(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.

Figure 4-49 shows the data guarantee range of data guarantee level 0.

Figure 4-49: Data guarantee range of data guarantee level 0

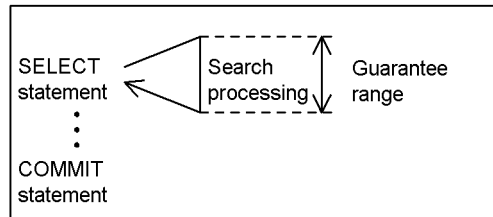


(2) Data guarantee level 1

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.

Figure 4-50 shows the data guarantee range of data guarantee level 1.

Figure 4-50: Data guarantee range of data guarantee level 1

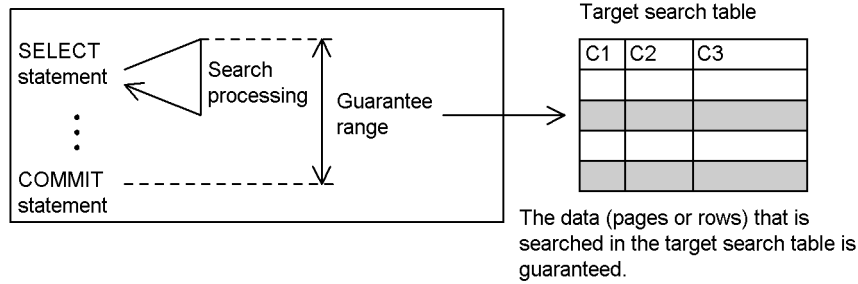


(3) Data guarantee level 2

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.

Figure 4-51 shows the data guarantee range of data guarantee level 2.

Figure 4-51: Data guarantee range of data guarantee level 2



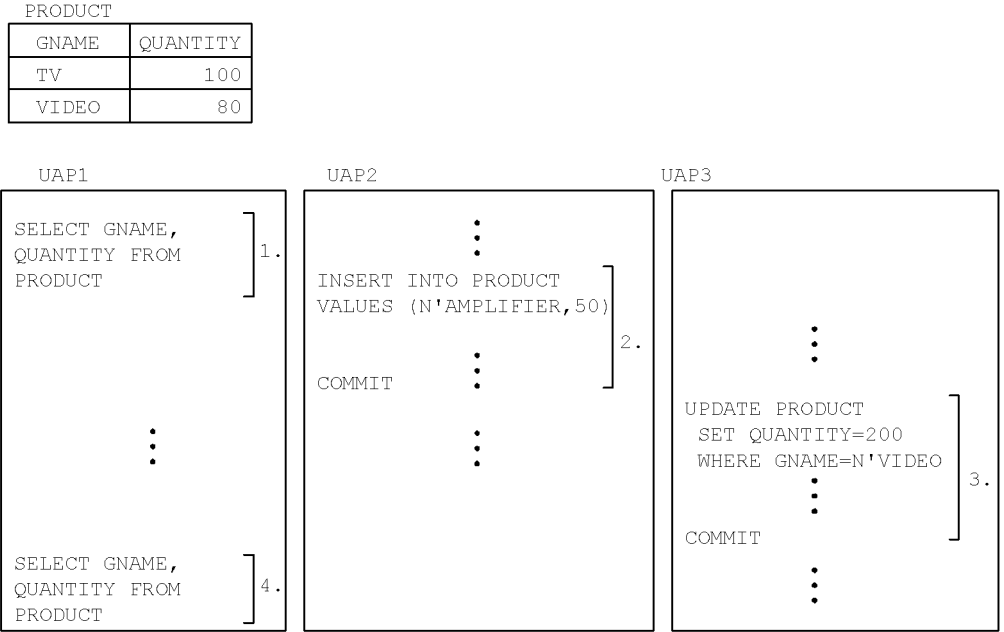
(4) Notes

1. If data guarantee level 0 is specified for a cursor declaration that accompanies an update, the specification is ignored, and level 1 is assumed.
2. If data guarantee level 0 or 1 is specified for a holdable cursor, the specification is ignored, and level 2 is assumed.

4.6.3 Example of search results when a data guarantee level is specified

Figure 4-52 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 4-52: Example of search results when a data guarantee level is specified



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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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.

4. UAP Design for Improving Performance and Handling

Data guarantee level	UAP1 search results		Explanation
	1.	4.	
2	TV 100 VIDEO 80	TV 100 VIDEO 80 AMPLIFIER 50	<ul style="list-style-type: none"> • 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.

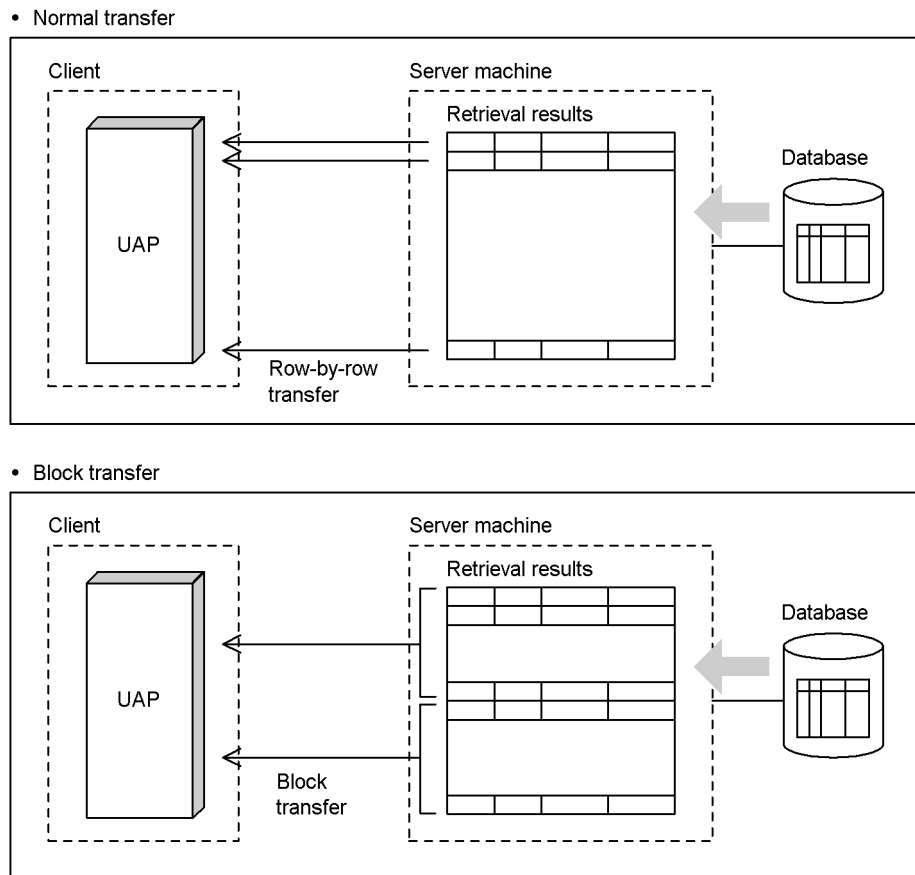
4.7 Block transfer facility

(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.

Figure 4-53 shows an overview of the block transfer facility.

Figure 4-53: Overview of block transfer facility



(2) Usage method

The block transfer facility is executed when both of the following conditions are satisfied:

4. UAP Design for Improving Performance and Handling

1. When at least two values are specified in client environment definition `PDBLK` or when at least one value is specified in `PDBLKBUFSIZE`
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 `PDBINARYBLK` 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 `PDBLKBUFSIZE` to specify the communication buffer size between the server and the client.

For retrievals in which the number of rows to be extracted (`PDBLK` specification value) is large, specifying `PDBLKBUFSIZE` 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 8 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.

PDBLK specification value	PDBLKBUFSIZE specification value	
	0	1 or higher
1	Block transfer facility does not apply.	<p>Number of rows = $\text{Min}(X, 4096)^*$</p> <p>X:</p> <p>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).</p> $(a - b) \geq \sum_{i=1}^n ci \text{ (unit: bytes)}$ <p>ci: Data length of the i-th row in the search results received with the <code>FETCH</code> statement</p> <p>a: Specified buffer size (PDBLKBUFSIZE value \times 1024)</p> <p>b: Header information and other information ($864 + 22 \times d + 2 \times e$)</p> <p>The d and e variables in the calculation expression for b are described below.</p> <p>d: Number of retrieval items specified in the <code>SELECT</code> clause</p> <p>e: Number of <code>BINARY</code>-type selection items in retrieval items specified in the <code>SELECT</code> clause</p>
2 or higher	Number of rows = PDBLK value	<p>Number of rows = $\text{MIN}(X, Y)^*$</p> <p>X: Number of rows that can be stored in specified buffer size (same as X shown above)</p> <p>Y: PDBLK value</p>

* Certain SQL statements may be able to transfer more than the calculated number of rows.

(5) Notes

- 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 `PDBLKBUFSIZE` 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

fetches 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
:xcpcod, :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",
            xcpcod[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",
                xcpcod[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 `QUANTITY` (`INTEGER`) columns.

```

#include <pdbsqllda.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(;;){

```

4. UAP Design for Improving Performance and Handling

```
array_num = 50 ;
EXEC SQL
    FETCH CR3 USING DESCRIPTOR :xsqlda BY :array_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;
}

```

4. UAP Design for Improving Performance and Handling

```
        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 <pdbsqllda.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(;;){
    array_num = 50 ;
    EXEC SQL
        FETCH CR3 USING DESCRIPTOR :xsqlda BY :array_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;

```

4. UAP Design for Improving Performance and Handling

```
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.

4. UAP Design for Improving Performance and Handling

```
#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;

```

4. UAP Design for Improving Performance and Handling

```

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 <pdbsqllda.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) */

```

4. UAP Design for Improving Performance and Handling

```

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 <pdbsqllda.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;

```

4. UAP Design for Improving Performance and Handling

```

----- (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     xquantity[50]; /* For specifying update value to QUANTITY
(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) */
```

4. UAP Design for Improving Performance and Handling

```
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 8 System Definition*
- The SQL optimization options description in the manual *HiRDB Version 8 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 AP process in a HiRDB client. This facility can be used only when creating or executing a UAP in the HP-UX, Solaris, or Windows environment.

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 AP process 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 AP process can connect to HiRDB servers on multiple server computers and execute SQL statements because each connection can be connected to a HiRDB 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:

4. UAP Design for Improving Performance and Handling

- HP-UX 11.0
- Solaris
- AIX 5L (single thread)
- Linux (single thread)
- Windows

(2) Processing overview

Figures 4-54 through 4-58 show an overview of multi-connection facility processing.

Figure 4-54: Overview of multi-connection facility processing (when multithreading is not used)

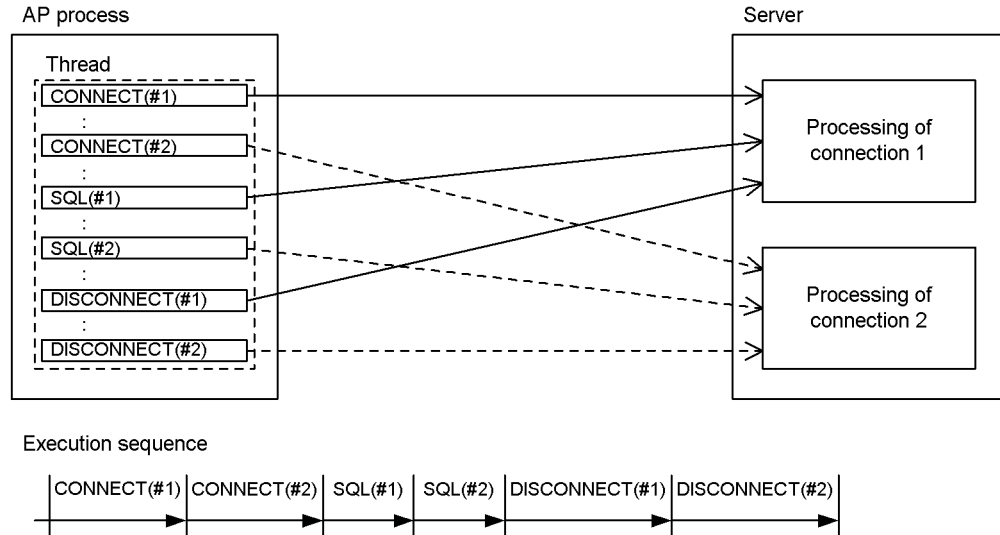
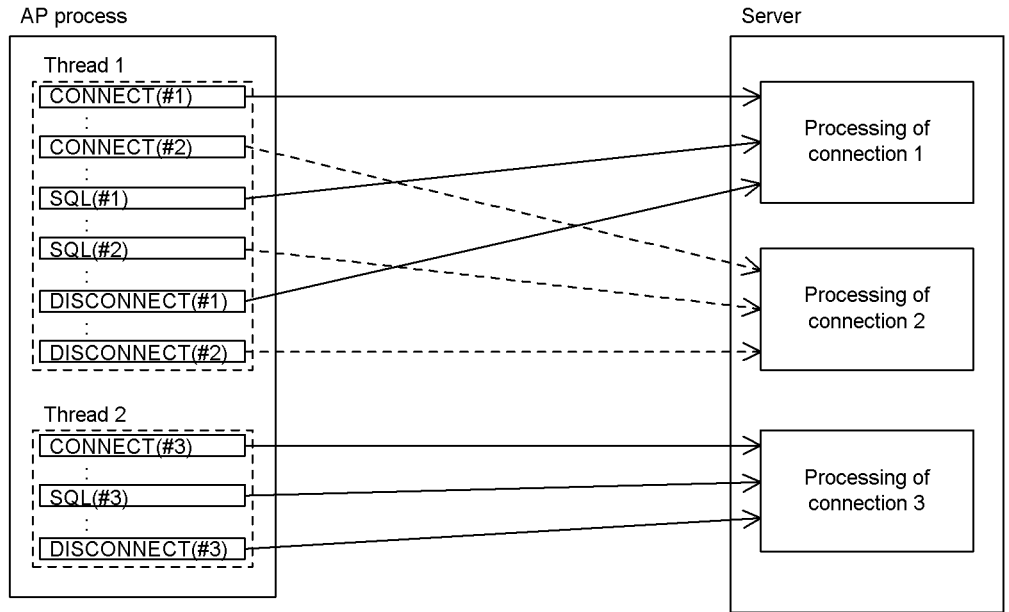
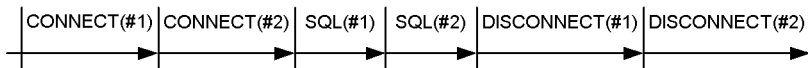


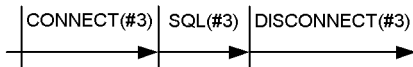
Figure 4-55: Overview of multi-connection facility processing (when multithreading is used)



Execution sequence of thread 1

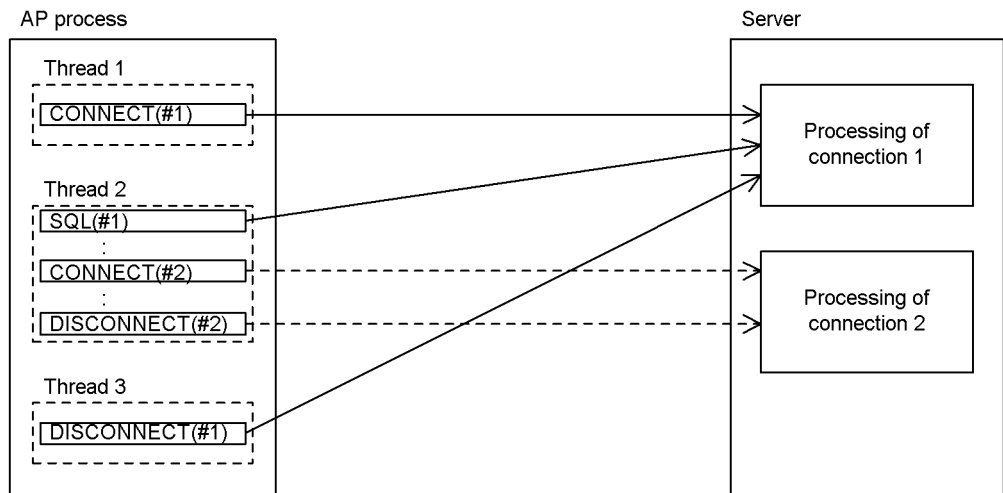


Execution sequence of thread 2



Note: The threads can execute SQL statements simultaneously because each connection is independent.

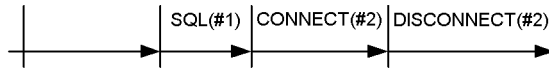
Figure 4-56: Overview of multi-connection facility processing (when a connection is shared by multiple threads)



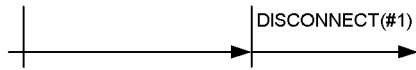
Execution sequence of thread 1



Execution sequence of thread 2

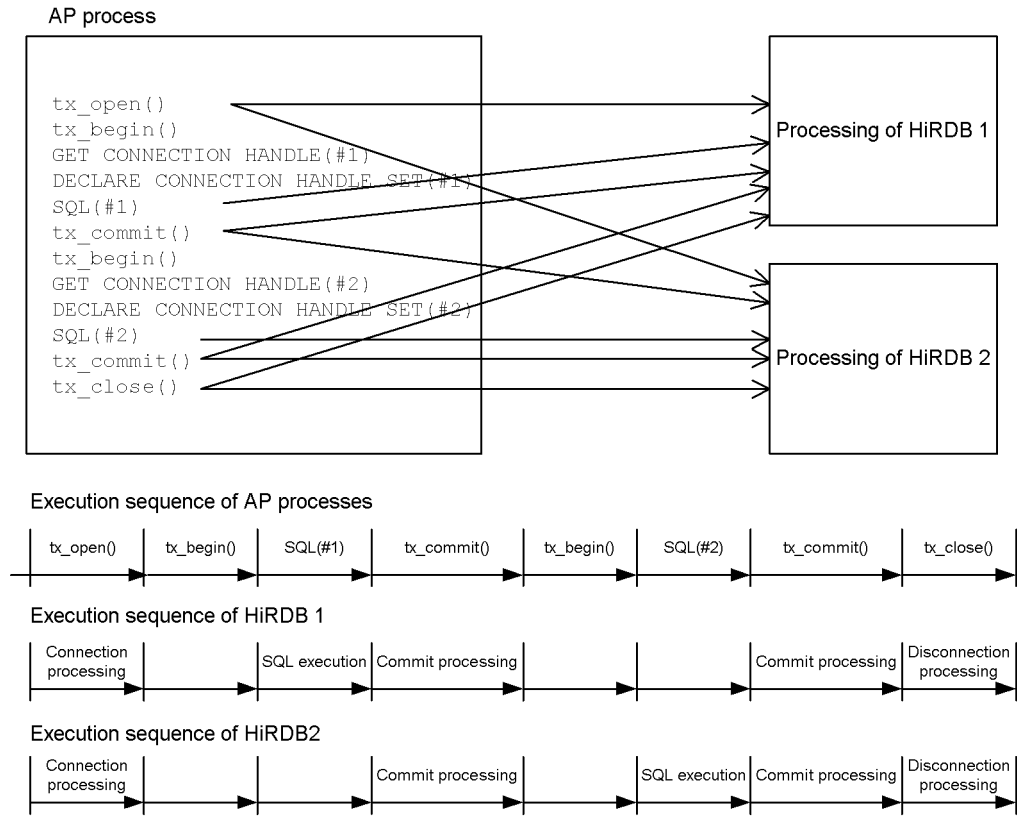


Execution sequence of thread 3



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).

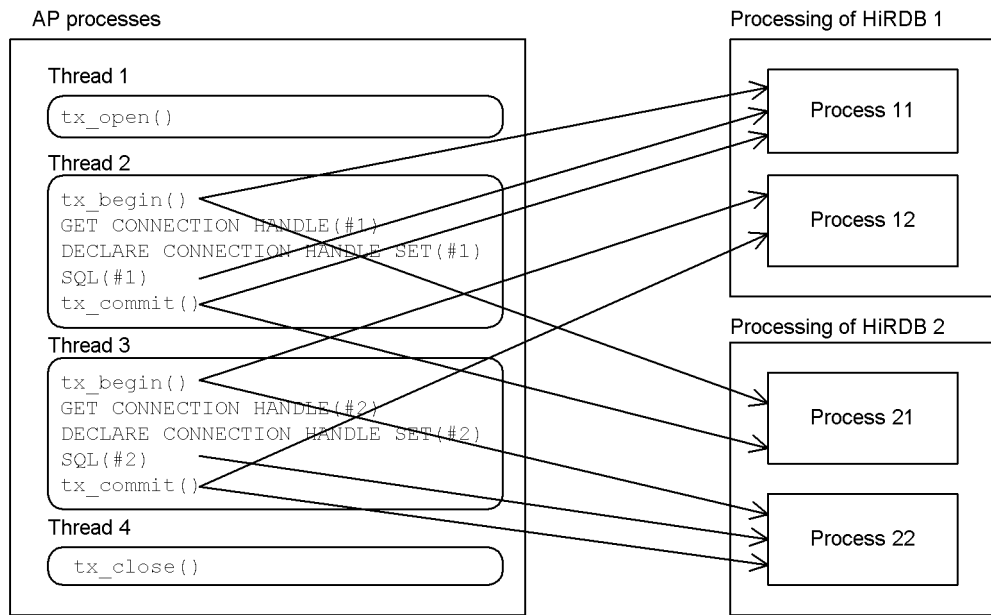
Figure 4-57: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a single-thread OLTP system)

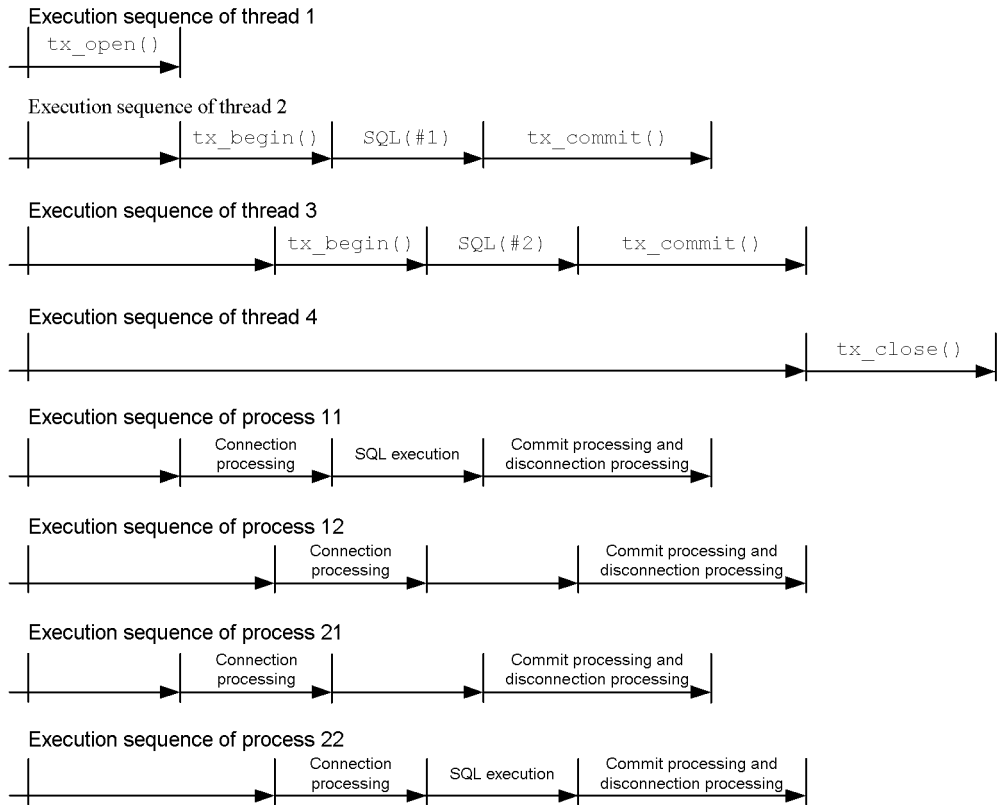


Explanation

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 4-58: Overview of multi-connection facility processing (when an AP uses an X/Open-compliant API in a multi-thread OLTP system)





Explanation

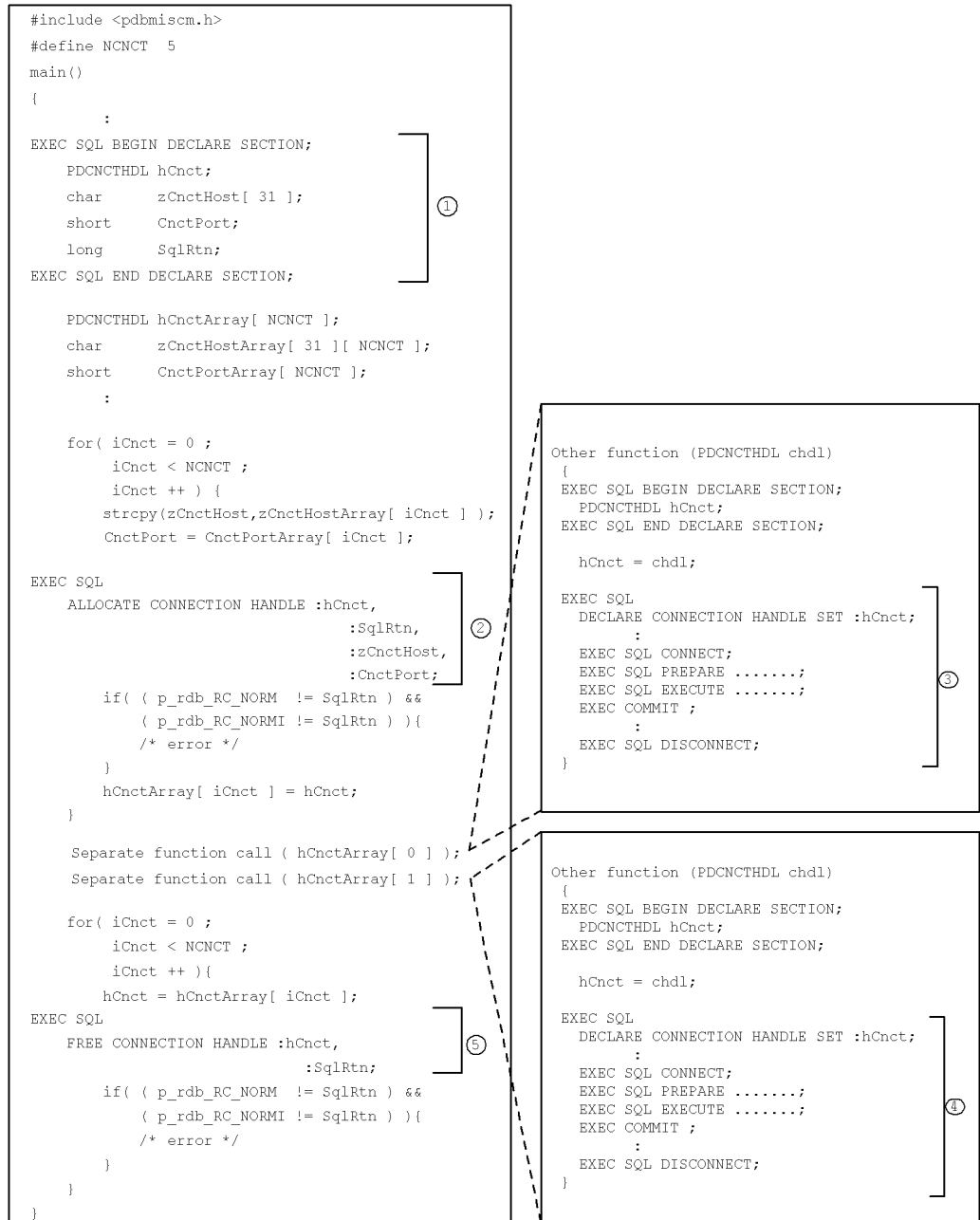
Register HiRDB 1 and HiRDB 2 in the OLTP system beforehand. When `tx_begin()` is executed, the OLTP system connects to all registered HiRDB systems. When an SQL statement is executed, select the connection destination for that SQL statement. Because the individual transactions are independent, SQL statements for different threads can be executed simultaneously.

(3) Coding example

(a) Normal UAPs

Figures 4-59 and 4-60 show coding examples of UAPs that use the multi-connection facility.

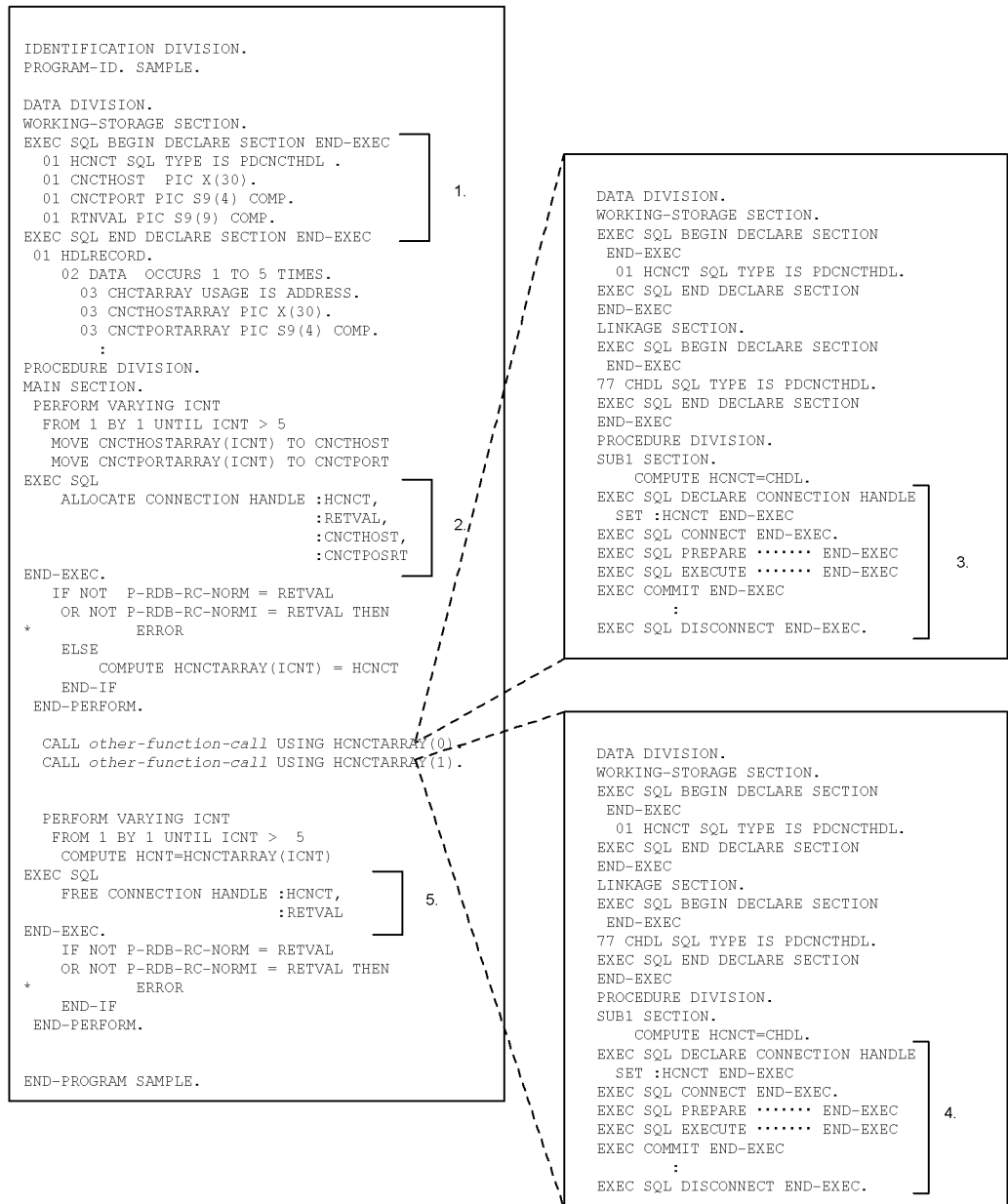
Figure 4-59: Coding example (C) of a UAP that uses the multi-connection facility



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 4-60: Coding example (COBOL) of a UAP that uses the multi-connection facility



Note

Specify the entire SQL, including the SQL prefix and terminator, in the B area (columns 12 to 72).

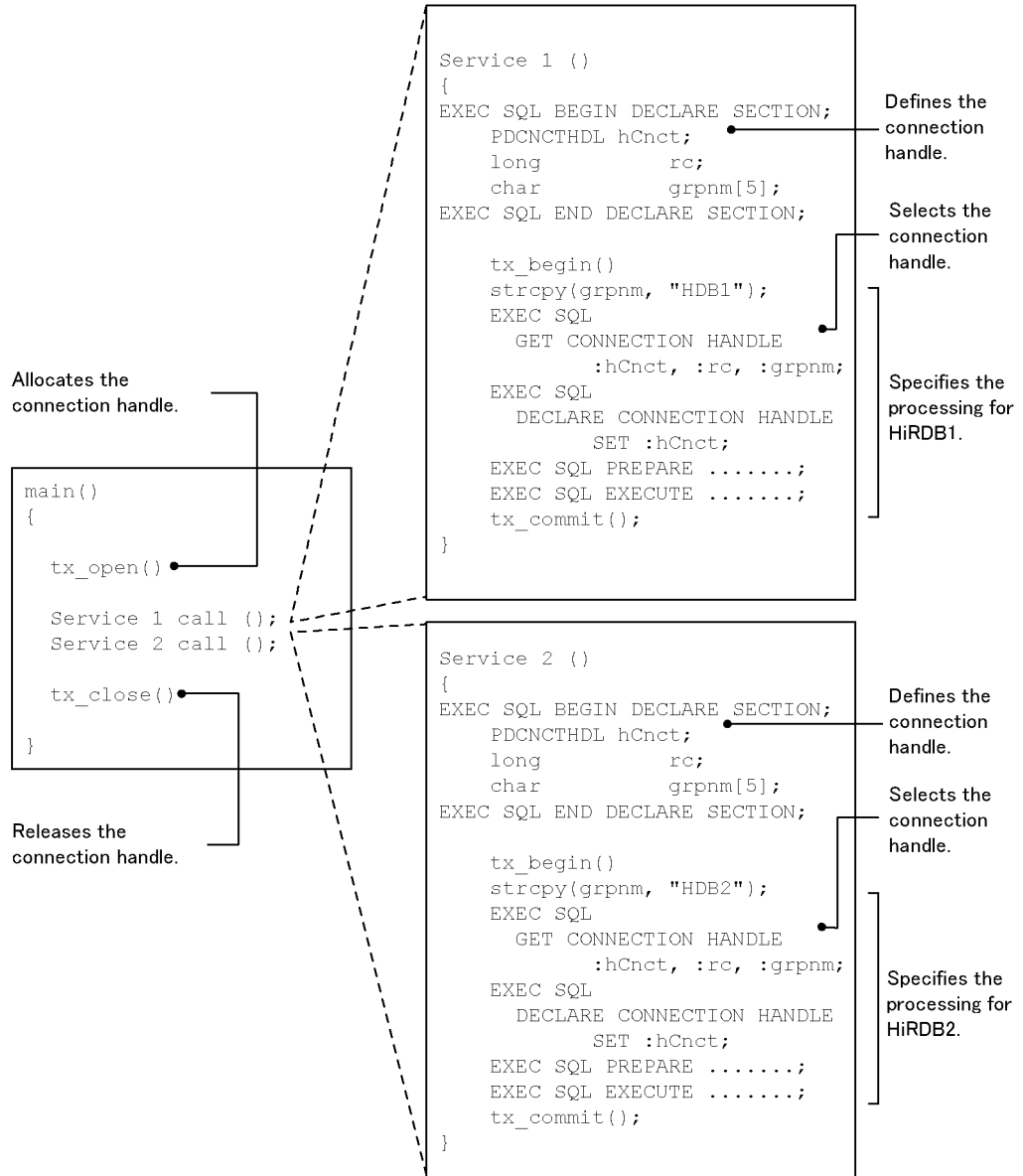
Explanation

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 4-61 and 4-62 show coding examples in which the multi-connection facility is used by UAPs that use an X/Open-compliant API under OLTP.

Figure 4-61: Coding example (C) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP

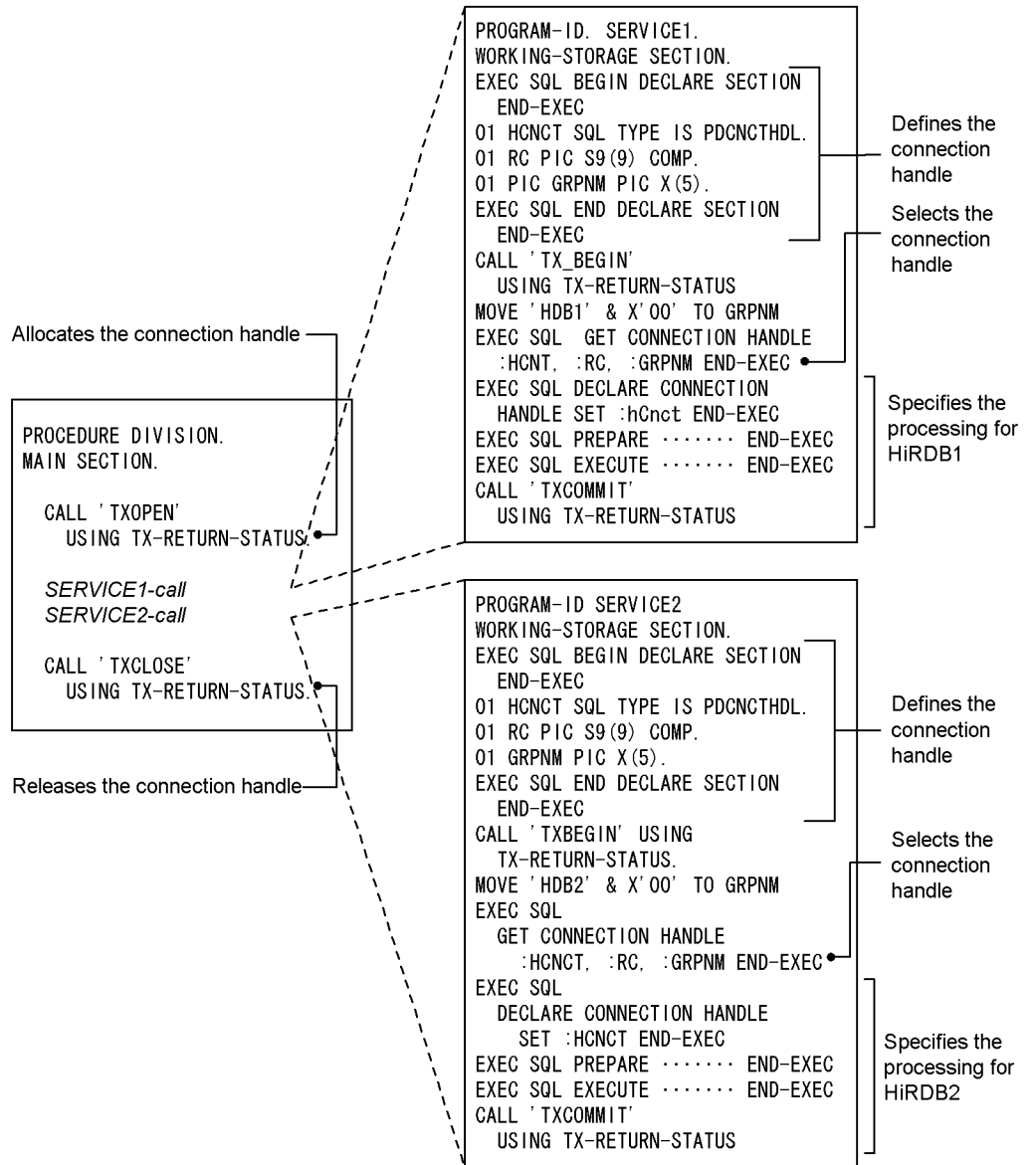


Explanation

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 8 Installation and Design Guide*.

Figure 4-62: Coding example (COBOL) in which the multi-connection facility is used by a UAP that uses an X/Open-compliant API under OLTP



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 8 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 8 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. The multi-connection function in Windows can only be used by UAPs that support multi-threads. Therefore, when specifying the C runtime library to be used in UAP compilation, select the multi-thread DLL (specify **Multithread DLL** in **Compile option: Code generation**).

9. 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 Using tables for managing numbers

(1) When tables should be used

Actual work operations involve numbering, including management of form and document numbers. There are instances when two users may try to get form numbers at the same time. The form numbers must be counted to ensure that when a user requests a form number, he or she does not get the same number that was issued earlier to another user. If a duplicate number is generated, a user may be forced to wait while another user is getting a form number. HiRDB provides a numbering function to minimize the effects of lock-release waiting.

(2) Designing tables

For the most efficient numbering, tables must be designed to minimize the effects of lock-release waiting. HiRDB minimizes the locking effects on tables that manage numbers by providing a facility that releases the lock on a row without waiting for the UAP to commit the transaction. This facility releases the lock when the table update (including addition and deletion) process is completed and disables rollback after that point. To use this facility, the table designer must specify the `WITHOUT ROLLBACK` option in the `CREATE TABLE` statement when defining the table.

(3) Conditions for applying `WITHOUT ROLLBACK` to operations

If the `WITHOUT ROLLBACK` option is specified when a table is defined, rollback becomes disabled as soon as a row is updated. Thus, if the UAP or the HiRDB system terminates abnormally, the active table with obtained numbers is rolled back correctly when the HiRDB system is restarted, and data integrity is maintained. However, the point to which the process that updates the number management table was rolled back is unknown. In this case, the operation has assigned a number but is no longer using it. Therefore, the `WITHOUT ROLLBACK` option is not suitable for operations that are adversely affected if a number is skipped. Use the `WITHOUT ROLLBACK` option only if the numbers do not have to be continuous.

(4) Example of a table that manages numbers

Figure 4-63 shows an example of a table that manages numbers.

Figure 4-63: Example of a table that manages numbers

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 details about the table definition example (specification of `without ROLLBACK` option), see the *HIRDB Version 8 Installation and Design Guide*.

(5) Example of a numbering application program

Following is an example of a numbering application program. The application program that manipulates the number management table and the operation table is assumed to be the same transaction.

This table manages form numbers and document numbers. In the following example, the SQL obtains the latest form number from the number management table and uses it in the operation.

```

INSERT INTO number-management-table VALUE ('form-number',1)      1
:
DECLARE CUR1 CURSOR FOR                                          2
SELECT number FROM number-management-table
WHERE type='form-number' FOR UPDATE OF number
OPEN CUR1                                                         3
FETCH CUR1 INTO :x_number                                       4
UPDATE number-management-table SET number=:x_number+1        5
WHERE CURRENT OF CUR1
CLOSE CUR1                                                         6
:
Access to operation table that used obtained number             7
:

```

Explanation

1. Enters 1 as the initial form number value in the number management table.
2. Retrieves the latest form number from the number management table and declares cursor CUR1.

4. UAP Design for Improving Performance and Handling

3. Opens cursor `CUR1`.
4. Fetches the form number into `x_number`.
5. Increments the number for the next user who retrieves a number (sets the latest number). When this process is completed, the lock on the row is released without waiting for the `COMMIT` statement.
6. Closes cursor `CUR1`.
7. Executes the user-defined operation based on the form number that was fetched into `x_number`.

Steps 3 through 7 are repeated for each number.

(6) Considerations when managing multiple number types

(a) Lock processing

If multiple rows are stored in a table for which the `WITHOUT ROLLBACK` option is specified and an index is not defined for that table, all rows are temporarily locked because the retrieval targets all rows. In such cases, a lock-release wait may occur between the form number numbering process and the document number numbering process. To avoid lock-release wait, specify `YES` for `PDLOCKSKIP` in the client environment definitions. This sets a search using condition evaluation with no lock. When a search using condition evaluation with no lock is set, the rows are not locked during retrieval processing, and only the rows that satisfy the retrieval condition are locked.

(b) Rollback processing

When handling multiple number types, do not execute processes that update multiple rows with one SQL execution. Lock release and rollback become disabled in each row when update processing for that row is completed. Therefore, if a UAP that updates multiple rows terminates abnormally, the HiRDB system may not be able to roll back the update of some of the rows.

(7) Example of numbering with a stored procedure

Registering a numbering process as a stored procedure is useful because numbering is often processed according to a fixed pattern.

Examples 1 through 3 show table definitions and stored procedures.

Example 1 defines a table with the `WITHOUT ROLLBACK` specification and uses a stored procedure to assign sequential numbers.

This example assigns number values that start from an initial value of 1 and increment by 1 up to the maximum integer value.

If the maximum integer value is exceeded, the HiRDB system returns an overflow error. However, if the default value setting facility (`PDDFLNVAL`) is used, a null

value is set instead of an overflow error, and a NOT NULL constraint violation error occurs. If a row with the initial value is not inserted beforehand, the table will not have any rows, and a cursor positioning error (the cursor is not positioned on any row) will occur when the UPDATE statement is executed. If multiple rows are inserted beforehand, the second and 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
<Sequence number assignment>
9
CALL owner_id.nextval (OUT:xnext_no);
:
Process that uses sequence number xnext_no that was assigned
:
CALL owner_id.nextval (OUT:xnext_no);
:

```

Explanation

1. Defines the `owner_id.sequence_tbl` key for assigning INTEGER values.
2. Defines the `owner_id.nextval` procedure, which assigns a sequence number and outputs it with the `next_no` parameter.
3. Retrieves the value in the `sequence_no` column of the `owner_id.sequence_tbl` table.
4. Sets the retrieved value to the `next_no` parameter.
5. Increments the `sequence_no` column in the `owner_id.sequence_tbl` table by 1.

6. Commits the transaction to validate the table and procedure definitions.
7. Inserts a row that has the initial value of 1 using an `INSERT` statement.
8. Commits the transaction to validate the inserted row.
9. Calls the `owner_id.nextval` procedure with a `CALL` statement, assigns a sequence number, and gets the value with the `next_no` parameter. The next sequence number is assigned each time a `CALL` statement is executed.

Example 2 defines a table with the `WITHOUT ROLLBACK` specification and uses a stored procedure to assign two or more types of sequence numbers.

For each sequence number identification key, this example assigns number values that start from an initial value of 1 and increment by 1 up to the maximum integer value.

If the maximum integer value is exceeded, the HiRDB system returns an overflow error. However, if the default value setting facility (`PDDLVAL`) is used, a null value is set instead of an overflow error, and a `NOT NULL` constraint violation error occurs. If a row with the initial value is not inserted beforehand for a sequence value identification key, the table will not have any rows and a cursor positioning error (the cursor is not positioned on any row) will occur when the `UPDATE` statement is executed. If multiple rows are inserted beforehand for a sequence number identification key, the second and subsequent rows are ignored.

Notes

1. An index cannot be defined for a table when `WITHOUT ROLLBACK` is specified. To prevent lock contention, specify `PDLOCKSKIP=YES` in the client environment definitions.
2. Because an index cannot be defined for a table when `WITHOUT ROLLBACK` is specified, divide the tables and the procedures if the number of sequence number types is extremely large.

```
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
:
(Initial value row is inserted for each sequence number type)

<Assignment of 'key_value_1' sequence number>
.....9
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
:
Processing when sequence number xnext_no assigned to
'seqkey_value_1' is used
:
xinput_key <-- 'key_value_1'
CALL owner_id.nextval(IN:xinput_key,OUT:xnext_no);
:

<'Assignment of key_value_2' sequence number>
.....9
xinput_key <-- 'key_value_2'
CALL owner_id.nextval(IN :xinput_key,OUT:xnext_no);
:
Processing when sequence number xnext_no assigned to
'key_value_2' is used
:
xinput_key <-- 'key_value_2'

```

```
CALL owner_id.nextval (IN:xinput_key,OUT:xnext_no);
:
```

Explanation

1. Defines the `owner_id.sequence_tbl` table for assigning `INTEGER` values to each sequence number identification key.
2. Defines the `owner_id.nextval` procedure, which enters a sequence number identification key using the `input_key` parameter, assigns a sequence number for that key, and outputs the sequence number with the `next_no` parameter.
3. Specifies a sequence number identification key for the `sequence_key` column in the `owner_id.sequence_tbl` table and retrieves the value in the `sequence_no` column.
4. Sets the retrieved value to the `next_no` parameter.
5. Increments the `sequence_no` column in the `owner_id.sequence_tbl` table by 1.
6. Commits the transaction to validate the table and procedure definitions.
7. Inserts a row that has the initial value of 1 using an `INSERT` statement, for each sequence number identification key.
8. Commits the transaction to validate the inserted row.
9. Calls the `owner_id.nextval` procedure with a `CALL` statement, assigns a sequence number, and gets the value with the `next_no` parameter. The next sequence number is assigned each time a `CALL` statement is executed.

Example 3 defines a table with the `WITHOUT ROLLBACK` specification and uses a stored procedure to assign sequence numbers that are incremented between a minimum value and a maximum value.

If a row with the initial value is not inserted beforehand, the table will not have any rows, and a cursor positioning error (where the cursor is not positioned on any row) will occur when the `UPDATE` statement is executed. If multiple rows are inserted beforehand, the second and 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=214783647 THEN
    SET update_no=-214783648;
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
<Sequence number assignment>        10
CALL owner_id.nextval(OUT:xnext_no);
    :
    Process that uses sequence number xnext_no that was assigned
    :
CALL owner_id.nextval(OUT:xnext_no);
    :

```

Explanation

1. Defines the `owner_id.sequence_tbl` key for assigning INTEGER values.
2. Defines the `owner_id.nextval` procedure, which assigns a sequence number to the `sequence_no` column of the `owner_id.sequence_tbl` table. The sequence numbers increment in values of 1, a minimum value of -2, 147, 483, 648, and a maximum value of 2, 147, 483, 647. The number incremented after the maximum number is the minimum value.
3. Retrieves the value in the `sequence_no` column of the `owner_id.sequence_tbl` table.
4. Sets the retrieved value to the `next_no` parameter.
5. If the retrieved value is the maximum value of 2, 147, 483, 647, this section sets the minimum value of -2, 147, 483, 648, as the next sequence number. Otherwise, this section increments the retrieved value by 1 and sets the result as the next sequence number.
6. Updates the `sequence_no` column in the `owner_id.sequence_tbl` table to the next sequence number value.
7. Commits the transaction to validate the table and procedure definitions.
8. Inserts a row that has the initial value of 1 with an `INSERT` statement.
9. Commits the transaction to validate the inserted row.

4. UAP Design for Improving Performance and Handling

10. Calls the `owner_id.nextval` procedure with a `CALL` statement, assigns a sequence number, and gets the value with the `next_no` parameter. The next sequence number is assigned each time a `CALL` statement is executed.

4.12 Narrowed search

4.12.1 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.

4.12.2 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 8 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 8 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 8 Installation and Design Guide*.

4.12.3 Search using lists

This section explains the method of searching using lists. Figure 4-64 shows an example of a search that uses a list.

Figure 4-64: 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.

Create LIST2 by selecting rows that have \$50.00 or higher as the price.

SQL statement

```
ASSIGN LIST LIST1
FROM (STOCK) WHERE PNAME=N'SKIRT'
```

SQL statement

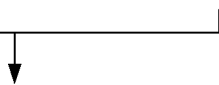
```
ASSIGN LIST LIST2
FROM (STOCK) WHERE PRICE >=50.00
```

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

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



4. UAP Design for Improving Performance and Handling

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

4.12.4 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 may 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 <code>DROP LIST</code> 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.

4.12.5 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.

4.12.6 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 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.

4.13 File output facility for BLOB data

4.13.1 What is the file output facility for BLOB data?

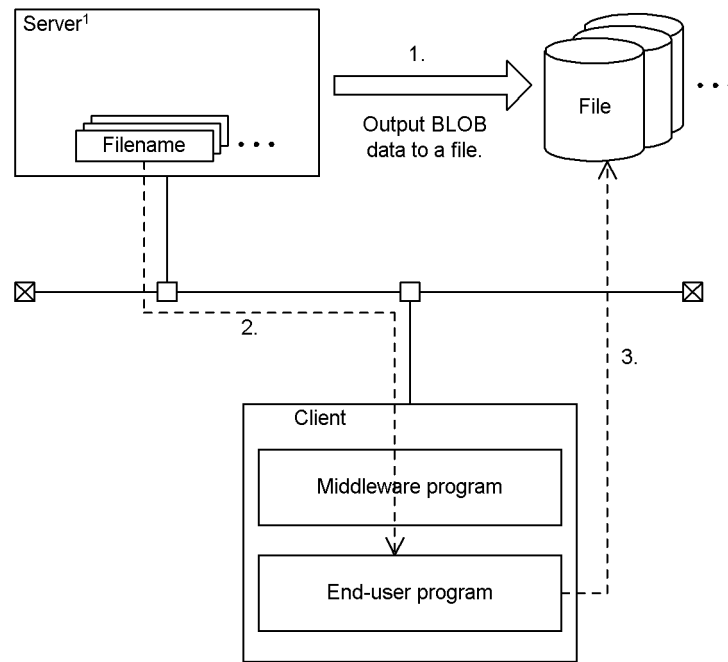
Before you can search BLOB data, you must prepare a memory area for storing BLOB data in the client. You will also need a send buffer for BLOB data returns in the server and memory for a receive buffer that accepts BLOB data in the client library. Consequently, a large amount of memory must be allocated 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.

Figure 4-65 shows an overview of the file output facility for BLOB data.

Figure 4-65: Overview of the file output facility for BLOB data



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.

4.13.2 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.

4.13.3 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 8 SQL Reference* manual.

4.13.4 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.

4.13.5 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
```

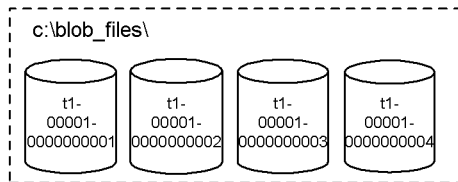


Retrieval results

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.

4. UAP Design for Improving Performance and Handling

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
```

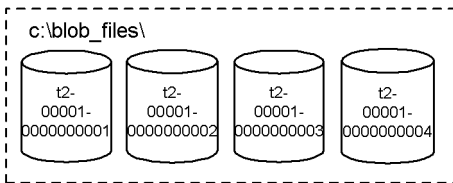


Retrieval results

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



4.14 Addition update and partial extraction facility for BLOB and BINARY data

4.14.1 What is the addition update and partial extraction facility for 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. The addition update and partial extraction facility for BLOB and BINARY data is used for solving this problem.

* 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

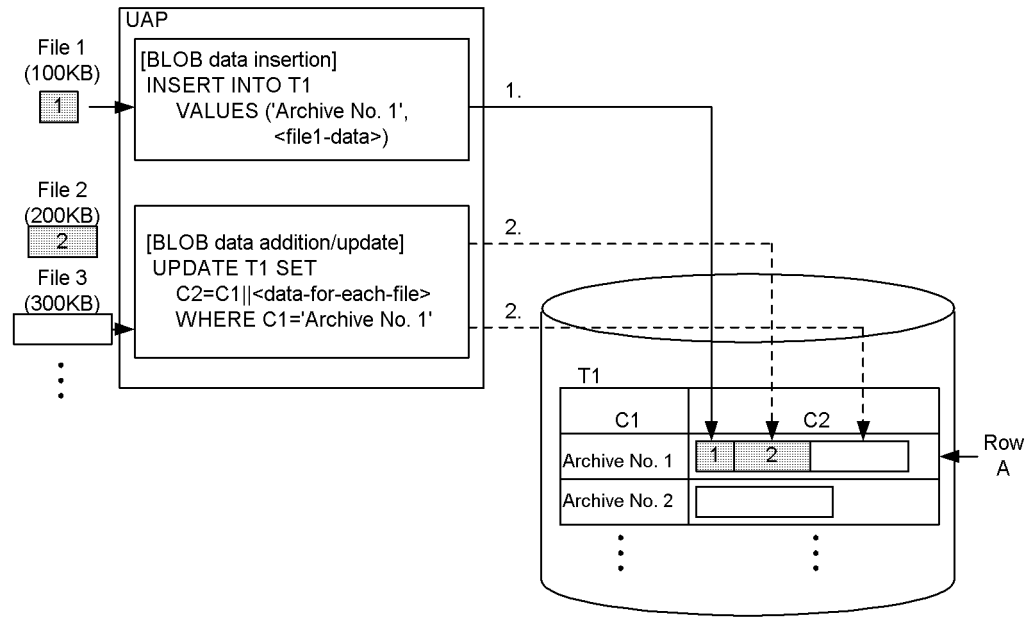
To extract only the specified portion from BLOB or BINARY data, specify the SUBSTR scalar function. The amount of memory used is suppressed to the amount of data to be extracted.

4.14.2 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.

4. UAP Design for Improving Performance and Handling

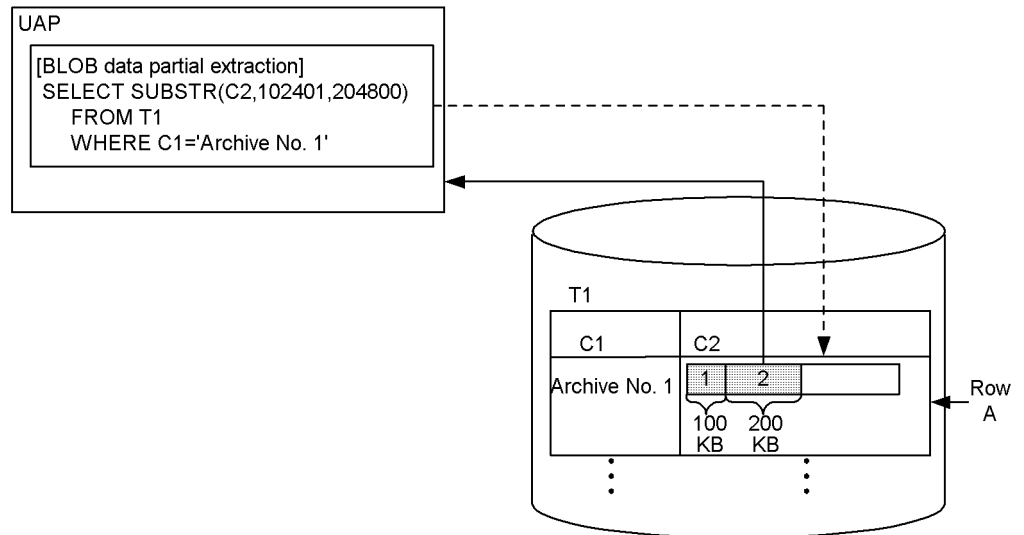


Explanation

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 $(100 \times 1024 + 1) = \text{byte } 102401$) of the data column for file 2 in column C2 of row A. Only the amount of data equivalent to the length of the data column in file 2 ($200 \times 1024 = 204800$ bytes) is extracted.

4.14.3 Notes about using the addition update and partial extraction facility for BLOB and BINARY data

When you use the addition update and partial extraction facility for `BLOB` and `BINARY` data, note the following points:

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 8 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

4. UAP Design for Improving Performance and Handling

improve the performance of `BLOB` data insertion, addition update, or partial extraction, you should set the data length to units of $128 \times 1024 \times n$ bytes (n is a nonzero positive integer).

4. The `BINARY` data described in this subsection refers to `BINARY` data with a minimum defined length of 32,001 bytes.

4.15 Retrieve first n records facility

4.15.1 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 8 SQL Reference*.

4.15.2 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 8 Installation and Design Guide*.

4.15.3 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 8 Command Reference*.

4.16 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.

4.16.1 Application criteria

If the HiRDB server is executing the system reconfiguration command* or update to the HiRDB update version*, the HiRDB client waits until that process ends. 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 is sometimes unable to detect that the system reconfiguration command or update to the HiRDB update version is being executed, and a communication processing error may occur. If you know ahead of time that the system reconfiguration command or update to the HiRDB update version is to be executed, use 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 system reconfiguration command or update to the HiRDB update version is being executed.

* The system reconfiguration command is executed with the `pdchgconf` command. Update to the HiRDB update version is executed with the `pdprgcopy` and `pdprgrenew` commands. For details about these operation commands, see the manual *HiRDB Version 8 Command Reference*.

4.16.2 Reconnect timings

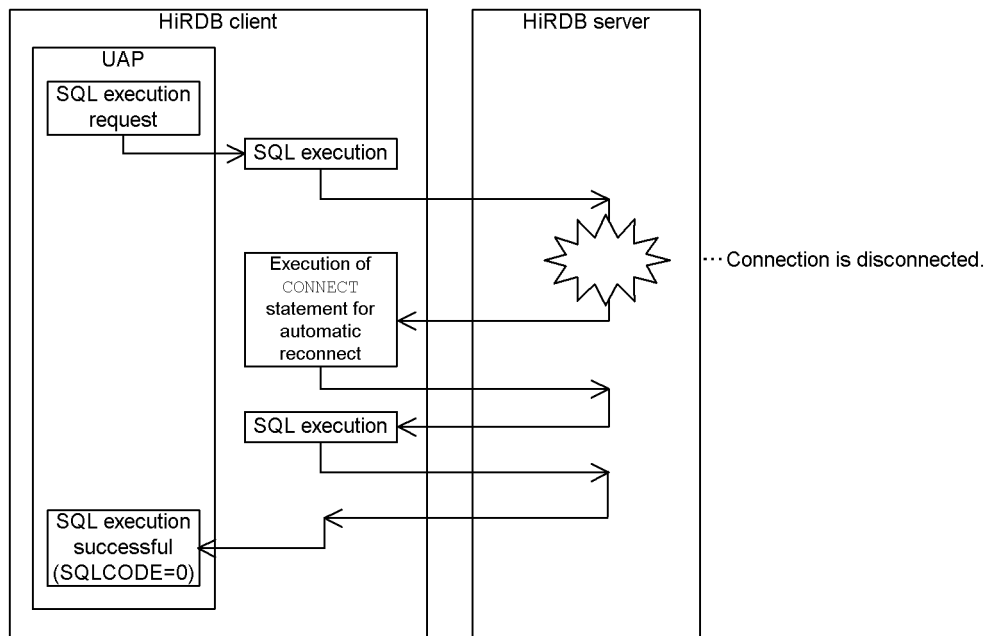
Reconnection is performed at the following times:

- 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. Figure 4-66 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 4-66: 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)



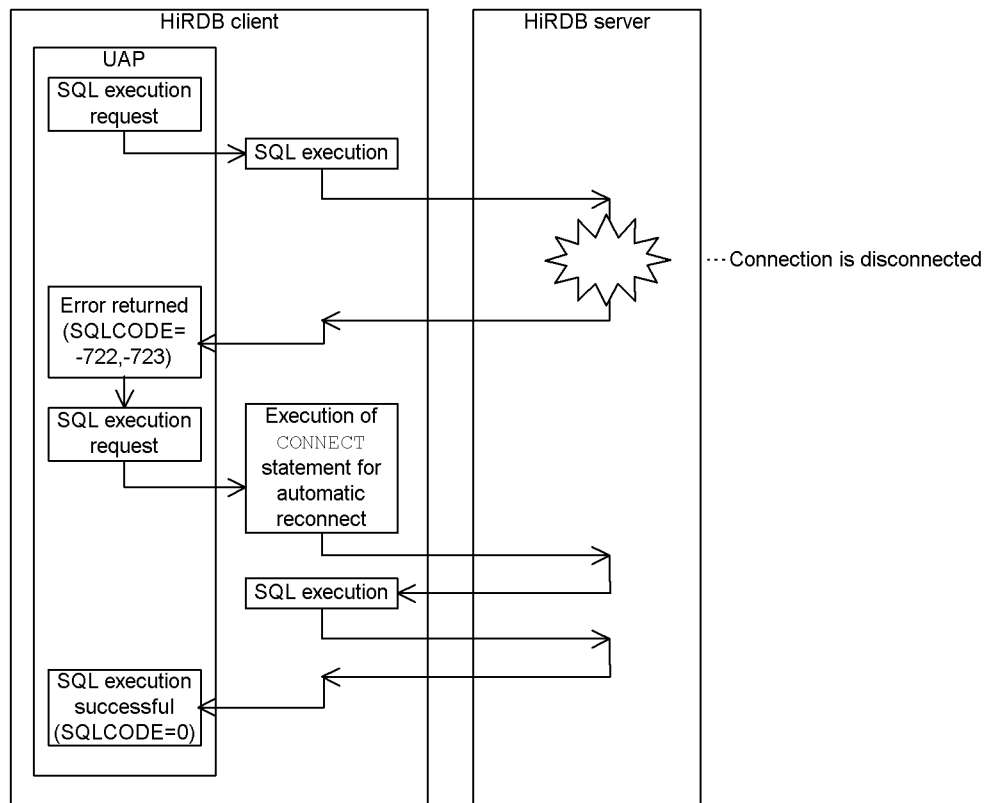
(2) When the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement

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. Figure 4-67 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 4-67: Reconnect timing (when the HiRDB client executes an SQL statement while the HiRDB server is processing the transaction for the previous SQL statement)



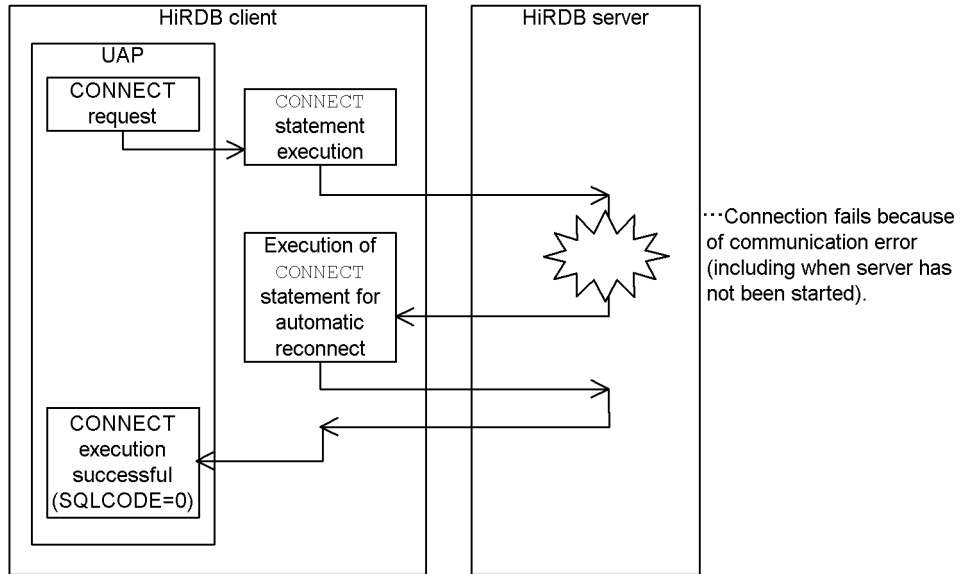
(3) When the HiRDB client executes the *CONNECT* statement

If the HiRDB client executes the *CONNECT* statement and the connection fails because of a communication error, the automatic reconnect facility executes reconnect processing.

Figure 4-68 shows the reconnect timing (when the HiRDB client executes the

CONNECT statement).

Figure 4-68: Reconnect timing (when the HiRDB client executes the CONNECT statement)



4.16.3 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. However, if the request from the UAP specifies a statement other than the `CONNECT` statement, HiRDB uses the `PDCWAITTIME` time to monitor the processing time. If the processing time for automatic reconnect exceeds the `PDCWAITTIME` time, HiRDB aborts the automatic reconnect process and returns an error to the UAP.

4.16.4 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 driver¹ or DABroker for JAVA² 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.

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.

4.17 Locator facility

4.17.1 What is the locator facility?

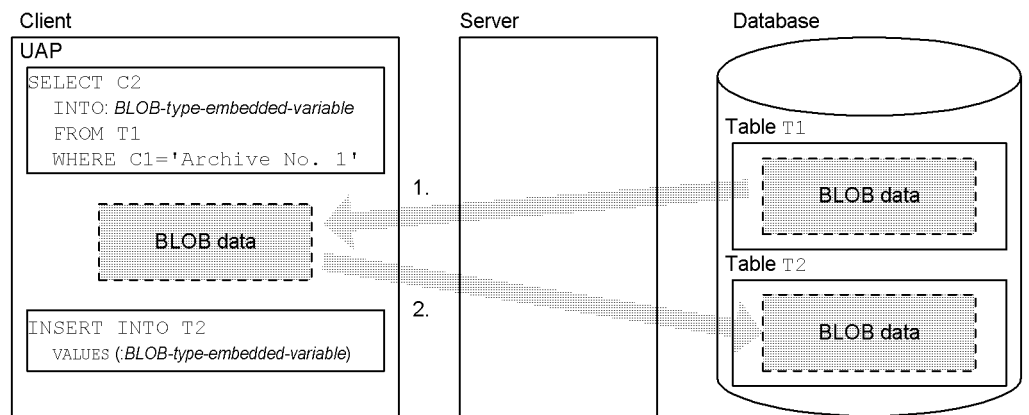
For the client UAP to accept retrieved `BLOB` or `BINARY` data in embedded variables of that data type, the client must have a memory area available for storing the data. Therefore, the memory resources of the client become overburdened when large object data is retrieved. Furthermore, the amount of 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.

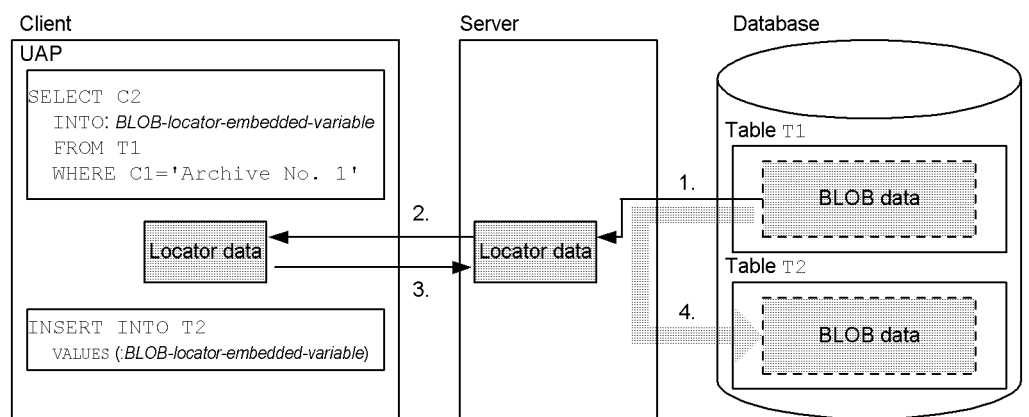
Figure 4-69 shows an overview of the locator facility.

Figure 4-69: Overview of the locator facility

- When the locator facility is not used



- When the locator facility is used



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.

4.17.2 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.

4.17.3 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.

4.17.4 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

4. UAP Design for Improving Performance and Handling

overburdened. To prevent this problem, use the `FREE LOCATOR` statement to invalidate locators that are no longer necessary.

4.18 Facility for returning the total number of hits

4.18.1 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 8 SQL Reference*.

4.18.2 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.

4.18.3 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 8 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

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 a dictionary table, see *F.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

Table 5-1 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.*)

Table 5-2 shows the HiRDB operations for combinations between receive and send functions for passing inter-function values.

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 operation
Not specified	Not specified	Can be executed
	Specified	

Receive function for passing inter-function values	Send function for passing inter-function values	HiRDB operation
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. Table 5-3 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:

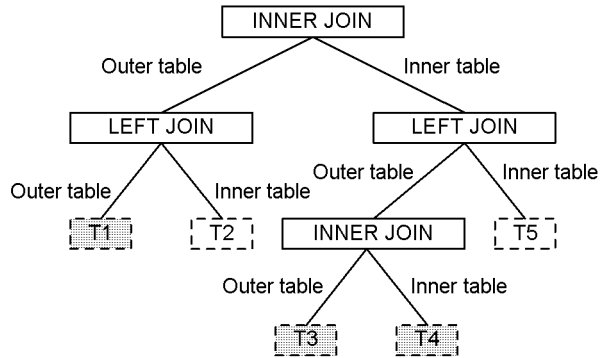
1. Only a base table column specification can be specified in the first argument. Also, the column cannot be an external-referencing column.
2. Arguments that include the following value expressions cannot be specified in any argument except the first:
 - Column specifications, except external-referencing 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 an 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.

Table 5-4 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:

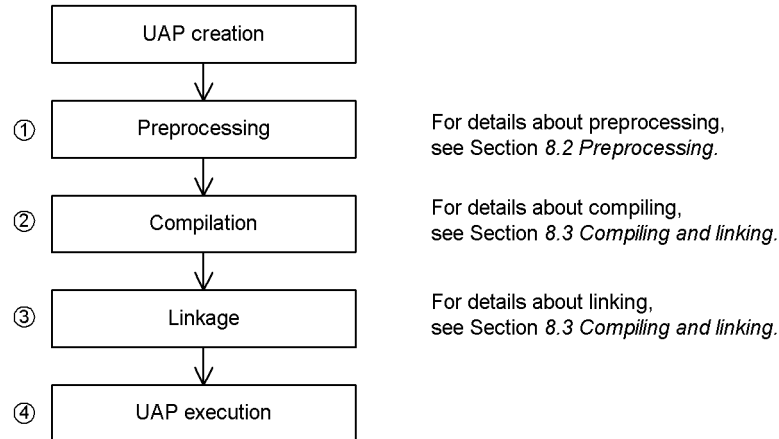
- 6.1 Types of HiRDB clients
- 6.2 Environment setup procedure for HiRDB clients
- 6.3 HiRDB client installation
- 6.4 Organization of directories and files for a HiRDB client
- 6.5 Setting the hosts file
- 6.6 Client environment definitions (setting environment variables)
- 6.7 Registering an environment 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.



The operations available to each type of HiRDB client are as follows:

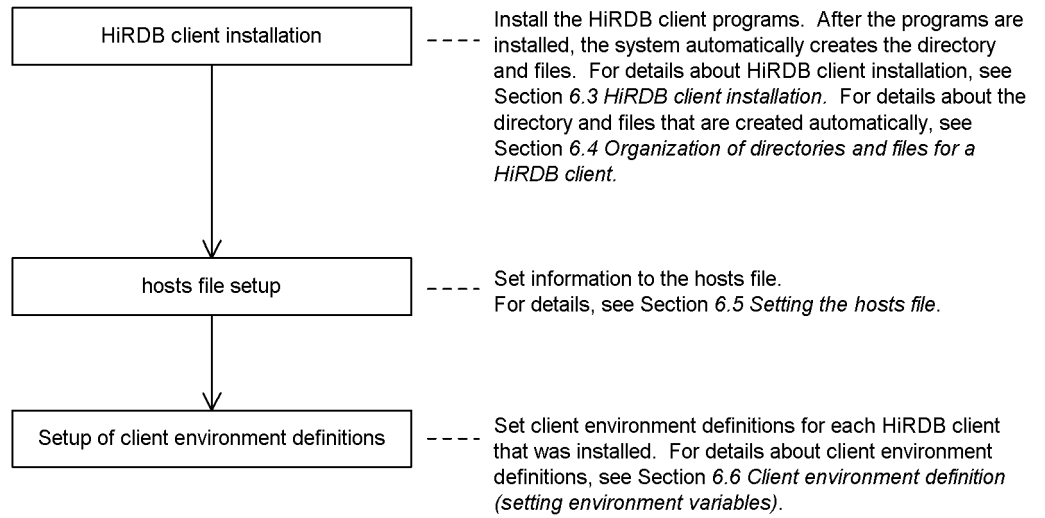
- HiRDB/Developer's Kit
(1) - (4) are available.
- HiRDB/Run Time
(4) only is available. Because the programs provided for the HiRDB client are also provided by the HiRDB server, use the HiRDB server functions to execute (1) - (3).

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.



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

For details about installing a HiRDB client on a UNIX client, see the manual for your operating system.

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 your name and company's 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 Method screen is displayed.

Depending on the setup method, you may be able to change the libraries to be installed. To install an OLE DB provider, the JDBC driver, the JBuilder distribution wizard, SQLJ, the ODBC 3.0 driver, and the HiRDB.NET data provider, select **Custom**.

Typical

This method installs the regular libraries and the X/Open-compliant libraries.

Compact

This method installs the regular libraries.

Custom

This method allows you to select which libraries and sample UAPs to install. This method also allows you to select and install an OLE DB provider, the JDBC driver, the JBuilder distribution wizard, SQLJ, the ODBC 3.0 driver, and the HiRDB.NET data provider.

After selecting the setup method, click the **Next** button.

5. Select the components to be installed

The Select Components screen is displayed.

Specify the components to be installed.

After specifying the components to be installed, click the **Next** button. Then, when the Start File Copy screen is displayed, check the current settings.

After checking the current settings, click the **Next** button.

6. Check the installation status

The execution status of the installation is displayed.

After checking the current setup, click the **Next** button to display the following screen, which shows the progress of installation.

Notes about the installation execution are described as follows.

6. Client Environment Setup

- If there is not enough space, you cannot click the **Next** button. In this case, either change the drive or delete unnecessary files to secure enough space.
- To cancel the installation, click the **Cancel** button. If installation is cancelled, start over from step 3.

7. Terminate the installation program

When installation is complete, the Setup Complete screen is displayed.

Notes

- If the software currently being used is returned to a previous version, note that the environment definition file will be set to the assumed value status (initial status) when the software is reinstalled.
- 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 in Windows, you must have Administrator or Power User privileges. If a user who does not have Administrator or Power User privileges installs the client, redistributed files are not updated, nor is the `PATH` system environment variable updated.

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 6-1 to 6-6 list the files and directories that are created automatically during HiRDB client installation on a client machine.

Table 6-1: Files and directories for workstation - HiRDB/Developer's Kit

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
Header files	/HiRDB/ include	SQLCA.CBL	C	C	C	C	C	C	C	C
		SQLDA.CBL	C	C	C	C	C	C	C	C
		SQLIOA.CBL	C	C	C	C	C	C	C	C
		pdbtypes.h	C	C	C	C	C	C	C	C
		pdberrno.h	C	C	C	C	C	C	C	C
		pdbmisc.h	C	C	C	C	C	C	C	C
		pdbmiscm.h	C	C	C	C	C	C	C	C
		pdbsqlda.h	C	C	C	C	C	C	C	C
		pddbhash.h	C	C	C	C	C	C	C	C
		pdauxcnv.h	C	C	C	C	C	C	C	C
		SQLCAM.cb1	C	C	C	C	C	C	C	C
		SQLDAM.cb1	C	C	C	C	C	C	C	C
		SQLIOAM.cb1	C	C	C	C	C	C	C	C
		SQLIOAMTH.cb1	C	C	C	—	—	C	C	C
SQLCAMTH.cb1	C	C	C	—	—	C	C	C		

6. Client Environment Setup

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
Archive files	/HiRDB/ client/ lib	libclt.a	C	C	C	C	C	C	C	C
		libclt64.a	—	—	C ³	—	C ³	—	C ³	—
		libcltxa.a	C	C	C	C	C	C	C	—
		libcltya.a	C	C	C	C	C	C	C	—
		libcltm.a	C	C	C	C	C	—	—	—
		libcltxam.a	NF	NF	NF	NF	NF	—	—	—
		libcltyam.a	NF	NF	NF	NF	NF	—	—	—
		libcltk.a	C	C	C	C	C	C	C	C
		libcltk64.a	—	—	C ³	—	C ³	—	C ³	—
		libclts.a	C	C	C	C	C	C	C	C
Shared library files ²	/HiRDB/ client/ lib	libzclt.sl	C	C	C	C	C	C	C	C
		libzclt64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzcltx.sl	C	C	C	C	C	C	C	—
		libzclty.sl	C	C	C	C	C	C	C	C
		libzcltm.sl	C	C	C	C	C	—	—	—
		libzcltxm.sl	NF	NF	NF	NF	NF	—	—	—
		libzcltym.sl	NF	NF	NF	NF	NF	—	—	—
		libzcltk.sl	C	C	C	C	C	C	C	C
		libzcltk64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzpdodbc.sl	C	C	C	—	—	—	—	—
		libsqlauxf.sl	C	C	C	C	C	C	C	C
		libsqlauxf64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzcltxk.sl	C	C	C	C	C	NF	NF	NF

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
		libzcltyk.sl	C	C	C	C	C	C	C	NF
		libzclts.sl	C	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	—
JDBC driver	/ <u>HiRDB</u> /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
Command utilities	/ <u>HiRDB</u> /client/utl	pdcpp	C	C	C	C	C	C	C	C
		pdocc	C	C	C	C	C	C	C	C
		pdcb1	C	C	C	C	C	C	C	C
		pdocb	C	C	C	C	C	C	C	C
		pdprep	C	C	C ³	C	C ³	C	C ³	C
		pdtrcmgr	C	C	C	C	C	C	C	C
		pdodbcsetup	C	C	C	—	—	—	—	—
	pdodbcconfig	C	C	C	—	—	—	—	—	
	/ <u>HiRDB</u> /bin	pddef	C	C	C	C	C	C	C	C
SQLJ	/ <u>HiRDB</u> /client/lib	pdsq1j.jar	—	C	—	C	—	C	—	C
		libpdparse.sl	—	C	—	C	—	C	—	C
		libpdsq1jn.sl	—	C	—	C	—	C	—	C
		/ <u>HiRDB</u> /client/utl	pdjava	—	C	—	C	—	C	—
JBuilder	/ <u>HiRDB</u> /jba	pdjba35.jar	—	—	—	C	C	—	—	C
		pdjba4.jar	—	—	—	C	C	—	—	C
		pdjba5.jar	—	—	—	C	C	—	—	C

6. Client Environment Setup

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
Message object file	<u>/HiRDB/</u> lib	msgtxt	C	C	C	C	C	C	C	C
Parsing libraries ²	<u>/HiRDB/</u> lib/ sjis	libasqap.sl	C	C	C ³	C	C ³	C	C ³	C
	<u>/HiRDB/</u> lib/ chinese		C	C	C ³	C	C ³	C	C ³	C
	<u>/HiRDB/</u> lib/ lang-c/		C	C	C ³	C	C ³	C	C ³	C
	<u>/HiRDB/</u> lib/ ujis		C	C	C ³	C	C ³	C	C ³	C
Sample source files	<u>/HiRDB/</u> client/ samplep /uap	CREATE.ec	C	C	C	C	C	C	C	C
		SAMPLE1.ec	C	C	C	C	C	C	C	C
		SAMPLE2.ec	C	C	C	C	C	C	C	C
		SAMPLE3.ec	C	C	C	C	C	C	C	C
		sample1.ecb	C	C	C	C	C	C	C	C
		sample.mk	C	C	C	C	C	C	C	C
		inputf1	C	C	C	C	C	C	C	C
		inputf2	C	C	C	C	C	C	C	C

Legend:

HP: HP-UX

HP (32): 32-bit mode HP-UX

HP (64): 64-bit mode HP-UX

Sol: Solaris

Sol (64): 64-bit mode Solaris

AIX: AIX 5L

AIX (64): 64-bit mode AIX 5L

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. For Solaris the suffix is `.so`. For AIX 5L, the suffix is `.a`.

³ The file operates in 64-bit mode.

Table 6-2: Files and directories for HiRDB/Run Time (UNIX client)

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
Archive files	<u>/HiRDB/</u> client/ lib	libclt.a	C	C	C	C	C	C	C	C
		libclt64.a	—	—	C ³	—	C ³	—	C ³	—
		libcltxa.a	C	C	C	C	C	C	C	—
		libcltya.a	C	C	C	C	C	C	C	—
		libcltm.a	C	C	C	C	C	—	—	—
		libcltxam.a	NF	NF	NF	NF	NF	—	—	—
		libcltyam.a	NF	NF	NF	NF	NF	—	—	—
		libcltk.a	C	C	C	C	C	C	C	C
		libcltk64.a	—	—	C ³	—	C ³	—	C ³	—
		libclts.a	C	C	C	C	C	—	—	C
Shared library files ²	<u>/HiRDB/</u> client/ lib	libzclt.sl	C	C	C	C	C	C	C	C
		libzclt64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzcltx.sl	C	C	C	C	C	C	C	—
		libzclty.sl	C	C	C	C	C	C	C	C
		libzcltm.sl	C	C	C	C	C	—	—	—
		libzcltxm.sl	NF	NF	NF	NF	NF	—	—	—

6. Client Environment Setup

Name	Dir ¹	File name	Platform							
			HP	HP (32)	HP (64)	Sol	Sol (64)	AIX	AIX (64)	Linux
		libzcltym.sl	NF	NF	NF	NF	NF	—	—	—
		libzcltk.sl	C	C	C	C	C	C	C	C
		libzcltk64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzpdodbc.sl	C	C	C	—	—	—	—	—
		libsqlauxf.sl	C	C	C	C	C	C	C	C
		libsqlauxf64.sl	—	—	C ³	—	C ³	—	C ³	—
		libzcltxk.sl	C	C	C	C	C	NF	NF	NF
		libzcltyk.sl	C	C	C	C	C	C	C	NF
		libzclts.sl	C	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	—
JDBC driver	/ <i>HiRDB</i> /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
SQLJ Runtime Library	/ <i>HiRDB</i> /client/lib	pdruntime.jar	—	C	—	C	—	C	—	C
Command utilities	/ <i>HiRDB</i> /client/utl	pdtrcmgr	C	C	C	C	C	C	C	C
		pdodbcsetup	C	C	C	—	—	—	—	—
		pdodbcconfig	C	C	C	—	—	—	—	—

Legend:

HP: HP-UX

HP (32): 32-bit mode HP-UX

HP (64): 64-bit mode HP-UX

Sol: Solaris

Sol (64): 64-bit mode Solaris

AIX: AIX 5L

AIX (64): 64-bit mode AIX 5L

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. For Solaris the suffix is `.so`. For AIX 5L, the suffix is `.a`.

³ The file operates in 64-bit mode.

Table 6-3: Files and directories for HiRDB/Developer's Kit (UNIX client in IPF machine)

Name	Directory*	File name	Platform	
			HP-UX	Linux
Header files	<u>/HiRDB</u> /include	SQLCA.CBL	C	C
		SQLDA.CBL	C	C
		SQLIOA.CBL	C	C
		pdbtypes.h	C	C
		pdberrno.h	C	C
		pdbmisc.h	C	C
		pdbmiscm.h	C	C
		pdbsqlda.h	C	C
		pddbhash.h	C	C
		pdauxcnv.h	C	C
		SQLCAM.cb1	C	C
		SQLDAM.cb1	C	C
		SQLIOAM.cb1	C	C
		SQLIOAMTH.CBL	C	C
SQLCAMTH.CBL	C	C		

6. Client Environment Setup

Name	Directory*	File name	Platform	
			HP-UX	Linux
Shared libraries	/ <u>HiRDB</u> /client/lib	libzclt.so	C	—
		libzclt64.so	C	C
		libzcltx.so	C	—
		libzcltx64.so	NF	NF
		libzclty.so	C	—
		libzclty64.so	NF	NF
		libzcltk.so	C	—
		libzcltk64.so	C	C
		libsqlauxf.so	C	—
		libsqlauxf64.so	C	C
		libzcltxk.so	NF	—
		libzcltxk64.so	NF	NF
		libzcltyk.so	NF	—
		libzcltyk64.so	NF	NF
		libzclts.so	C	—
		libzclts64.so	C	C
		libzcltxs.so	C	—
		libzcltxs64.so	NF	NF
		libzcltys.so	C	—
		libzcltys64.so	NF	NF
JDBC drivers	/ <u>HiRDB</u> /client/lib	libjjdbc.so	C	C
		libjjdbc32.so	C	—
		pdjdbc.jar	C	C
		pdjdbc2.jar	C	C
		pdjdbc32.jar	C	—

Name	Directory *	File name	Platform	
			HP-UX	Linux
Command utilities	/ <u>HiRDB</u> /client/utl	pdcpp	C	C
		pdocc	C	C
		pdcb1	C	C
		pdccb	C	C
		pdprep	C	C
		pdtrcmgr	C	C
	pddef	C	C	
SQLJ	/ <u>HiRDB</u> /client/lib	pdsq1j.jar	C	C
		pdsq1j32.jar	C	—
		pdsq1jn.so	C	C
		pdsq1jn32.so	C	—
		libpdparse.so	C	—
	pdjava	C	C	
Message object file	/ <u>HiRDB</u> /lib	msgtxt	C	C
Syntax analysis libraries	/ <u>HiRDB</u> /lib/sjis	libasqap.so	C	C
	/ <u>HiRDB</u> /lib/chinese		—	—
	/ <u>HiRDB</u> /lib/lang-c		C	C
	/ <u>HiRDB</u> /lib/ujis		C	C

6. Client Environment Setup

Name	Directory*	File name	Platform	
			HP-UX	Linux
Sample source files	/ <u>HiRDB</u> /client/sampleap/uap	CREATE.ec	C	C
		SAMPLE1.ec	C	C
		SAMPLE2.ec	C	C
		SAMPLE3.ec	C	C
		Sample1.ecb	C	C
		Sample.mk	C	C
		inputf1	C	C
		inputf2	C	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.

* The underlined portion indicates the HiRDB installation directory.

Table 6-4: Files and directories for HiRDB/Run Time (UNIX client in IPF machine)

Name	Directory*	File name	Platform	
			HP-UX	Linux
Shared libraries	/HiRDB/client/lib	libzclt.so	C	—
		libzclt64.so	C	C
		libzcltx.so	C	—
		libzcltx64.so	NF	NF
		libzclty.so	C	—
		libzclty64.so	NF	NF
		libzcltk.so	C	—
		libzcltk64.so	C	C
		libsqlauxf.so	C	—
		libsqlauxf64.so	C	C
		libzcltxk.so	NF	—
		libzcltxk64.so	NF	NF
		libzcltyk.so	NF	—
		libzcltyk64.so	NF	NF
		libzclts.so	C	—
		libzclts64.so	C	C
		libzcltxs.so	C	—
		libzcltxs64.so	NF	NF
		libzcltys.so	C	—
		libzcltys64.so	NF	NF

6. Client Environment Setup

Name	Directory*	File name	Platform	
			HP-UX	Linux
JDBC drivers	/ <u>HiRDB</u> /client/lib	libjjdbc.so	C	C
		libjjdbc32.so	C	—
		pdjdbc.jar	C	C
		pdjdbc2.jar	C	C
		pdjdbc32.jar	C	—
SQLJ runtime files	/ <u>HiRDB</u> /client/lib	pdruntime.jar	C	C
		pdruntime32.jar	C	—
		pdnativert.jar	C	C
		pdnativert32.jar	C	—
		pdsqjln.so	C	C
Command utilities	/ <u>HiRDB</u> /client/utl	pdtrcmgr	C	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.

* The underlined portion indicates the HiRDB installation directory.

Table 6-5: Files and directories for HiRDB/Developer's Kit (Linux (EM64T))

Name	Directory*	File name
Header files	/ <i>HiRDB</i> /include	SQLCA.CBL
		SQLDA.CBL
		SQLIOA.CBL
		pdbtypes.h
		pdberrno.h
		pdbmisc.h
		pdbmiscm.h
		pdbsqlda.h
		pddbhash.h
		pdauxcnv.h
		SQLCAM.cbl
		SQLDAM.cbl
		SQLIOAM.cbl
		SQLIOAMTH.CBL
SQLCAMTH.CBL		

6. Client Environment Setup

Name	Directory*	File name
Shared libraries	/ <u>HiRDB</u> /client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzcltk.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so
		libzcltxk.so
		libzcltyk.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
JDBC driver	/ <u>HiRDB</u> /client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
Command utilities	/ <u>HiRDB</u> /client/utl	pdcpp
		pdocc
		pdcb1
		pdocc
		pdprep
		pdtrcmgr
		/ <u>HiRDB</u> /bin

Name	Directory *	File name
SQLJ	<u>/HiRDB</u> /client/lib	pdsqj.jar
		libpdparse.so
		libpdsqjln.so
		pdruntime.jar
	<u>/HiRDB</u> /client/utl	pdjava
Message object file	<u>/HiRDB</u> /lib	msgtxt
Parsing libraries	/HiRDB/lib/sjis	libasqap.so
	<u>/HiRDB</u> /lib/chinese	
	<u>/HiRDB</u> /lib/lang-c	
	<u>/HiRDB</u> /lib/ujis	
	<u>/HiRDB</u> /lib/utf8	
Sample source files	<u>/HiRDB</u> /client/sampleap/uap	CREATE.ec
		SAMPLE1.ec
		SAMPLE2.ec
		SAMPLE3.ec
		Sample1.ecb
		Sample.mk
		inputf1
		inputf2

* The underlined portion indicates the HiRDB installation directory.

Table 6-6: Files and directories for HiRDB/Run Time (Linux (EM64T))

Name	Directory*	File name
Shared libraries	/ <u>HiRDB</u> /client/lib	libzclt.so
		libzclt64.so
		libzcltx.so
		libzclty.so
		libzcltk.so
		libzcltk64.so
		libsqlauxf.so
		libsqlauxf64.so
		libzcltxk.so
		libzcltyk.so
		libzclts.so
		libzcltxs.so
		libzcltys.so
JDBC driver	/ <u>HiRDB</u> /client/utl	libjjdbc.so
		pdjdbc.jar
		pdjdbc2.jar
SQLJ runtime	/ <u>HiRDB</u> /client/lib	libpdsqln.so
Command utility	/ <u>HiRDB</u> /client/utl	pdtrcmgr

* The underlined portion indicates the HiRDB installation directory.

■ **Archived files and shared library files used for each purpose**

Table 6-7 shows the archived files that are used for each purpose. Table 6-8 shows the shared library files that are used for each purpose.

Table 6-7: Archived files used for each purpose (UNIX client)

Purpose	File used
Normal UAP	libclt.a

Purpose		File used
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

¹ The connection type can be switched between static connection and dynamic connection by the switch registered to TM.

Table 6-8: Shared library files used for each purpose (UNIX client)

Purpose			File used
Normal UAP			libzclt.sl
XA interface connection	Dynamic connection	Single thread	libzcltx.sl libzcltxs.sl (for OTS)
		Multiple threads	libzcltxk.sl
	Static or dynamic connection ¹	Single thread	libzclty.sl libzcltys.sl (for OTS or TUXEDO)
		Multiple threads	libzcltyk.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. For Solaris, the suffix is .so; for AIX 5L, the suffix is .a.

¹ 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

Table 6-9 lists the library files used by each transaction manager.

Table 6-9: 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
TPBroker	libzcltxk.sl	N
	libzcltyk.sl	N
TUXEDO	libzcltys.sl	Y
WebLogic Server	libzcltyk.sl	N
TP1/EE	libzcltyk.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. For Solaris, the suffix is `.so`; for AIX 5L, the suffix is `.a`.

6.4.2 Directories and files for Windows clients

Tables 6-10 through 6-16 list the files and directories that are created automatically during HiRDB client installation.

Table 6-10: Files and directories for HiRDB/Developer's Kit (Windows client)

Name	Directory	File name
Header files	xxx\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		SQLCA.CBL
		SQLIOA.CBL
		PDBMISCM.H
		SQLDA.CBL
		PDBSQLDA.H
		SQLLOAD.CBL
		SQLCAD.CBL
		PDDHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
SQLCAMTH.CBL		
Message object file	xxx\LIB	msgtxt
Linkage libraries	xxx\LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTX32.LIB
		PDCLTXM.LIB
		PDSQLAUXF.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB
JBuilder files	xxx\JBA	PDJBA35.JAR*
		PDJBA4.JAR*
		PDJBA5.JAR*

6. Client Environment Setup

Name	Directory	File name
Command utilities	xxx\UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.EXE
		PDPREP7.EXE
		PDTRCMGR.EXE
		PDCLTADM.EXE
DLL files	xxx\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTX32.DLL
		PDCLTXM.DLL
		PDSQLAUXF.DLL
		PDOLEDB.DLL*
		PDCLTXS.DLL
		PDCLTXM5.DLL
JDBC drivers	xxx\UTL	JJDBC.DLL*
		PDJDBC.JAR*
		PDJDBC2.JAR
SQLJ	xxx\UTL	PDSQLJ.JAR*
		PDPARSE.DLL*
		PDJAVA.EXE*
		PDSQLJN.DLL*
HiRDB.NET data providers	xxx\UTL	PDDNDP.DLL*
		PDDBDPCORE.DLL*

Name	Directory	File name
Interface definition file	xxx\LIB	HIRDB.PKG
Sample files	xxx\SAMPLEAP	CREATE.EC
		SAMPLE1.EC
		SAMPLE2.EC
		SAMPLE3.EC
		SAMPLE1.ECB
		INPUTF1
		INPUTF2
README file	xxx	README.TXT
Environment definition file	\WINDOWS	HIRDB.INI

* This file is created when **Custom** installation is selected.

Notes

1. xxx 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 6-11: Files and directories for HiRDB/Run Time (Windows client)

Name	Directory	File name
Command utilities	xxx\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTP32.DLL
		PDCLTX32.DLL
		PDTRCMGR.EXE
		PDSQLAUXF.DLL
		PDCLTXM.DLL
		PDOLEDB.DLL*

6. Client Environment Setup

Name	Directory	File name
		PDCLTADM.EXE
		PDCLTXS.DLL
		PDCLTXM5.DLL
		JJDBC.DLL*
		PDJDBC.JAR*
		PDJDBC2.JAR*
		PDRUNTIME.JAR*
		PDDNDP.DLL*
		PDDBDPCORE.DLL*
		HIRDB.PKG
README file	xxxx	README.TXT
Environment definition file	\WINDOWS	HIRDB.INI

* This file is created when **Custom** installation is selected.

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 6-12: Files and directories for HiRDB/Developer's Kit (Windows client in IPF machine)

Name	Directory	File name
Header files	xxx\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		PDBMISC.H
		SQLDA.CBL
		PDBSQLDA.H
		SQLIOA.CBL
		SQLCA.CBL
		SQLLOAD.CBL
		SQLCAD.CBL
		PDDHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
SQLCAMTH.CBL		
Linkage libraries	xxx\LIB	PDCLTM64.LIB
		PDCLTX64.LIB
		PDCLTXM64.LIB
		PDSQLAUXF64.LIB
		PDCLTXS64.LIB

6. Client Environment Setup

Name	Directory	File name
Command utilities	xxxx\UTL	PDCPP.EXE
		PDOCC.EXE
		PDCBL.EXE
		PDOCB.EXE
		PDPREP.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	PDSQLJN.DLL
		PDSQLJ.JAR
		PDPARSE.DLL
Sample files	xxxx\SAMPLEAP	CREATE.EC
		SAMPLE1.EC
		SAMPLE2.EC
		SAMPLE3.EC
		SAMPLE1.ECB
		INPUTF1
		INPUTF2
README file	xxxx	README.TXT

Name	Directory	File name
Environment definition file	\Windows	HIRDB.INI

Notes

1. *xxx* 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 6-13: Files and directories for HiRDB/Run Time (Windows client in IPF machine)

Name	Directory	File name
Command utilities	xxx\UTL	PDCLTM64.DLL
		PDCLTX64.DLL
		PDCLTXM64.DLL
		PDSQLAUXF64.DLL
		PDCLTXS64.DLL
		PDTRCMGR.EXE
		PDCLTADM.EXE
		PDJDBC.JAR
		PDJDBC2.JAR
		JJDBC.DLL
		PDRUNTIME.JAR
		PDNATIVEVERT.JAR
		PDSQLJN.DLL
README file	xxx	README.TXT
Environment definition file	\Windows	HIRDB.INI

Notes

1. *xxx* 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 6-14: Files and directories for HiRDB/Developer's Kit (EM64T machine Windows client)

Name	Directory	File name
Header files	xxx\INCLUDE	PDBTYPES.H
		PDBERRNO.H
		PDBMISC.H
		PDBMISCM.H
		SQLDA.CBL
		PDBSQLDA.H
		SQLIOA.CBL
		SQLCA.CBL
		SQLLOAD.CBL
		SQLCAD.CBL
		PDBHASH.H
		PDAUXCNV.H
		SQLIOAMTH.CBL
SQLCAMTH.CBL		
Linkage libraries	xxx\LIB	CLTDLL.LIB
		PDCLTM32.LIB
		PDCLTM50.LIB
		PDCLTM64.LIB
		PDCLTX32.LIB
		PDCLTXM.LIB
		PDSQLAUXF.LIB
		PDSQLAUXF64.LIB
		PDCLTXS.LIB
		PDCLTXM5.LIB

Name	Directory	File name
Jbuilder addins	xxxx\JBA	PDJBA35 . JAR
		PDJBA4 . JAR
		PDJBA5 . JAR
Command utilities	xxxx\UTL	PDCPP . EXE
		PDOCC . EXE
		PDCBL . EXE
		PDOCB . EXE
		PDPREP . EXE
		PDJAVA . EXE
		PDTRCMGR . EXE
		PDCLTADM . EXE
DLL files	xxxx\UTL	CLTDLL . DLL
		PDCLTM32 . DLL
		PDCLTM50 . DLL
		PDCLTM64 . 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
		PDSQLJN . DLL

6. Client Environment Setup

Name	Directory	File name
HiRDB data providers	xxxx\UTL	PDDNDP.DLL
		PDDBDPCORE.DLL
ODBC 3.0 drivers	\Windows\system32	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.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

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.

Table 6-15: Files and directories for HiRDB/Run Time (EM64T machine Windows client)

Name	Directory	File name
Command utilities	xxxx\UTL	PDTRCMGR.EXE
		PDCLTADM.EXE

Name	Directory	File name
DLL files	xxx\UTL	CLTDLL.DLL
		PDCLTM32.DLL
		PDCLTM50.DLL
		PDCLTM64.DLL
		PDCLTX32.DLL
		PDCLTXM.DLL
		PDOLEDB.DLL
		PDSQLAUXF.DLL
		PDSQLAUXF64.DLL
		PDPARSE.DLL
		PDCLTXS.DLL
JDBC driver	xxx\UTL	JJDBC.DLL
		PDJDBC.JAR
		PDJDBC2.JAR
SQLJ runtime	xxx\UTL	PDSQLJN.DLL
HiRDB data providers	xxx\UTL	PDDNDP.DLL
		PDDBDPCORE.DLL
ODBC 3.0 drivers	\Windows\system32	pdodbcdrv3.dll
		pdodbstp3.dll
		pdclto32.dll
README files	xxx	README.TXT
Environment definition file	\Windows	HIRDB.INI

Notes:

1. xxx 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.

Table 6-16: 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

Table 6-17 lists the linkage library files that are used according to their purpose.

Table 6-17: 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)
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

Table 6-18 lists the library files used by each transaction manager.

Table 6-18: 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	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

Table 6-19 lists the libraries and compilers.

Table 6-19: 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		

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. If DNS is used, the `hosts` file does not need to be set.

- IP address
- Host name

(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 8 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 ]
```

6. Client Environment Setup

```

$
PDFESGRP=FES-group [, switchover-FES-group [, switchover-FES-group] . . . ]
$ PDCLTRCVPORT=client-recv-port-number
$ PDCLTRCVADDR={ client-IP-address | client-host-name }
$ PDTMID=OLTP-identifier
$ PDXAMODE={ 0 | 1 }
$
PDTXACANUM=maximum-number-of-concurrent-transaction-executions-per-process
$ PDXARCVWTIME=transaction-recovery-wait-time
$ PDXATRCFILEMODE={ LUMP | SEPARATE }
$
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 }
$ PDLANG={ UTF-8 | SJIS | CHINESE | 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
$ PDRINTERVAL=CONNECT-retry-interval-with-automatic-reconnect-facility
$ PDUAPENVFILE=UAP-environment-definition-file-name
$ PDDBBUFLRU={ YES | NO }
$ PDHATRQUEUEING=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 }

```

```

$ PSENDMEMSIZE=data-send-memory-size-in-client
$ PDRECVMEMSIZE=data-recv-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-connection
$
$ PDCLTPATH=trace-file-storage-directory
$ PDSQLTRACE=SQL-trace-file-size
$ PDUAPERLOG=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
$ 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 }

```

6. Client Environment Setup

```

$ 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
$ PDCMMTBFDL={YES|NO}
$ PDPRPCRCLS={YES|NO}
$ PDAUTOCONNECT={ON|OFF}
$ PDDDLDEAPRP={YES|NO}
$ PDCURSORLVL={0|1|2}
$ PDDELRVWDFILE=SQL-reserved-word-deletion-file-name
$ PDHJHASHINGMODE={TYPE1|TYPE2}
$ PDBLKF=block-transfer-row-count
$ PDBINARYBLKF={YES|NO}
$ PDBLKBUFFSIZE=communication-buffer-size
$ PDNODELAYACK={YES|NO}
$ PDBBACCS=generation-number-of-RDAREA-to-be-accessed
$ PDBBORGUAP={YES|NO}
$ PDSPACEVLVL={0|1|3}
$ PDCLTRDNODE=XDM/RD-E2-database-identifier
$ PDTP1SERVICE={YES|NO}
$ PDCNSTRNTNAME={LEADING|TRAILING}
$ PDBESCONHOLD={YES|NO}
$ PDBESCONHTI=BES-connection-holding-period
$ PDRDABLKF=batch-retrieval-row-count
$ 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}
$ PDPLGPFSSZ=initial-size-of-delayed-batch-creation-index-information-file
$
PDPLGPFSSZEXP=extension-value-of-delayed-batch-creation-index-information-file

$ export PDHOST PDNAMEPORT PDFESHOST PDSERVICEGRP PDSRVTYPE
    PDSERVICEPORT PDFESGRP PDCLTRCVPORT PDCLTRCVADDR PDTMID
PDXAMODE
    PDXACANUM PDXARCVWTIME PDXATRCFILEMODE PDUSER
    PDCLTAPNAME PDCLTLANG PDLANG PDBBLOG PDEXWARN

```

```

PDSUBSTRLEN
PDCLTCNVMODE PDCLTGAIJIDLL PDCLTGAIJUFUNC PDCLTGRP
PDTCPCONOPT
PDAUTORECONNECT PDRCCOUNT PDRINTERVAL PDUAPENVFILE
PDDBBUFLRU PDHATRQUEUEING PDASTHOST PDASTPORT
PDSYSTEMID PDASTUSER
PDCMDWAITTIME PDCMDTRACE PDIPC PSENDMEMSIZE
PDRECVMEMSIZE PDCWAITTIME PSDWAITTIME PSDWATCHTIME
PDCWAITTIMEWRNPNT PDKALVL PDKATIME PDTIMEDOUTRETRY
PDNBLOCKWAITTIME PDCONNECTWAITTIME PDCLTPATH PDSQLTRACE
PDUAPERLOG PDERRSKIPCODE PDPRMTRC PDPRMTRCSIZE
PDTRCMODE
PDUAPREPLVL PDREPPATH PDTRCPATH PDSQLTRCOPENMODE
PDSQLTEXTSIZE
PDUAPEXERLOGUSE PSUAPEXERLOGPRMSZ PDVWOPTMODE
PDTAAPINFPATH
PDTAAPINFMODE PDTAAPINFMODE PDSTJTRNOUT PDLOCKLIMIT
PDDLKPRIO
PDLOCKSKIP PDFORUPDATEEXLOCK PDISLLVL PDSQLOPTLVL
PDADDITIONALOPTLVL PDHASHTBLSIZE PDDFLNVAL PDAGGR
PDCMMTBFDL PDPRPCRCLS PDAUTOCONNECT PDDDLDEAPRP
PDCURSORLVL
PDDELRVWDFILE PDHJHASHINGMODE PDBLKF PDBINARYBLKF
PDBLKBUFSIZE
PDNODELAYACK PDBBACCS PDBBORGUAP PDSPACELVL PDCLTRDNODE
PDTP1SERVICE PDCNSTRNTNAME PDBESCONHOLD PDBESCONHTI
PDRDABLKF
PDODBSTATCAHE PDODBESCAPE PDGDATAOPT PDODBLOCATOR
PDODBSPLITSIZE
PDODBCWRNSKIP PDJETCOMPATIBLE PDPLGIXMK PDPLGPFSSZ
PDPLGPFSSZEXP

```

(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-recv-port-number
% setenv PDCLTRCVADDR {client-IP-address
                       | client-host-name}

% setenv PDTMID OLTP-identifier
% setenv PDXAMODE {0|1}
% setenv PDTXACANUM

maximum-concurrent-transaction-execution-count-per-process
% setenv PDXARCVWTIME transaction-recovery-wait-time
% setenv PDXATRCFILEMODE {LUMP|SEPARATE}
% 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}
% setenv PDLANG {UTF-8|SJIS|CHINESE|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 PDHATRQUEUEING 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 PSENDMEMSIZE data-send-memory-size-in-client
% setenv PDRECVMEMSIZE data-recv-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 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 PDVWOPTMODE {0|1|2}
% setenv PDTAAPINFPATH access-path-information-file-output-directory-name
% setenv PDTAAPINFMODE {0|1}
% setenv PDTAAPINFPSIZE 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 PDCMMTBFDL {YES|NO}
% setenv PDPRPCRCLS {YES|NO}
% setenv PDAUTOCONNECT {ON|OFF}
% setenv PDDDLDEAPRP {YES|NO}
% setenv PDCURSORLVL {0|1|2}
% setenv PDDELRSVWDFILE SQL-reserved-word-deletion-file-name
% setenv PDHJHASHINGMODE {TYPE1|TYPE2}
% setenv PDBLKF block-transfer-row-count
% setenv PDBINARYBLKF {YES|NO}
% setenv PDBLKBUFFSIZE communication-buffer-size
% setenv PDNODELAYACK {YES|NO}
% 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 PDRDABLKF batch-retrieval-row-count
% 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*.
- If different directories are used for storing shared libraries for the client during UAP creation and UAP execution, specify the `-w1,+s` option in the `cc` or `ccb1` command. During UAP execution, specify the name of the directory containing the shared libraries in the `SHLIB_PATH` environment variable. For details about the `cc` and `ccb1` commands, see *8.3 Compiling and linking*.
- 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

In the Windows environment, the directory is set automatically to 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 8 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-recv-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 }
PDLANG={ UTF-8 | SJIS | CHINESE | 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 }
PDHATRQUEUEING=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

```

```

PDI PC={MEMORY | DEFAULT }
PDS ENDMEMSIZE=data-send-memory-size-in-client
PDR ECVMEMSIZE=data-recv-memory-size-in-client
PDC WAITTIME=maximum-client-wait-time
PDS WAITTIME=maximum-server-wait-time-during-transaction-processing
PDS WATCHTIME=maximum-server-wait-time-outside-transaction-processing
PDC WAITTIMEWRNPNT=output-timing-for-SQL-runtime-warning
PDK ALVL={ 0 | 1 | 2 }
PDK ATIME=packet-send-interval
PDT IMEDOUTRETRY=retry-count

PDN BLOCKWAITTIME=connection-establishment-monitoring-time-in-nonblock-mode

PDC ONNECTWAITTIME=maximum-wait-time-in-HiRDB-client-during-server-connection
ion
  PDC LTPATH=trace-file-storage-directory
  PDS QLTRACE=SQL-trace-file-size
  PDU APERLOG=error-log-file-size
  PDE RRSKIPCODE=SQLCODE [ , SQLCODE ] . . .
  PDP RMRTRC={ YES | NO | IN | OUT | INOUT }
  PDP RMRTRCSIZE=maximum-data-length-of-parameter-information-output-to-SQL-trace
  PDT RCMODE={ ERR | NONE }
  PDU APREPLVL={ [s] [u] [p] [r] | a }
  PDE RPPATH=storage-directory-for-UAP-statistical-report-files
  PDT RCPATH=storage-directory-for-dynamic-SQL-trace-files
  PDS QLTRCOPENMODE={ CNCT | SQL }
  PDS QLTEXTSIZE=SQL-statement-size
  PDS QLEXECTIME={ YES | NO }
  PDR CTRACE=reconnect-trace-file-size

PDT RNLNPATH=storage-directory-for-files-to-which-WRITE-LINE-statement-value-expression-values-are-output

PDT RNLNFILSZ=maximum-size-of-output-files-for-WRITE-LINE-statement-value-expression-values
  PDT RNLNCOMSZ=total-size-of-WRITE-LINE-statement-value-expression-values
  PDU APEXERLOGUSE={ YES | NO }
  PDU APEXERLOGPRMSZ=maximum-data-length-of-parameter-information
  PDV WOPTMODE={ 0 | 1 | 2 }
  PDT AAPINFMPATH=access-path-information-file-output-directory-name
  PDT AAPINFMODE={ 0 | 1 }
  PDT AAPINFMSZ=access-path-information-file-size
  PDS TJTRNOUT={ YES | NO }
  PDL OCKLIMIT=maximum-locked-resource-request-count-per-user
  PDD LKPRIO={ 96 | 64 | 32 }

```

```

PDFORUPDATEEXLOCK={ YES | NO }
PDLOCKSKIP={ 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
PDCMMTBFDL={ YES | NO }
PDPRPCRCLS={ YES | NO }
PDAUTOCONNECT={ ON | OFF }
PDDDLDEAPRP={ YES | NO }
PDCURSRLVL={ 0 | 1 | 2 }
PDDELRSVWDFILE=SQL-reserved-word-deletion-file-name
PDHJHASHINGMODE={ TYPE1 | TYPE2 }
PDBLKF=block-transfer-row-count
PDBINARYBLKF={ YES | NO }
PDBLKBUFFSIZE=communication-buffer-size
PDNODELAYACK={ YES | NO }
PDDBACCS=generation-number-of-RDAREA-to-be-accessed
PDDBORGUAP={ YES | NO }
PDSPACEVLVL={ 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 }
PDPLGPFSSZ=initial-size-of-delayed-batch-creation-index-information-file
PDPLGPFSSZEXP=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

Table 6-20 shows the OpenTP1 definitions in which the environment variables are specified. Environment variables other than those shown in Table 6-20 can be specified as needed.

The `putenv` format is used to specify environment variables.

Table 6-20: OpenTP1 definitions in which the environment variables are specified

Environment variable	System environment definition	Transaction service definition	User service default definition	User service definition
H1RDB_PDHOST ⁹	M ¹	N	N	N
H1RDB_PDNAMEPORT ⁹	M ²	N	N	N
H1RDB_PDTMID ⁹	O ^{3,4}	N	N	N
H1RDB_PDXAMODE ⁹	O ⁵	N	N	N
PDHOST	N	M ^{1,6}	M ^{1,6}	O ^{1,6,7}

6. Client Environment Setup

Environment variable	System environment definition	Transaction service definition	User service default definition	User service definition
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
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

Environment variable	System environment definition	Transaction service definition	User service default definition	User service definition
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.

Table 6-21 shows the TP1/LiNK definitions in which the environment variables are specified.

Table 6-21: 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
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

Environment variable	Transaction Service Environment Variables field	User Server Environment Variables field	
		Global field	Local field
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 as the client

If a UAP under TPBroker is used as the client, the client environment definitions must be specified in the TPBroker system definitions. For details about the TPBroker 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 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 recovery process for transaction failures:**

Specify the client environment definitions in the transaction definition. In this case, use the `tsdefvalue` command of TPBroker 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.

Table 6-22 shows the TPBroker definitions in which the environment variables are specified.

6. Client Environment Setup

Table 6-22: TPBroker 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
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

Environment variable	Transaction completion process	Transaction recovery process	Each UAP
PDCWAITTIMEWRNPNT	O	O	O
PDTCPCONOPT	O	O	O
PDAUTORECONNECT	N	N	N
PDRCCOUNT	N	N	N
PDRCINTERVAL	N	N	N
PKALVL	N	N	N
PKATIME	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 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).

Table 6-23 shows which environment variables can be specified.

Table 6-23: Environment variable specification status (for a UAP under TUXEDO)

Environment variable	Specification status
<code>HiRDB_PDHOST</code>	N
<code>HiRDB_PDNAMEPORT</code>	N
<code>HiRDB_PDTMID</code>	N
<code>HiRDB_PDXAMODE</code>	N
<code>PDHOST</code>	M ¹

Environment variable	Specification status
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
PKALVL	N
PKATIME	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

6. Client Environment Setup

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.

Table 6-24 shows which environment variables can be specified.

Table 6-24: Environment variable specification status (for a UAP under WebLogic Server)

Environment variable	Specification status
<code>HIRDB_PDHOST</code>	N
<code>HIRDB_PDNAMEPORT</code>	N
<code>HIRDB_PDTMID</code>	N
<code>HIRDB_PDXAMODE</code>	N
<code>PDHOST</code>	M ³
<code>PDNAMEPORT</code>	M
<code>PDTMID</code> ⁴	O ¹
<code>PDXAMODE</code> ⁴	M
<code>PDUSER</code>	M
<code>PDSWAITTIME</code>	M
<code>PDCLTAPNAME</code>	O ²
<code>PDSWATCHTIME</code>	N
<code>PDAUTORECONNECT</code>	N
<code>PDRCCOUNT</code>	N
<code>PDRCINTERVAL</code>	N
<code>PKALVL</code>	N
<code>PKATIME</code>	N
<code>PDRCTRACE</code>	N

Environment variable	Specification status
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 process 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 UNIX version)

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*.

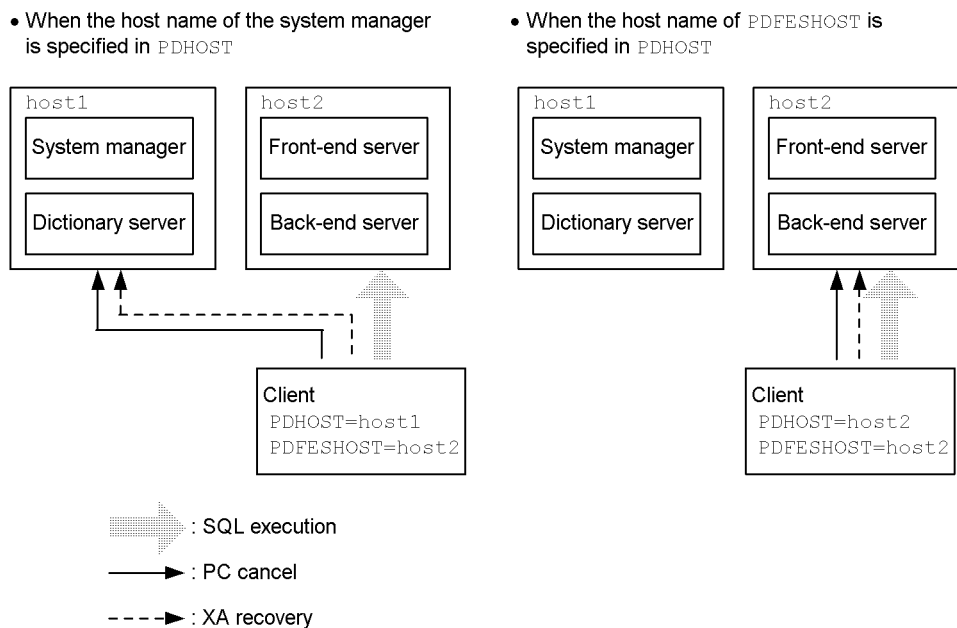
Be sure to specify `PDXAMODE`. If the value specified for the OpenTP1 system in which TP1/EE is executed and the value specified for `PDXAMODE` are different, specify `PDXAMODE` in the user service definitions of the OpenTP1 system.

(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.

Figure 6-1 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	
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, or TP1/LiNK) operating with multiple processes	All processes operating as a UAP specify the same PDFESHOST. ¹			Y
		Each process operating as a UAP specifies a separate PDFESHOST. ¹	Client environment definition of a process operating as UAP for which PDFESHOST is specified ²	The same PDFESHOST is specified for all connection destinations of a process operating as a UAP for which PDFESHOST is specified.	Y
				A different PDFESHOST is specified for each connection destination of a process operating as 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
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
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
 - 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

Table 6-25 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 6-25: Client environment definitions

No.	Environment variable	Function	Category
1	<code>PDHOST</code>	Specifies the host name of the HiRDB system to be connected.	System configuration ³
2	<code>PDNAMEPORT</code>	Specifies the port number of the HiRDB system.	
3	<code>PDFESHOST</code>	Specifies the host name of the front-end server.	
4	<code>PDSERVICEGRP</code>	Specifies the server name of the single server or front-end server.	
5	<code>PDSRVTYPE</code>	Specifies the HiRDB server type.	

6. Client Environment Setup

No.	Environment variable	Function	Category	
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
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	HiRDB_PDHOST	Specifies the host name of the HiRDB server to be connected.		
16	HiRDB_PDNAMEPORT	Specifies the port number of the HiRDB server.		
17	HiRDB_PDTMID	Specifies a unique identifier for each OLTP when multiple OLTPs access one HiRDB server.		
18	HiRDB_PDXAMODE	Specifies whether the transaction transfer function is to be used when the HiRDB client is linked with an OLTP system.		
19	PDUSER	Specifies the authorization identifier and password. This environment variable can be omitted in the UNIX environment.	User execution environment	

No.	Environment variable	Function	Category
20	PDCLTAPNAME	Specifies UAP identification information (UAP identifier) for accessing the HiRDB server.	
21	PDCLTLANG	Specifies the character code classification used in the descriptions of UAPs to be processed by the preprocessor.	
22	PDLANG	Specifies that the character code used when the UAP is executed is either Unicode or EUC Chinese kanji code. In the Linux version, specifies that SJIS is used as the character code. This environment variable is not valid in a Windows environment.	
23	PDDBLOG	Specifies whether the database update log is to be collected when a UAP is executed.	
24	PDEXWARN	Specifies whether return codes with warnings are to be accepted from the server.	
25	PDSUBSTRLEN	Specifies the maximum number of bytes used to represent one character.	
26	PDCLTCNVMODE	Specifies whether character codes are to be converted if the HiRDB server and the HiRDB client use different character code classifications.	
27	PDCLTGAIJIDLL	Specifies the name of the user-defined external character conversion DLL file.	
28	PDCLTGAIJIFUNC	Specifies the name of the user-defined external character conversion function.	
29	PDCLTGRP	Specifies a client group name when the connection frame guarantee facility for client groups is used.	
30	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.	
31	PDAUTORECONNECT	Specifies whether the automatic reconnect facility is to be used.	
32	PDRCCOUNT	Specifies the number of times the <code>CONNECT</code> statement is to be retried by the automatic reconnect facility.	

6. Client Environment Setup

No.	Environment variable	Function	Category
33	PDRINTERVAL	Specifies the retry interval for CONNECT statement execution by the automatic reconnect facility.	
34	PDUAPENVFILE	Specifies the UAP environment definition file that defines the execution environment if the UAP is to be executed in a separate environment.	
35	PDDBBUFLRU	Specifies whether the LRU method is used for caching in global buffer pages accessed by the UAP.	
36	PDHATRQUEUEING	Specifies that the client is not using the transaction queuing facility.	
37	PDASTHOST	Specifies the host name of HiRDB Control Manager - Agent to be connected when the UAP is executed.	Command execution from a UAP
38	PDASTPORT	Specifies the port number of Control Manager - Agent to be connected when the UAP is executed.	
39	PDSYSTEMID	Specifies the HiRDB identifier of the HiRDB server managed by HiRDB Control Manager - Agent to be connected when the UAP is executed.	
40	PDASTUSER	Specifies the user name and password for the OS that will run commands.	
41	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.	
42	PDCMDTRACE	Specifies the size of the command trace file when a file is output during UAP execution.	
43	PDIPC	Specifies the communication method between processes.	
44	PDSENDMEMSIZE	Specifies the data storage area size when the client sends data to the server while the inter-process memory communication facility is used.	

No.	Environment variable	Function	Category
45	PDRECVMEMSIZE	Specifies the data storage area size when the client receives data from the server while the inter-process memory communication facility is used.	
46	PDCWAITTIME	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
47	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.	
48	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.	
49	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.	
50	PDKALVL	Specifies whether the facility that sends packets regularly from the HiRDB client to the HiRDB server is to be used.	
51	PDKATIME	Specifies the interval for sending packets regularly from the HiRDB client to the HiRDB server.	
52	PDTIMEOUTRETRY	Specifies the number of times the <code>connect ()</code> system call is to be retried if an error occurs in the <code>connect ()</code> system call executed when the HiRDB client connects with the HiRDB server.	
53	PDNBLOCKWAITTIME	Specifies the connection establishment monitoring time in nonblock mode when completion of the connection between the HiRDB server and client is monitored.	

6. Client Environment Setup

No.	Environment variable	Function	Category
54	PDCONNECTWAITTIME	Specifies the maximum wait time that the HiRDB client waits from a response from the HiRDB server during connection with the HiRDB server.	
55	PDCLTPATH	Specifies the storage directory for the SQL trace file and error log file created by the HiRDB client.	Trouble-shooting
56	PDSQLTRACE	Specifies the size of the SQL trace file (in byte units) into which the SQL trace of the UAP is to be output.	
57	PDUAPERLOG	Specifies the size of the error log file (in byte units) into which the error log of the UAP is to be output.	
58	PDERRSKIPCODE	Specifies that a specific error log is not to be output.	
59	PDPRMTRC	Specifies whether parameter information and retrieval data is to be output to the SQL trace information.	
60	PDPRMTRCSIZE	Specifies the maximum length of the parameter information and retrieval data to be output to the SQL trace information.	
61	PDTRCMODE	Specifies whether troubleshooting information other than SQL trace information is to be output.	
62	PDUAPREPLVL	Specifies output information for UAP statistical reports.	
63	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.	
64	PDTRCPATH	Specifies the storage directory for dynamic SQL trace files.	
65	PDSQLTRCOPENMODE	Specifies the open mode for the SQL trace file when PDREPPATH is specified.	
66	PDSQLTEXTSIZE	Specifies the size of the SQL statement to be output to the SQL trace information.	
67	PDSQLEXECTIME	Specifies whether the SQL runtime is to be output to the SQL trace information.	

No.	Environment variable	Function	Category
68	PDRCTRACE	Specifies the size of the file that outputs the UAP reconnect trace.	
69	PDWRTLNPATH	Specifies the storage directory for files to which value expression values of WRITE LINE statements are to be output.	
70	PDWRTLNFILSZ	Specifies the maximum size of the files to which value expression values of WRITE LINE statements are to be output.	
71	PDWRTLNCOMSZ	Specifies the total size of the value expression values in WRITE LINE statements.	
72	PDUAPEXERLOGUSE	Specifies whether the facility for output of extended SQL error information is to be used.	
73	PDUAPEXERLOGPRMSZ	Specifies the maximum data length in the parameter information to be output to the error log file and the SQL error report when the facility for output of extended SQL error information is used.	
74	PDVWOPTMODE	Specifies whether or not the access path information file is to be obtained.	Access path display utility
75	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
76	PDTAAPINFMODE	Specifies the file name format of the access path information file when it is output to the HiRDB client side.	
77	PDTAAPINFSIZE	Specifies the file size of the access path information file when it is output to the HiRDB client side.	
78	PDSTJTRNOUT	Specifies whether UAP statistical information is to be output to the client side.	Output unit of UAP statistical information
79	PDLOCKLIMIT	Specifies the maximum number of lock requests that a UAP can issue to one server.	Lock control
80	PDDLKPLIO	Specifies the deadlock priority value of a UAP.	
81	PDLOCKSKIP	Specifies whether search using condition evaluation with no lock is to be performed.	

6. Client Environment Setup

No.	Environment variable	Function	Category
82	PDFORUPDATEEXLOCK	Specifies whether <code>WITH EXCLUSIVE LOCK</code> is to be applied to the lock option of the SQL statement in which the <code>FOR UPDATE</code> clause was specified (or assumed).	
83	PDISLLVL	Specifies the data guarantee level for SQL statements.	SQL-related
84	PDSQLOPTLVL	Specifies the optimization method (SQL optimization option) for determining the most efficient access path in consideration of the database status.	
85	PDADDITIONALOPTLVL	Specifies the optimization method (SQL extension optimizing option) for determining the most efficient access path in consideration of the database status.	
86	PDHASHTBLSIZE	Specifies the hash table size to be used when hash join or subquery hash execution is applied in SQL optimization.	
87	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.	
88	PDAGGR	Specifies the maximum number of groups allowed in each server so that the memory size used in <code>GROUP BY</code> processing can be determined.	
89	PDCMMTBFDDL	Specifies whether a transaction that has that executed a data manipulation SQL statement must execute <code>commit</code> processing automatically before it executes a definition SQL statement.	
90	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 <code>PREPARE</code> statement.	
91	PDAUTOCONNECT	Specifies whether autoconnection is to be executed if an SQL statement is executed while the client is not connected to HiRDB.	
92	PDDDLDEAPRP	Specifies whether the definition information of a table being used by a closed holdable cursor can be changed by another UAP between transactions.	

No.	Environment variable	Function	Category
93	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.	
94	PDDELRVWDFILE	Specifies the name of the SQL reserved word deletion file when the SQL reserved word deletion file is used.	
95	PDHJHASHINGMODE	Specifies the hashing method when Apply hash join, subquery hash execution is selected as the SQL extension optimizing option.	
96	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
97	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.	
98	PDBLKBUFFSIZE	Specifies the size of the server-client communication buffer used by the block transfer facility.	
99	PDNODELAYACK	Specifies whether immediate acknowledgement is to be used. This environment variable is limited to the AIX 5L version.	Use of immediate acknowledgment for HiRDB communication
100	PDBBACCS	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
101	PDBBORGUAP	Specifies that the UAP is to be executed for the original RDAREA for online reorganization hold.	Updatable online reorganization
102	PDSPACEVL	Specifies the space conversion level for data storage, comparison, and retrieval.	Space conversion for data
103	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

6. Client Environment Setup

No.	Environment variable	Function	Category
104	PDTP1SERVICE	Specifies whether OpenTP1 service names are to be reported to XDM/RD E2 when the XDM/RD E2 connection facility is used.	
105	PDRDCLTCODE	Specifies the character code classification that the client uses when the XDM/RD E2 connection facility is used.	
106	PDCNSTRNTNAME	Specifies the position of the constraint name definition when a referential or check constraint is defined.	Referential or check constraint
107	PDBESCONHOLD	Specifies whether the BES connection holding facility is to be used.	BES connection holding facility
108	PDBESCONHTI	Specifies the BES connection holding period when the BES connection holding facility is used.	
109	PDRDABLKF	Specifies the number of rows to be sent in one transfer when retrieval results are transferred from a distributed server to a distributed client.	Distributed database
110	PDODBSTATCACHE	Specifies whether column information or index information that is collected the first time an ODBC function (<code>SQLColumns()</code> or <code>SQLStatistics()</code>) is issued is to be cached.	ODBC functions
111	PDODBESCAPE	Specifies whether the "&" ESCAPE character is to be specified for the pattern character string in a retrieval that uses a cataloging ODBC function.	
112	PDGDATAOPT	Specifies that the <code>SQLGetData</code> function of ODBC is to fetch data from columns, even if the data has already been fetched from those columns.	
113	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.	
114	PDODBSPLITSIZE	Specifies the partition acquisition size when <code>PDODBLOCATOR=YES</code> is specified.	
115	PDODBCWRNSKIP	Specifies whether warnings are to be skipped when an ODBC connection is used.	

No.	Environment variable	Function	Category
116	PDJETCOMPATIBLE	Specifies whether the ODBC 3.0 driver is to be operated in a Microsoft Jet database engine compatible mode not based on the ODBC 3.0 specification.	
117	PDPLGIXMK	Specifies whether delayed batch creation of plug-in indexes is to be used.	Plug-ins
118	PDPLUGINNSUB ²	For details, see the manual for the target plug-in.	
119	PDPLGPFSZ	Specifies the initial size of the index information file for delayed batch creation of plug-ins.	
120	PDPLGPFSZEXP	Specifies the extension size of the index information file for delayed batch creation of plug-ins.	
121	PDJDBFILEDIR	Specifies the file output destination of the Exception trace log using the Type4 JDBC driver.	JDBC driver
122	PDJDBFILEOUTNUM	Specifies the number of outputs to the Exception trace log file using the Type4 JDBC driver.	
123	PDJDBONMEMNUM	Specifies the number of acquired information items in the Exception trace log memory using the Type4 JDBC driver.	
124	PDJDBTRACELEVEL	Specifies the trace acquisition level of the Exception trace log using the Type4 JDBC driver.	

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*.

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 version

If the IP address is to be inherited, specify the host name of the primary system.

- For the Windows version

If the IP address is to be inherited, specify the virtual network name registered for the MSCS network name. For details about the virtual network, see the *HiRDB Version 8 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

1. 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.

(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 8 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 version

If the IP address is to be inherited, specify the host name of the primary system.

- For the Windows version

If the IP address is to be inherited, specify the virtual network name registered for the MSCS network name. For details about the virtual

network, see the *HiRDB Version 8 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

1. Do not specify an FQDN if the version of the HiRDB server to be connected is earlier than 05-03.
2. In some cases, after the maximum client wait time (`PDCWAITTIME`) elapses, the server process cannot be cancelled and thus remains.

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_sqlc` 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_sqlc` 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 5L version.

PC

Specify this server type if the HiRDB server is the Linux or Windows version.

(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. Specify the value that was specified for the `pd_service_port` operand in the system definition of the HiRDB server to be connected.

If multiple front-end servers are used, the high-speed connection port number of the front-end server to be connected should be specified. For details about the `pd_service_port` operand, see the manual *HiRDB Version 8 System Definition*.

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:

HiRDB/Single Server

- PDHOST
- PDNAMEPORT
- PDSERVICEGRP

HiRDB/Parallel Server

- PDHOST
- PDNAMEPORT
- PDFESHOST
- PDSERVICEGRP

Note

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.

(7) *PDFESGRP=FES-group[,switchover-FES-group[,switchover-FES-group]...]*

~ <character string> ((up to 1024 bytes))

This environment variable can be specified when HiRDB/Parallel Server is used and the `pd_service_port` operand is specified in the system definition.

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`). A specification example is shown below.

```
host1: fes1:20001
```

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.

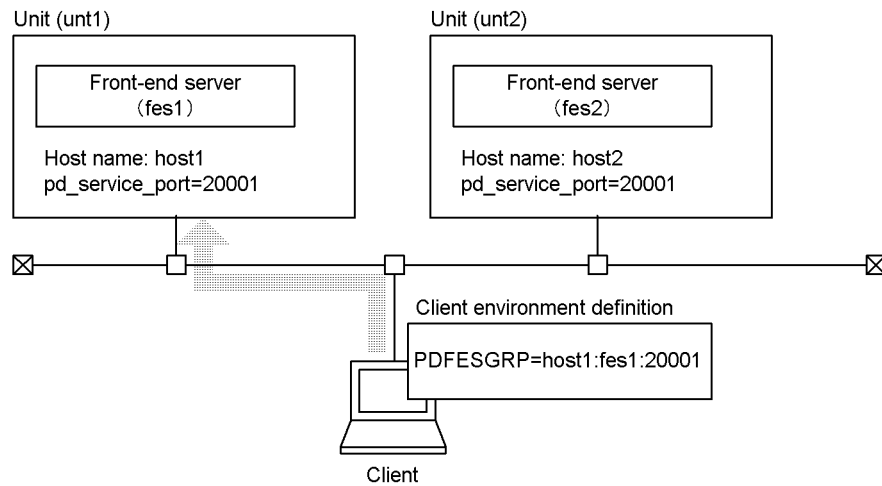
A switchover FES group is specified in the same way as a FES group.

Notes

1. When this environment variable is specified, the specifications for `PDFESHOST`, `PDSERVICEGRP`, and `PDSERVICEPORT` 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.

Usage example

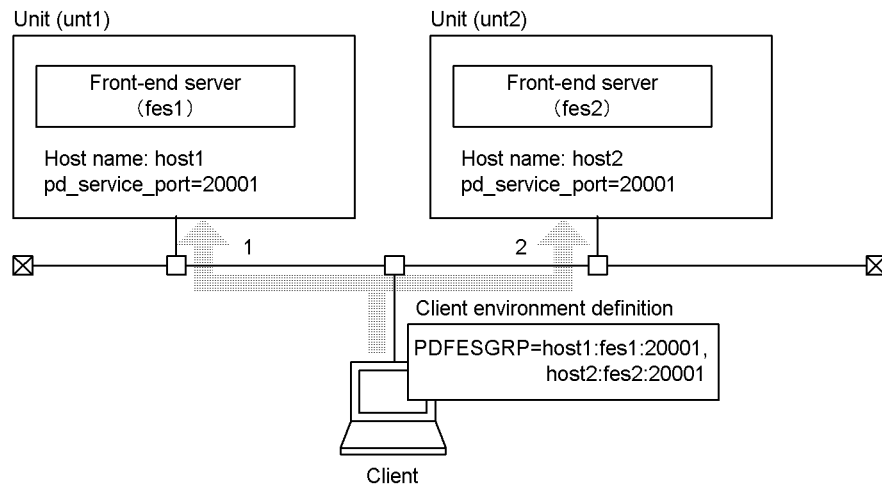
When only one FES group is specified



Explanation

If only one FES group is specified, the client is connected only to front-end server fes1 of host host1.

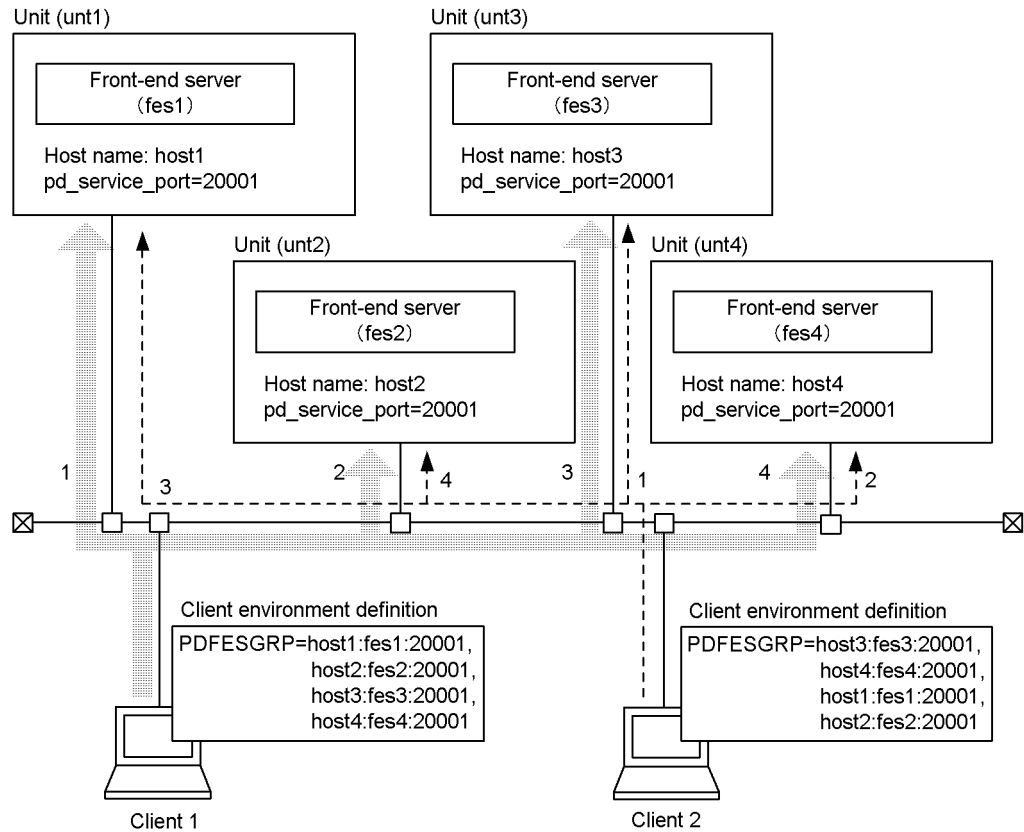
When one FES group and one switchover FES group are specified



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-recv-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 process 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 process. Therefore, if a range that includes the same port numbers is specified for multiple UAP processes 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 connected through Microsoft Jet, multiple connections with the HiRDB server are established automatically.
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. The default value is the IP address corresponding to the standard host name of the client host.
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. If PDIPC=MEMORY is specified, the PDCLTRCVADDR specification is ignored.
4. If 1 is specified in the `pd_change_clt_ipaddr` operand of the system definitions, the PDCLTRCVADDR specification is ignored.

(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 8 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 `PDXAMODE` setting must match.

When the client is linked with TPBroker

At the completion of a TPBroker transaction, a transaction completion process that is different from the one used by the UAP process is used. Therefore, 1 must

be specified in `PDXAMODE` when the client is linked with TPBroker. If 0 is specified, UAP process 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 process is used. Therefore, 1 must be specified in `PDXAMODE` when the client is linked with TPBroker. If 0 is specified, UAP process 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 process transactions cannot be completed.

When the client is linked with TP1/EE (limited to the UNIX version)

Specify 0 in `PDXAMODE`. If `PDXAMODE` is omitted or if 1 is specified, transactions sometimes cannot be completed.

(12) `PDXACANUM=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*)
 × (*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:

$$\text{specification-value} = a \times b \div (c - d \times e)$$

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 X/Open-compliant API, 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.

The SEPARATE specification is recommended if the UAP is a resident process.

(15) *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 version

If the IP address is to be inherited, specify the host name of the primary system.

- For the Windows version

When the IP address is inherited, specifies a virtual network name registered for the MSCS network name. For details about the virtual network, see the *HiRDB Version 8 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.

(16) *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 8 System Definition*.

(17) *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.

(18) *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 8 Installation and Design Guide*.

(19) `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.
3. If the directory server linkage facility is used, the directory server performs user ID and password management and user authentication when HiRDB is connected (HiRDB does not perform these tasks). Therefore, you must

specify the user ID and password registered in the directory server. If you specify a user ID and password that are not registered in the directory server and then execute a utility, an error will occur during user authentication.

(20) 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 `pdlis` 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.

(21) PDCLTLANG={SJIS|CHINESE|UJIS|C}

This environment variable specifies the character code classification to be used in the UAPs to be processed by the preprocessor. For the Windows version, only SJIS can be specified, and SJIS is assumed when this environment variable is omitted.

SJIS

`ja_JP.SJIS` (`ja_JP` or `ja_JP.PCK`) is set as the character code classification. For Linux, use `PDLANG` to set SJIS.

CHINESE

`chinese_s` is set as the character code classification.

6. Client Environment Setup

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.

During UAP preprocessing, the character code classification is determined as shown in the following table.

PDCLTLANG	Client operating system				
	HP-UX	Solaris	AIX 5L	Linux	Windows
SJIS	ja_JP.SJIS	ja_JP.PCK	ja_JP	Error	ja_JP.SJIS
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
No setting*	ja_JP.SJIS	ja	ja_JP	ja	ja_JP.SJIS
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
SJIS	C	—	—	—
CHINESE	—	C	—	—
UJIS	—	—	C	—
C	C ²	—	C ²	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`.

(22) **`PDLANG={UTF-8|SJIS|CHINESE|ANY}`**

This environment variable is not valid in the Windows environment.

This environment variable specifies that the character code classification for UAP execution is Unicode (UTF-8) or the EUC Chinese character code. In Linux, this environment variable specifies `SJIS` if `SJIS` is to be used. When this environment variable is omitted, the specification value of the `LANG` environment variable is assumed.

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.

(23) **`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 8 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

In a distributed database environment, the system collects a log of all database updates, regardless of how this environment variable is specified at the server.

(24) 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*.

(25) 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 8 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 8 SQL Reference*.

(26) 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.

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.

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 or from version 02-06 or later of HiRDB SQL Executer.

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 or from version 02-06 or later of HiRDB SQL Executer.

Note that the converted character code range is that of Unicodes (UTF-16), including surrogate pairs.

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 (SQLCND A)
- Error messages stored in the SQL Communications Area

- Data type names stored in the Type Name Descriptor Area (SQLTND)

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
SJIS	No conversion necessary	UTF8, UTF8MS, UTF8_TXT, UTF8_EX, UTF8_EX2, UTF8MS_TXT	UJIS, UJIS2	NOUSE
Unicode (UTF-8)	Cannot be specified	No conversion necessary	Cannot be specified	NOUSE
UJIS	Cannot be specified	Cannot be specified	No conversion necessary	NOUSE
UCS-2	Cannot be specified	UCS2_UTF8	UCS2_UJIS	Not needed
C	NOUSE	NOUSE	NOUSE	NOUSE

Legend:

Not needed: Cannot be specified because conversion is not possible.

No conversion: Does not need to be specified because code conversion is not necessary.

Tables 6-26 and 6-27 show the differences in character code conversions when UTF8, UTF8_EX, and UTF8_EX2 are specified.

Table 6-26: 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	

Character received from HiRDB server (Unicode (UTF-8))	PDCLTCNVMODE setting	Character code after HiRDB client conversion (SJIS)
	UTF8_EX2	

* Character code is not converted.

Table 6-27: 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 (SQLCND A), error messages stored in the SQL Communications Area, or data type names stored in the Type Name Descriptor Area (SQLTND A):

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.

(27) PDCLTGAIJIDLL=user-defined-external-character-conversion-DLL-file-name

~ <character string>

This environment variable is valid only in the Windows version.

This environment variable specifies the name of the DLL file for user-defined external character conversion. This environment variable is valid only if a value other than `NOUSE` is specified for `PDCLTCNVMODE`. If this environment variable is omitted, double-byte external characters are converted to double-byte number signs (#).

(28) PDCLTGAIJIFUNC=user-defined-external-character-conversion-function-name

~ <character string>

This environment variable is valid only in the Windows version.

This environment variable specifies the name of the user-defined external character conversion function. This environment variable is valid only when `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 WINAPI2
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 Unicoes)
- 4: Data conversion from the HiRDB server to the HiRDB client (for Unicoes)
- 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

Note

When the Gaiji conversion DLL passes a Unicode, the data is converted to 2 bytes of UCS-2 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 character string size is fixed to two bytes.

`instr[0]` = First byte of external character to be converted

`instr[1]` = Second byte of external character to be converted

`outstr`

Indicates the pointer to the post-conversion external character storage area. The character string side is fixed to two bytes. A calling side HiRDB client library secures or releases the area.

`outstr[0]` = First byte of character code (external character) after conversion

`outstr[1]` = Second 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 (Unicode) 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`.

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

direct	instr	ostr	PDCLTCNVMODE
4	Unicode external character codes	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 Unicode character codes	External EUC Japanese kanji character codes	UCS2_UJIS
6	External EUC Japanese kanji character codes	External Unicode character codes	UCS2_UJIS

The following table shows the external code ranges for each character code classification.

Character code	First byte	Second byte
Shift JIS kanji codes	0xf0-0xfc	0x40-0x7e
		0x80-0xfc
Windows signed character set	0xf0-0xfa	0x40-0x7e
		0x80-0xfc
EUC Japanese kanji codes	0xf5-0xfe	0xa1-0xfe
Unicodes or MS-Unicodes*	0xe0-0xf8	0x00-0xff

* 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.

(29) 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 8 System Operation Guide*.

(30) *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
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 8 Installation and Design Guide*.

(31) **`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 *4.16 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 an application uses an X/Open-compliant API to access the HiRDB server, the `PDAUTORECONNECT` specification is ignored and NO is always assumed.
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.

(32) *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.

(33) *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.

(34) `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 8 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.

(35) `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.

(36) PDHATRQUEUING=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 8 System Operation Guide*.

(37) 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.

(38) 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 version,

/etc/services; for the Windows version
%windir%\system32\drivers\etc\services).

(39) 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.

(40) 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.

(41) 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.

(42) 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 *10.1.5 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.`)

(43) `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 in the same host, the `PDIPC` specification is ignored (the system assumes that `DEFAULT` was specified). In this case, the connection process may take longer.
2. If you use the XA interface library for multiple threads (`pdcltxm.dll` for Windows clients, and `libzcltxk.sl(so)` or `libzcltyk.sl(so)` for UNIX clients) to access HiRDB with the XA interface, and a UAP running on TPBroker or Weblogic Server is set as the client, the specification of this environment variable is ignored, and it is assumed the `DEFAULT` specification was specified for `PDIPC`.
3. If `PDIPC=MEMORY` is specified in UNIX-version clients, HiRDB allocates a common memory size equal to values specified for `PSENDMEMSIZE` 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 `PSENDMEMSIZE` and `PDRECVMEMSIZE`.
4. If `PDIPC=MEMORY` is specified, the specification for `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.

(44) `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:

$$\text{specification-value (bytes)} = \lceil (400 + 16 \times \text{number-of-retrieved-columns} + 16 \times \text{number-of-?-parameters} + \text{SQL-statement-length}) \div 4096 \rceil \times 4$$

The value calculated with this formula differs from the data size that is actually sent during communication.

(45) `PDRECVMEMSIZE=`data-recv-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:

$$\text{specification-value (bytes)} = \lceil (600 + 25 \times \text{number-of-retrieved-columns} + \sum \text{column-data-length}) \div 4096 \rceil \times 4$$

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* × *number-of-array-columns* or *column-data-length* × *number-of-repetition-column-elements*.

If `PDBLKF` is specified, calculate the value based on the following formula:

$$\text{specification-value (bytes)} = \uparrow (600 + 19 \times \text{number-of-retrieved-columns} + (7 \times \text{number-of-retrieved-columns} + \sum \text{column-data-length}) \times \text{PDBLKF-value}) / 4096 \uparrow \times 4$$

The value calculated with this formula differs from the data size that is actually sent during communication.

(46) **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.

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.

(47) **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.

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.

(48) `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.

If 0 is specified, the HiRDB server continues to wait until it receives a request from the HiRDB client.

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.

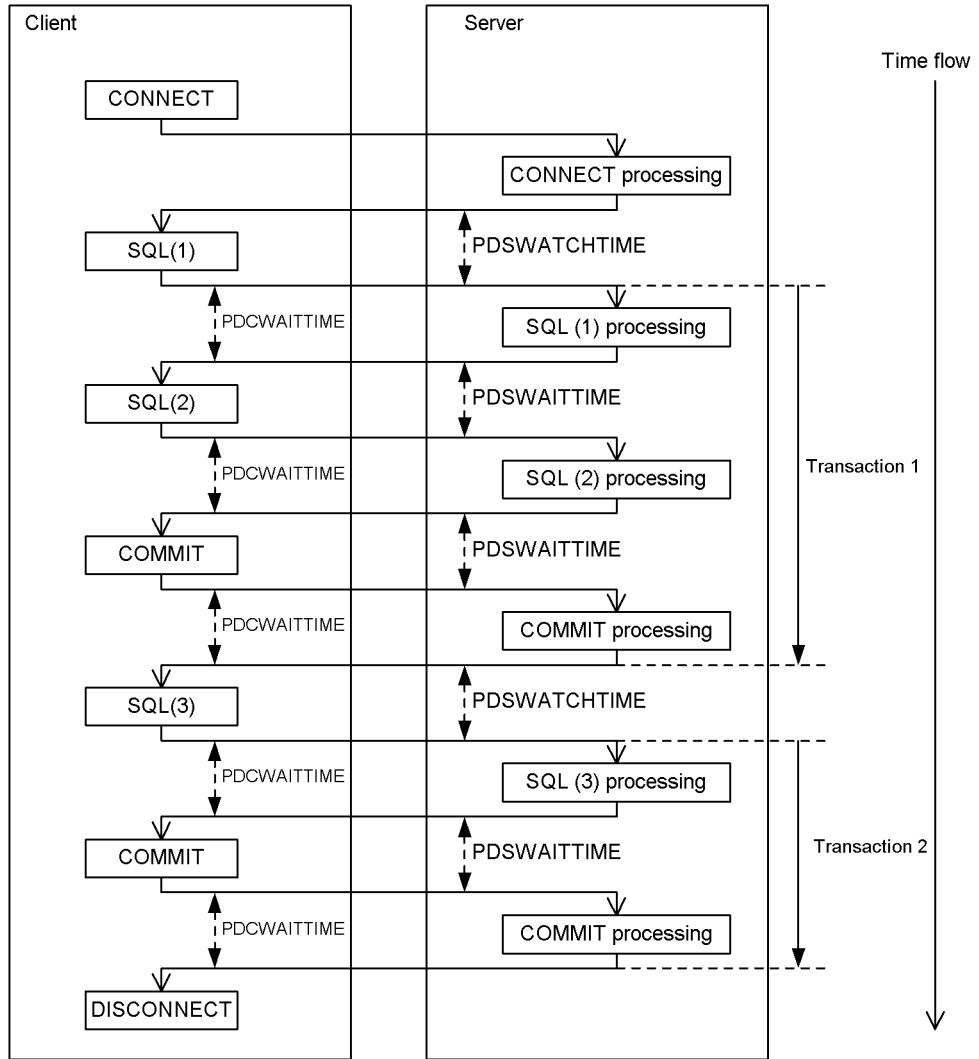
Relationship to the system definition

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 8 System Definition*.

Relationship with other environment variables

Figure 6-2 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.

(49) 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 8 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 8 System Definition*.

(50) 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 an application uses an X/Open-compliant API to access HiRDB, the actual PDKALVL specification is ignored and 0 is always assumed.

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 `PKALVL`, 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 `PKALVL`, 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 `PKALVL` and specify a time that is shorter than the firewall monitoring time in `PKATIME`. (For example, if the firewall monitoring time is 1,200 seconds, specify 1,000 seconds in `PKATIME`.)
 - 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 `PKALVL` and specify a time that is shorter than the `PDSWATCHTIME` monitoring time in `PKATIME`. (For example, if the `PDSWATCHTIME` monitoring time is 3,600 seconds, specify 3,000 seconds in `PKATIME`.)
 - 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.

(51) 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.

(52) 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 version) 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. In particular, switchover to the standby system takes a long time if a network failure or other failure occurs while the facility for system switchover without IP address inheritance is being used. In an environment that uses the facility for system switchover without IP address inheritance, the switchover time to the standby system can be shortened by setting a small number of retries.

(53) 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.

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:

$$\text{MAX}(A + 1, 8)$$

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.

(54) 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* × 0.2, 300)

(55) PDCLTPATH=trace-file-storage-directory

~ <path name> ((path name of current directory))

This environment variable specifies the storage directory for SQL trace files and error log files created by the HiRDB client.

(56) 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 *10.1.1 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.

(57) PDUAPERLOG=error-log-file-size

~ <unsigned integer> ((0, 4096-2000000000)) <<65536>> (bytes)

This environment variable specifies the size of the file into which the error log of the UAP is to be output.

If 0 is specified, the maximum file size is assumed, and an 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 the error log, see *10.1.2 Error logging*.

Relationship with other environment variables

The error log 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`.)

(58) `PDERRSKIPCODE=SQLCODE[,SQLCODE]...`

This environment variable specifies SQLCODEs for which message output to the 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 X/Open-compliant API, two 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.

(59) `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 *10.1.1 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.

(60) 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.

(61) 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.

(62) 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 *10.1.4 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 the output of access path information or SQL runtime interim results is specified, the server load may increase because SQL objects are re-created even if they already exist in the buffer.
2. UAP unit information is not output for programs that use an X/Open-compliant API under OLTP.
3. If the size of the access path information or SQL runtime interim results exceeds one 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 PDIPC=MEMORY in the client environment definitions, PDIPC=DEFAULT results.

(63) 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 pdHHMMSSmmm_XXX_1.trc and pdHHMMSSmmm_XXX_2.trc.

(64) 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 *10.1.6 SQL trace dynamic acquisition facility*.

(65) 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.

(66) 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.

(67) 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.

(68) 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 *10.1.7 Reconnect trace facility*.

(69) 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 8 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

pdwrtlnxxxx-1.trc and pdwrtlnxxxx-2.trc

xxxx: Process ID when the UAP is executed

(70) 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.

(71) 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 `PDWRTLNCOMSZ` specification value, the excess information is ignored. In this case, `**PDWRTLNCOMSZover**` is output in the following line.

(72) 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 *10.1.3 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.

(73) *PDUAPEXERLOGPRMSZ=maximum-data-length-of-parameter-information*

~ <unsigned integer> ((0-32008)) (bytes)

This environment variable specifies the maximum data length of parameter information to be output to error log and SQL error report files 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.

(74) *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 8 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. For details about the access path information file for HiRDB SQL Tuning Advisor, see *10.1.8 Access path information file 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-version 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 *10.1.4 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

SQL type	Condition	PDVWOPTMODE specification value		
		0	1	2
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.

(75) 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. For details about the access path information file for HiRDB SQL Tuning Advisor, see *10.1.8 Access path information file for HiRDB SQL Tuning Advisor*.

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.

(76) 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 `pdtaapinfHHMMSSmmm_XXX_1` and `pdtaapinfHHMMSSmmm_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)

(77) 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.

(78) 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 8 System Definition*. For details about the `pdstbegin` command, see the manual *HiRDB Version 8 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.

(79) PDLOCKLIMIT=maximum-locked-resource-request-count-per-user

~ <unsigned integer> ((0-32,767)) <<0>>

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.

Estimation method

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 8 System Definition*. For details about lock processing, see 3.4 *Locking*.

(80) 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
	For distributed server of distributed database	64
—	Utility	64
	Operation command	Specification value of the <code>pd_command_deadlock_priority</code> operand in the system definition
	Other operation command	64

— : Not applicable

(81) 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*.

(82) 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 specification value of the PDISLLVL 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.

(83) 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`.

For the cursor declaration used with update processing, a value of 1 is always assumed, 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 8 SQL Reference manual*.

(84) 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 8 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.

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 8 SQL Reference*.

Specification values for the SQL optimization option

Table 6-28 shows the values that can be specified for the SQL optimization option.

Table 6-28: Specification values of the SQL optimization option

Number	Optimization method	Specification value	
		Identifier	Unsigned integer
1	Forced nest-loop-join	"FORCE_NEST_JOIN"	4

Number	Optimization method	Specification value	
		Identifier	Unsigned integer
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
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

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 6-28.

- HiRDB/Single Server

Specify item numbers 4, 6, 8, 9,14, and 16. 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"
```

- HiRDB/Parallel Server

Table 6-29 shows the recommended specification values for the SQL optimization option.

Table 6-29: 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 be 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 valid only for 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 valid only when a HiRDB/Parallel Server is used.

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.

1. 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

2. 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*.

3. 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 specification is valid 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)*.

4. 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)*.

5. 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.

6. 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.

7. 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.

Table 6-30 shows the relationships between SQL statements that create work tables and the option for suppressing creation of update-SQL work tables.

Table 6-30: 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.

8. 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*.

9. 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 8 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 8 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 8 Installation and Design Guide*.

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.

(85) `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

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 8 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 8 SQL Reference*.

Specification values for the SQL extension optimizing option

Table 6-31 shows the specification values for the SQL extension optimizing option.

Table 6-31: 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	Suppressing foreign server execution of SQL statements that contain join operations	"DETER_JOIN_SQL"	67108864
4	Forcing foreign server execution of SQL statements that contain direct products	"FORCE_CROSS_JOIN_SQL"	13421728
5	Suppressing derivation of unconditionally created high-speed search conditions for foreign server execution	"DETER_FSVR_DERIVATIVE_COND"	1073741824

Note 1

Item numbers 2 to 5 become effective when the value for application of optimizing mode 2 based on cost is specified.

Note 2

Item numbers 3 to 5 are optimization methods that become effective when data is retrieved from an external table. In other cases, these methods are ineffective.

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. Suppressing foreign server execution of SQL statements that contain join

operations

When SQL statements for accessing a foreign table are created from queries that contain accesses to the foreign table, this optimization method suppresses the creation of SQL statements that contain join operations.

When this optimization method is specified, HiRDB does not create SQL statements that contain join operations. Instead, HiRDB creates SQL statements that fetch foreign table data that becomes the join input, and HiRDB performs the join processing.

For details about suppressing foreign server execution of SQL statements that contain join operations, see the manual *HiRDB External Data Access Version 7*.

4. Forcing foreign server execution of SQL statements that contain direct products

When this optimization option is specified, HiRDB tries to create SQL statements that contain direct products as much as possible when it creates SQL statements that access a foreign table from queries that contain accesses to the foreign table.

For details about forcing foreign server execution of SQL statements that contain direct products, see the manual *HiRDB External Data Access Version 7*.

5. Suppressing derivation of unconditionally created high-speed search conditions for foreign server execution

High-speed retrieval conditions that are derived unconditionally and can be executed in foreign servers can be suppressed.

When high-speed search conditions are derived, generation and execution of the high-speed search condition may take a long time and the resulting access path may not be the intended path. In such cases, specify this optimization method.

If the value for deriving high-speed search conditions is specified in the SQL optimization options, this optimization method is ignored even when specified.

For details about deriving high-speed search conditions, see *4.5.11 Deriving high-speed search conditions*.

(86) 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.

(87) ***PDDLVAL={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 8 SQL Reference* manual.

(88) ***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 8 Installation and Design Guide*.

(89) **PDCMMTBDDL={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.

(90) **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.

(91) 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.

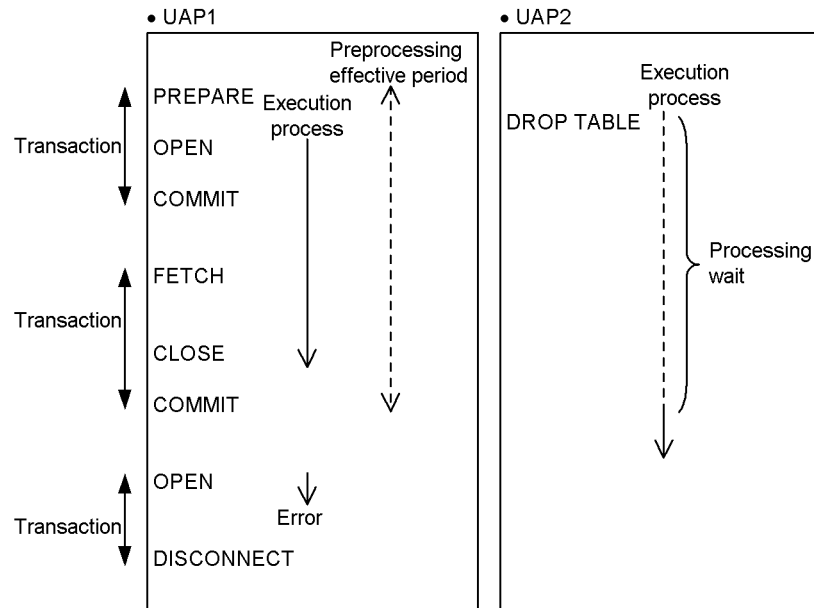
(92) PDDLDEAPRP={YES|NO}

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.



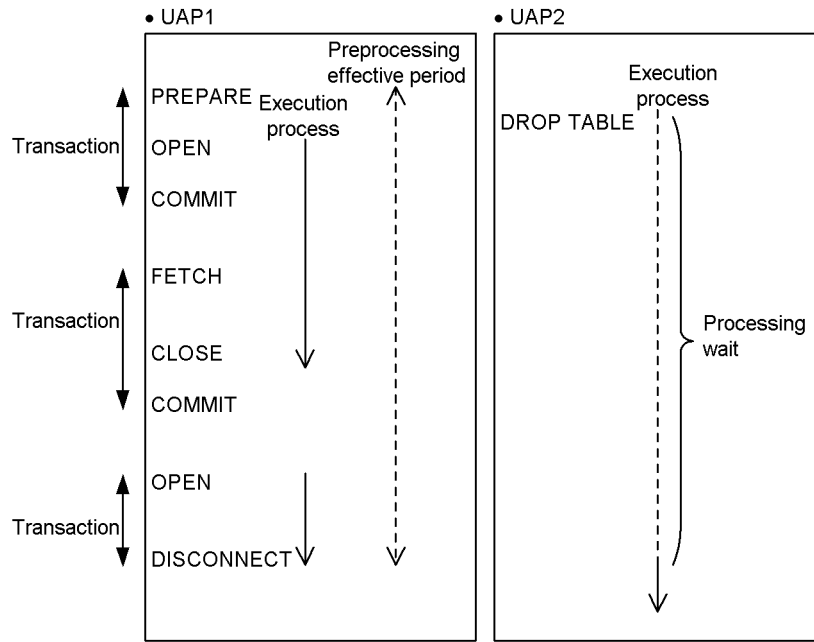
Explanation

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.

(93) 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.

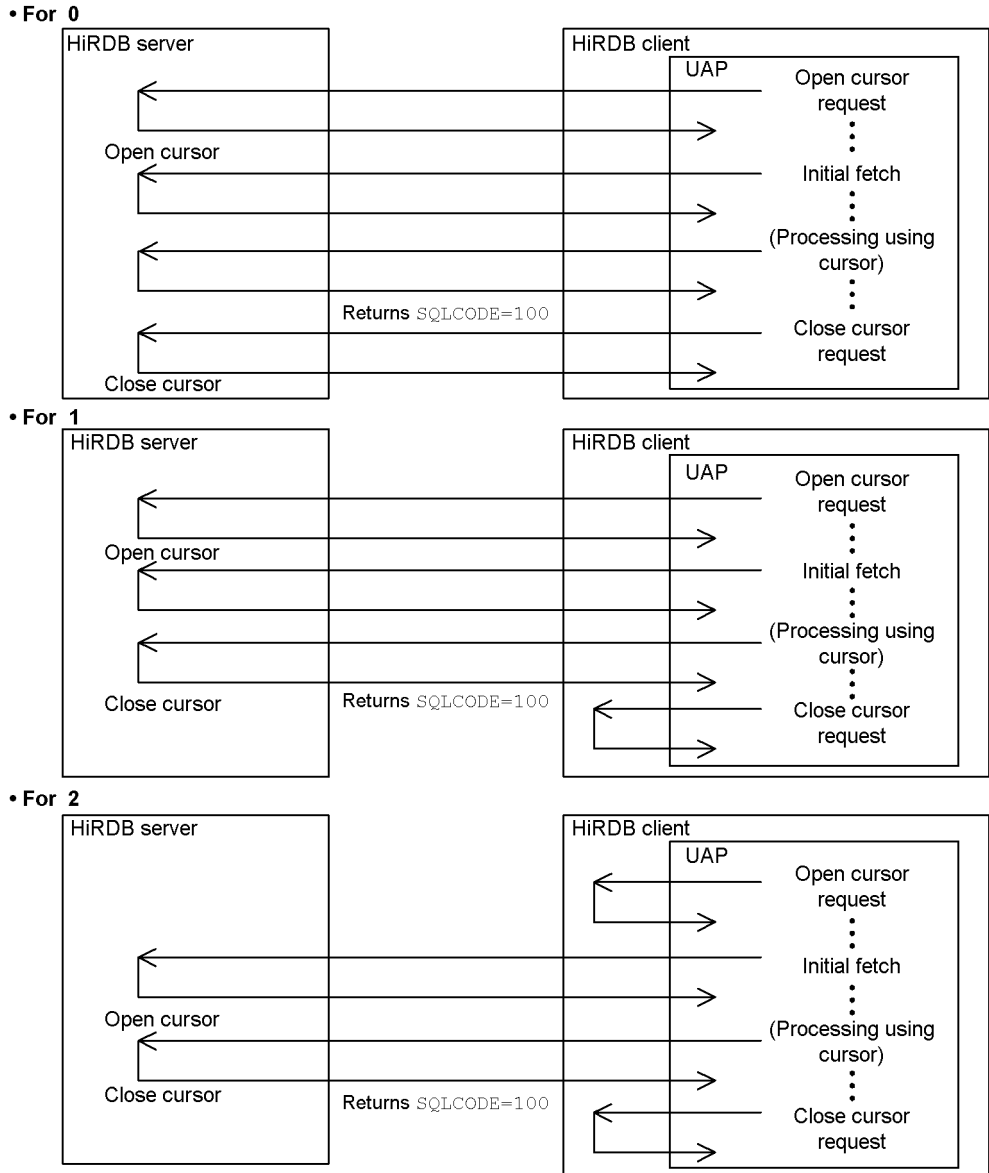
2:

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.

Figure 6-3 provides an overview of the processing for each setting.

Figure 6-3: Overview of processing for each setting of PDCURSRLVL



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.

(94) `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 8 System Definition*.

Note

For a Windows version 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.

(95) 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.

(96) 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 `PDBLKBUFSIZE` 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 8 Installation and Design Guide*. The calculation expression for memory required in the client is shown below.

Memory calculation expression (kilobytes)

$$= \uparrow (600 + 19 \times \text{retrieval-column-count} + (7 \times \text{retrieval-column-count} + \sum \text{defined-column-length}^*) \times \text{PDBLKF-value}) \div 4096 \uparrow \times 4$$

* The unit is bytes.

(97) 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 PDBLKBUFSIZE is 1 or higher.

(98) PDBLKBUFSIZE=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 PDBLKBUFSIZE 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

(99) PDNODELAYACK={YES|NO}

This environment variable is limited to the AIX 5L 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 8 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 5L 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.

(100) **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.

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.

(101) **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.

(102) PDSPACEVL={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 *G.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 *G.2 Space conversion function*.

(103) 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.

(104) 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 version. 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.

(105) 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 (SQLCND A), and data type names stored in the Type Name Descriptor Area (SQLTND A) 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.

(106) 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.

(107) 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 8 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.

(108) 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.

(109) PDRDABLKF=batch-retrieval-row-count

~ <unsigned integer> ((1-4096))

This environment variable can be specified when the HiRDB is the HP-UX or AIX 5L version.

This environment variable specifies the number of rows to be transferred in one transfer operation when retrieval results are transferred from a distributed server to a distributed client. A value roughly between 50 and 80 should be specified. Specify 1 for this environment variable if batch retrieval is not used.

Specifying a large value reduces the communications overhead and the search time but increases the amount of required memory. Therefore consider the balance with the memory size when determining the value. The following formula can be used to calculate the amount of required memory:

$(328 + 32 a + b) c$ (in bytes)

a

Number of items specified in the SELECT clause

b

Total length of the character string data output by one transfer operation

c

Value specified for PDRDABLKF

Relationship with other definitions

When PDRDABLKF is omitted, the number of batch retrieval rows specified by the `block_fetch_count` operand (SQL environment definition of DF/UX) is assumed. For details about the `block_fetch_count` operand, see the manual *Distributed Database System DF/UX*.

(110) PDODBSTATCACHE={0|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 $60,000 / (50 + \text{table-owner-name-length} + \text{table-name-length} + \text{column-name-length} + \text{comment-length})$ rows

`SQLStatistics()`

Approximately $60,000 / (50 + \text{table-owner-name-length} + \text{table-name-length} + \text{index-name-length} + \text{column-name-length})$ rows

(111) `PDODBESCAPE={0|1}`

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 be 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.

(112) PDGDATAOPT={YES|NO}

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.

(113) 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.

(114) 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.

(115) PDODBCWRNSKIP={YES|NO}

This environment variable specifies whether warnings should be skipped for ODBC connections. For non-ODBC connections, this environment variable is ignored even if it is 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.

(116) **`PDJETCOMPATIBLE={YES|NO}`**

This environment variable specifies whether the ODBC 3.0 driver is to be operated in Microsoft Jet database engine compatible mode rather than based on the ODBC 3.0 specifications.

YES:

The ODBC 3.0 driver operates in Microsoft Jet database engine compatible mode.

NO:

The ODBC 3.0 driver operates as specified.

Application standard

Specify this environment variable when an application program uses a Microsoft Jet database engine, such as 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.

(117) **`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 8 System Operation Guide*.

YES

Use delayed batch creation of plug-in indexes.

NO

Do not use delayed batch creation of plug-in indexes.

(118) PDPLUGINNSUB

For details, see the manual for the target plug-in.

(119) PDPLGPFSSZ=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.

(120) PDPLGPFSSZEXP=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.

(121) 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 *17.13 Exception trace log*. For other details, see system property `HiRDB_for_Java_FileDIR` in *17.13.1(2)(b) System property settings*.

(122) 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 *17.13 Exception trace log*. For other details, see system property `HiRDB_for_Java_FileOutNUM` in *17.13.1(2)(b) System property settings*.

(123) PDJDBONMEMNUM=number-of-acquired-information-items-in-exception-t race-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 *17.13 Exception trace log*. For other details, see system property `HiRDB_for_Java_OnMemNUM` in *17.13.1(2)(b) System property settings*.

(124) 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 *17.13 Exception trace log*. For other details, see system property `HiRDB_for_Java_TraceLevel` in *17.13.1(2)(b) System property settings*.

6.6.5 Environment variables and connection types for HiRDB servers

Table 6-32 shows the relationships among environment variables and connection types for connecting with the HiRDB server.

Table 6-32: 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

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
PDSERVICEPORT	—	S	—	S	—	—	S
PDSERVICEGRP	—	S	—	S	—	S	S
PDSRVTYPE	—	Δ	—	Δ	—	—	Δ

S: Must be specified.

Δ: Must be specified if the HiRDB server is the Linux or Windows version.

—: 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:

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 if you want to use the connected front-end server efficiently.

6.6.6 Specifying client environment definitions for foreign table access

When the foreign server is a HiRDB or XDM/RD E2 server and a foreign table is to be accessed, the specification values of some client environment definitions also apply to the foreign server. There are also client environment definitions for which fixed values are always applied to the foreign server.

Foreign tables can be accessed (the HiRDB External Data Access facility can be used) when the HiRDB is the HP-UX or AIX 5L version.

(1) When the foreign server is a HiRDB server**(a) Client environment definitions for which specification values are applied**

The specification values of the following client environment definitions are also applied to the foreign HiRDB server:

- PDISLLVL
- PDCLTAPNAME *
- PDEXWARN *
- PDDLKPRIO *
- PDLOCKSKIP *
- PDCWAITTIMEWRNPNT *

* The specification value is applied if the foreign HiRDB is Version 06-02 or later.

(b) Client environment definitions for which fixed values are applied

For the following client definitions, fixed values are applied to the external HiRDB server:

- PDSTJTRNOUT (fixed value: NO)
- PDDFLNVAL (fixed value: NOUSE)
- PDSWATCHTIME (fixed value: 0)
- PDAUTOCONNECT (fixed value: OFF)

(2) When the foreign server is an XDM/RD E2 server**(a) Foreign environment definition for which a fixed value is applied**

For the following client environment definition, a fixed value is applied to the foreign XDM/RD E2 server:

- PDSRVTYPE (fixed value: VOS3)

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 8 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.
- 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=name-service-port-number
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).

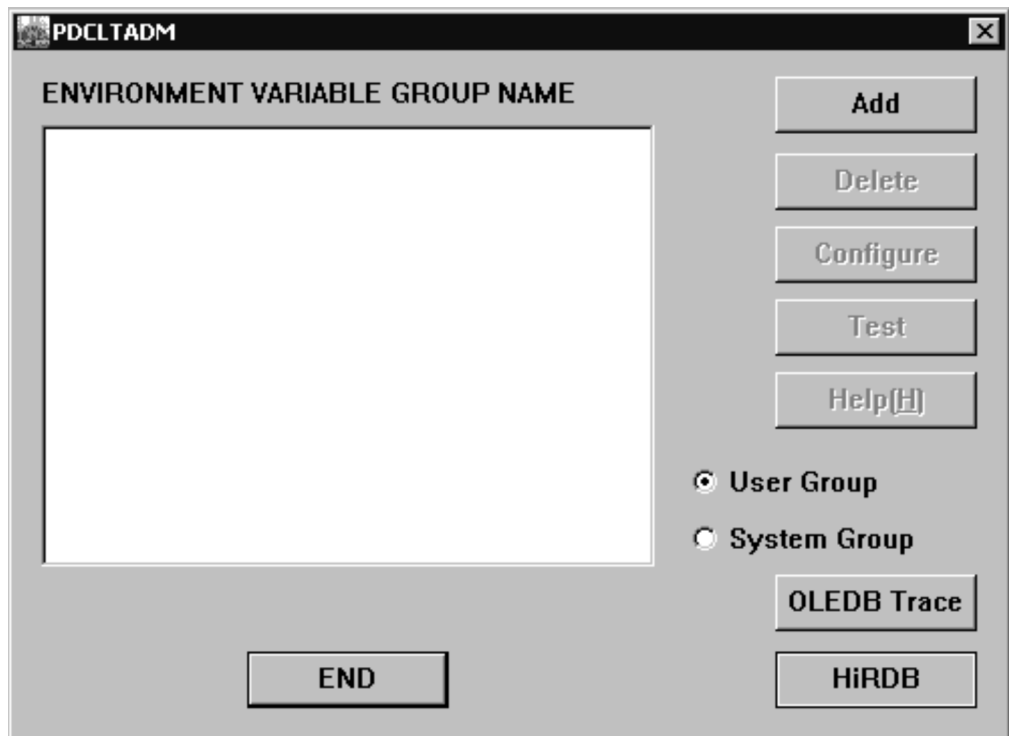
The rest of this section describes procedures for registering environment variables in the registry with the tool for registering HiRDB client environment variables.

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.



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 8 Installation and Design Guide*.

(2) Registering an environment variable group

The **HiRDB Client Environment Variable Setup** dialog box is displayed.

The screenshot shows the PDCLTADM dialog box with the following fields and sections:

- Group name**: Text input field.
- PDCLTPATH**: Text input field.
- PDHOST**: Text input field.
- PDUSER**: Text input field.
- PDUSER Configure**: A sub-section containing:
 - User ID**: Text input field.
 - Password**: Text input field.
- PDNAMEPORT**: Text input field.
- PDSQLTRACE**: Text input field.
- PDSWAITTIME**: Text input field.
- PDCWAITTIME**: Text input field.
- PDSWATCHTIME**: Text input field.
- Other environment variable Value**: A dropdown menu and a text input field.
- Client environment variable**: A table with two columns: **variable name** and **value**.

variable name	value

Explanation

Group Name

Specify the group name with up to 30 characters.

Environment variable fields

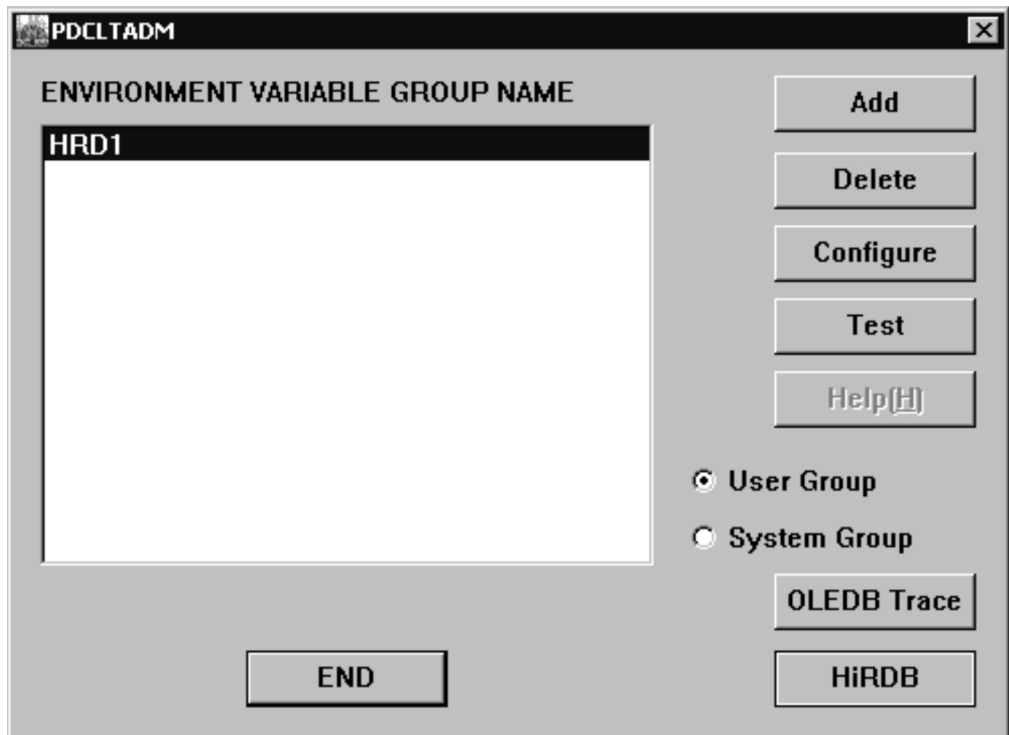
Specify a setting for each environment variable. For details about each environment variable, see *6.6.4 Environment definition information*.

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:

PDCLTADM [X]

Group name

PDCLTPATH

PDHOST

PDUSER

PDUSER Configure

User ID Password

PDNAMEPORT PDSQLTRACE

PDSWAITTIME PDCWAITTIME

PDSWATCHTIME

Other environment variable Value

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.

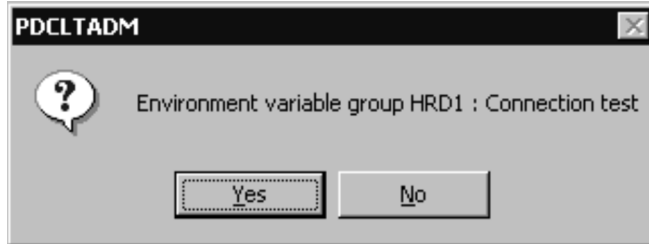
Click the **Cancel** button to cancel the client environment variable settings and return

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:



Click **Yes**.

If connection to HiRDB is successful, the following dialog box is displayed:



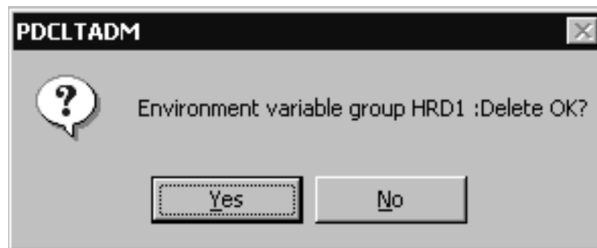
If connection to HiRDB fails, the following dialog box is displayed. If the error is due to the contents of an environment variable, change the settings of the environment variable group.



(5) Deleting an environment variable group

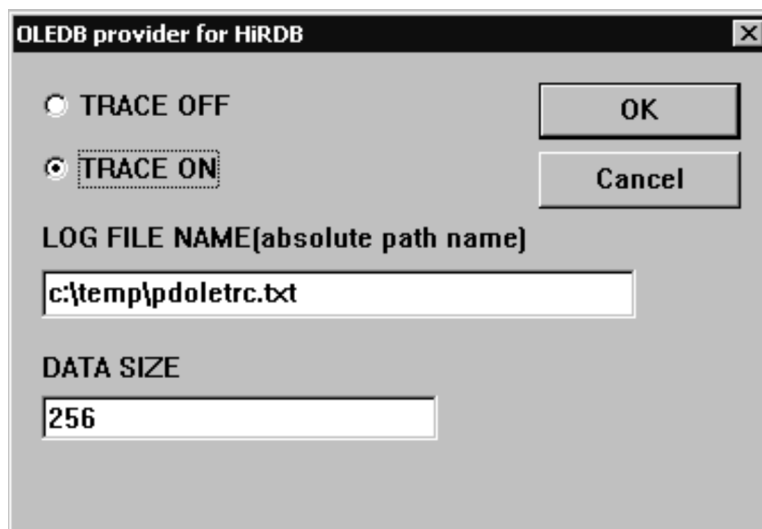
From the environment variable group name list in the **Tool for Registering HiRDB Client Variables** dialog box, select the environment variable group name to be deleted, and click the **Delete** button.

The following dialog box is displayed:

**(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:



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=name-service-port-number
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

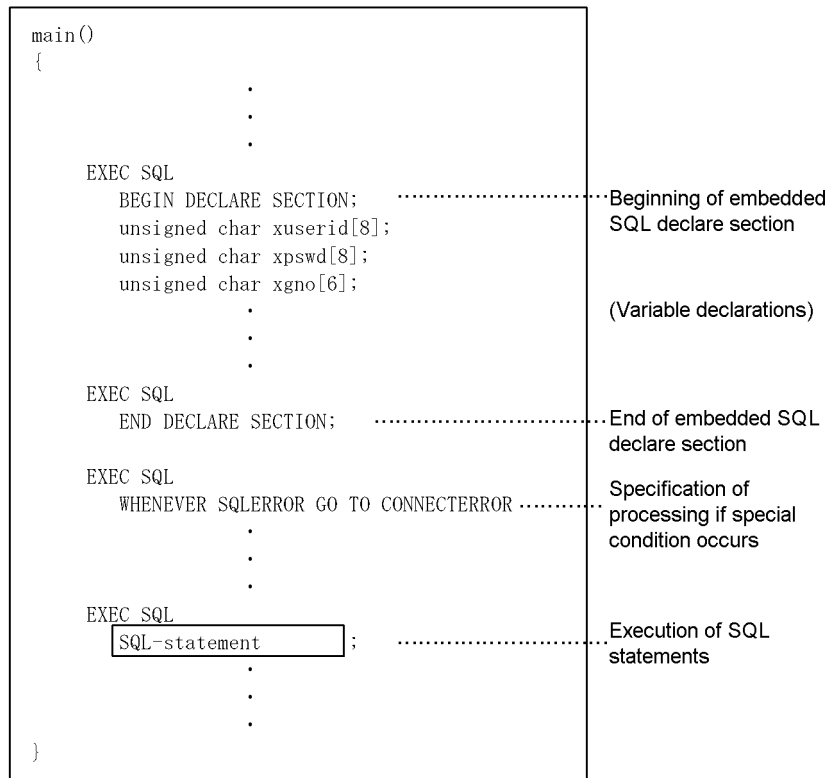
7.1 Overview

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



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 8 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 MANNNAME 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 blank 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;

```

- Table 7-1 shows the items that can be described within an embedded SQL declare section.

Table 7-1: Items that can be described within an embedded SQL declare 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 *E. 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.

Table 7-2 shows the locations where SQL statements can be described.

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 (`AFXxxxxx.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 8 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 31 characters of the variable name are distinguished. The subsequent characters are not distinguished.
 - 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:
- The following keywords, each of which starts with two underscore (`_`) characters, undergo syntax analysis but are treated as meaningless phrases. These keywords are those defined by Visual C++ 6.0. For sources that are

created with a older version of Visual C++, this restriction also applies to keywords that begin with one underscore character.

`__asm, __except, __forceinline, __int32, __stdcall, __based, __far, __inline, __int64, __try, __cdecl, __fastcall, __int8, __leave, __uidof, __declspec, __finally, __int16, and __near`

- 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.
 - Variable-length arrays 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. When you specify the `-E` option, you cannot specify trigraphs, such as `??` or `??=`. 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.

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 8 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

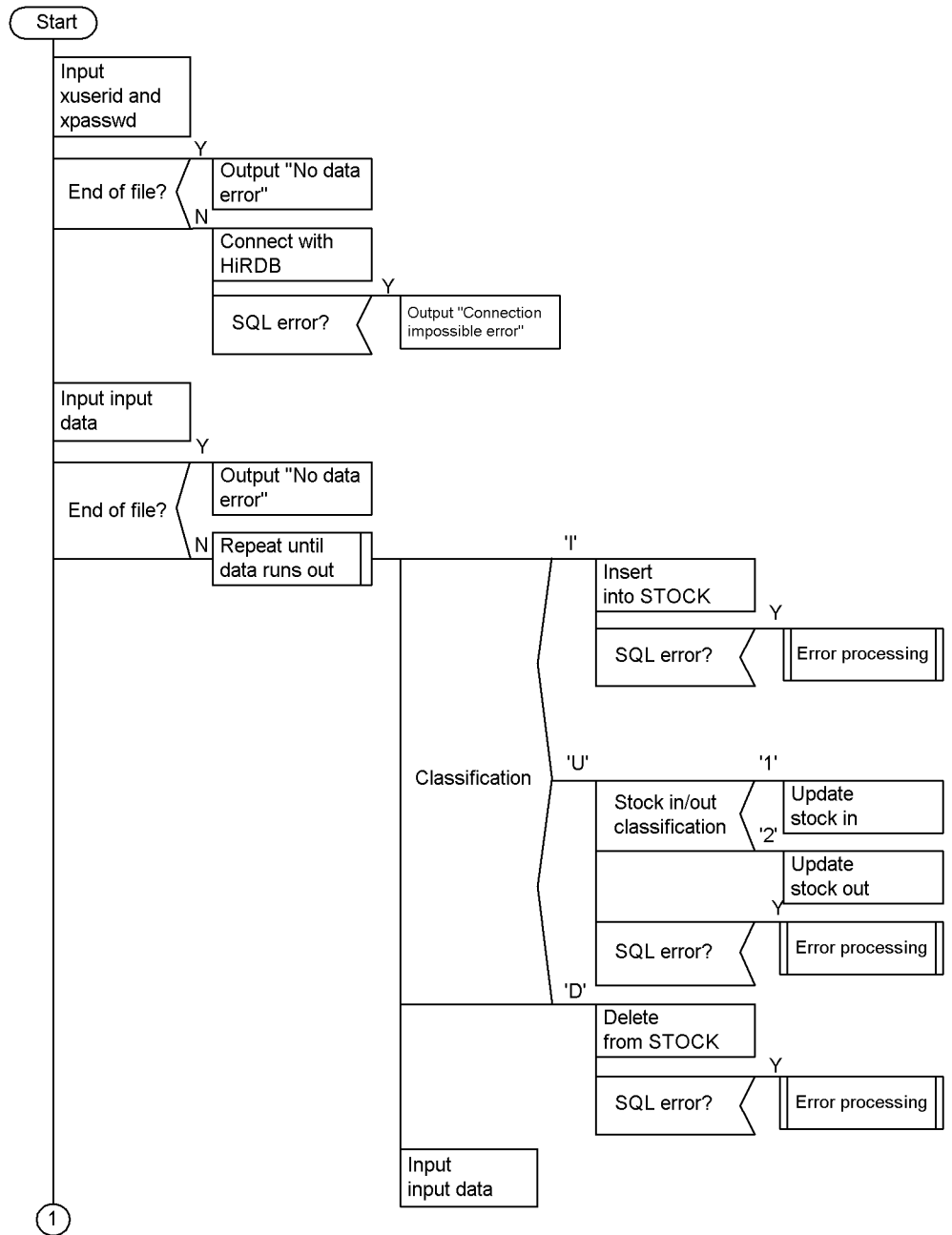
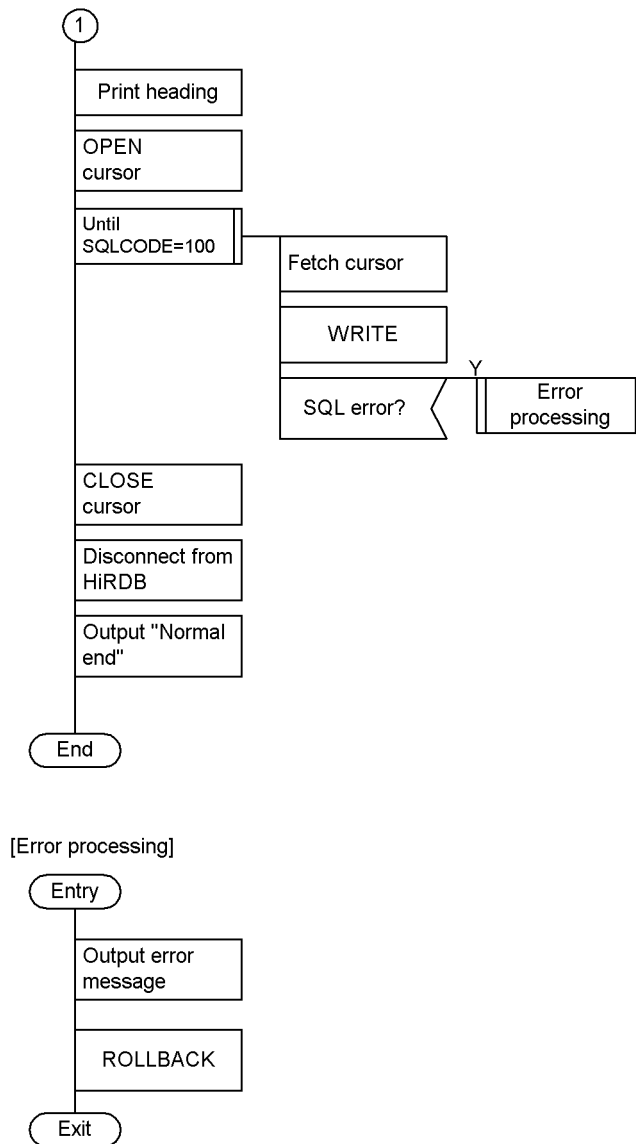


Figure 7-3: Flowchart example of an embedded SQL UAP written in C

**(b) Coding example**

A coding example of an embedded SQL UAP written in C follows:

```

1 #include <string.h>
2 #include <stdlib.h>
3
4 #define MAXCOLUMN 80          /* max column in one line */
  
```

```

5 #define INFILE "inputf1" /* input data file name */
6
7 /* declare functions */
8 void abnormalend();
9 void connecterror();
10
11 FILE *input = NULL;
12
13 main()
14 {
15     /* input data */
16     char indata[MAXCOLUMN + 1];
17
18     char in_userid[31];
19     char in_passwd[31];
20     char in_type;
21     char in_pcode[5];
22     char in_pname[17];
23     char in_color[3];
24     int in_price;
25     int in_stock;
26     char in_flux;
27
28     /* variables for SQL */
29     EXEC SQL BEGIN DECLARE SECTION;
30     char xuserid[31];
31     char xpasswd[31];
32     char xpcode[5];
33     char xpname[17];
34     char xcolor[3];
35     int xprice;
36     int xstock;
37     EXEC SQL END DECLARE SECTION;
38
39     /* input file open */
40     input = fopen(INFILE, "r");
41     if (input == NULL) {
42         /* input file open error */
43         fprintf(stderr, "can't open %s.", INFILE);
44         goto FIN;
45     }
46
47     /* get userid/passwd */
48     fgets(indata, 81, input);
49     sscanf(indata, "%30s %30s", xuserid, xpasswd);
50     if (feof(input)) {
51         fprintf(stderr, "*** error *** no data for connect
                    ***");

```

7. UAP Creation

```

52     goto FIN;                                2
53 }                                             2
54 printf("connect start,\n");                  2
55 EXEC SQL WHENEVER SQLERROR PERFORM connecterror; (a) 2
56 EXEC SQL CONNECT USER :xuserid USING :xpasswd; (b) 2
57 printf("connected,\n");                      2
58
59 /* read data from inputfile */
60 EXEC SQL WHENEVER SQLERROR PERFORM abnormalend;
61 fgets(indata, MAXCOLUMN, input);
62
63 while (!feof(input)) {
64     sscanf(indata, "%c %4s %16s %2s %8d %8d %c",
65         &in_type, in_pcode, in_pname, in_color,
66         &in_price, &in_stock, &in_flux);
67     switch (in_type) {
68     case 'I':
69         strncpy(xpcode, in_pcode, 4);
70         strncpy(xpname, in_pname, 8);
71         strncpy(xcolor, in_color, 2);
72         xprice = in_price;
73
74         xstock = in_stock;
75         EXEC SQL                                3
76             INSERT INTO
77             STOCK(PCODE,PNAME,COLOR,PRICE,SQUANTITY)
78             VALUES (:xpcode, :xpname, :xcolor, :xprice, :xstock);
79             3
80         break;
81     case 'U':
82         strncpy(xpcode, in_pcode, 4);
83         xstock = in_stock;
84         if (in_flux == '1') {
85             EXEC SQL                                (a) 4
86                 UPDATE STOCK
87                 SET SQUANTITY =SQUANTITY+:xstock
88                 WHERE PCODE=: xpcode;
89             (a) 4
90         } else {
91             EXEC SQL                                (b) 4
92                 UPDATE STOCK
93                 SET SQUANTITY=SQUANTITY-:xstock
94                 WHERE PCODE=:xpcode;
95             (b) 4
96         }
97     }
98     break;
99     case 'D':
100        strncpy(xpcode, in_pcode, 4);
101        EXEC SQL
102            DELETE FROM STOCK WHERE PCODE=:xpcode;
103        5
104        5

```

```

95         break;
96     }
97     fgets(indata, MAXCOLUMN, input);
98 }
99
100 /* print stock list */
101 EXEC SQL                                     6
102     DECLARE CR1 CURSOR FOR                 6
103         SELECT PCODE,PNAME,COLOR,PRICE,SQUANTITY FROM
104         STOCK;                             6
105 EXEC SQL OPEN CR1;                          7
106
107 /* print title */
108 printf("\n\n");
109 printf(" ***** Stock Table List *****\n\n");
110 printf(" Product code Product name Color Price
111         Current stock\n");
112 printf(" ---- -\n");
113
114 /* FETCH */
115 SQLCODE = 0;
116 while (SQLCODE <= 100) {
117     EXEC SQL WHENEVER NOT FOUND GO TO FINISH;
118     EXEC SQL                               8
119     FETCH CR1 INTO :
120     xpcode,:xpname,:xcolor,:xprice,:xstock; 8
121     EXEC SQL WHENEVER NOT FOUND CONTINUE;
122     printf(" %4s %-16s %2s %8d %8d\n",
123     xpcode, xpname, xcolor, xprice, xstock);
124 }
125
126 FINISH:
127 /* finish */
128 EXEC SQL CLOSE CR1;                         (a) 9
129 EXEC SQL COMMIT;                           (b) 9
130 printf(" *** normal ended ***\n");
131
132 FIN:
133 if (input != NULL) {
134     fclose(input);
135 }
136 EXEC SQL WHENEVER SQLERROR CONTINUE;
137 EXEC SQL WHENEVER NOT FOUND CONTINUE;
138 EXEC SQL WHENEVER SQLWARNING CONTINUE;
139 EXEC SQL DISCONNECT;                       10
140 return(0);
141 }

```

```

139
140
141 Void connecterror()
142 {
143
144     printf("\n***** error *** cannot connect ***\n");
145     fclose(input);
146     EXEC SQL DISCONNECT;
147     exit(1);
148 }
149
150
151 void abnormalend()
152 {
153     int  wsqlcode;
154
155     wsqlcode = -SQLCODE;
156     printf("\n*** HiRDB SQL ERROR SQLCODE = %d \n",
wsqlcode);
157     printf("SQLERRMC = %s\n", SQLERRMC);
158
159     EXEC SQL ROLLBACK;                (a) //
160     EXEC SQL DISCONNECT;            (b) //
161     exit(2);
162 }

```

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)

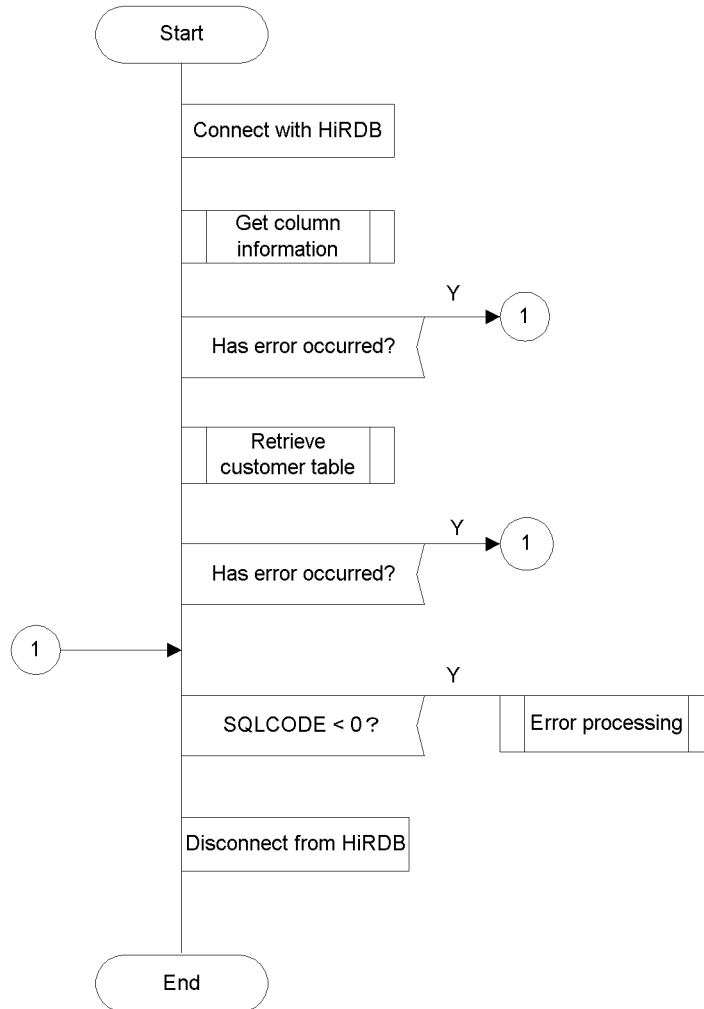


Figure 7-5: PAD chart for program example 2 (2/4)

[Column information acquisition]

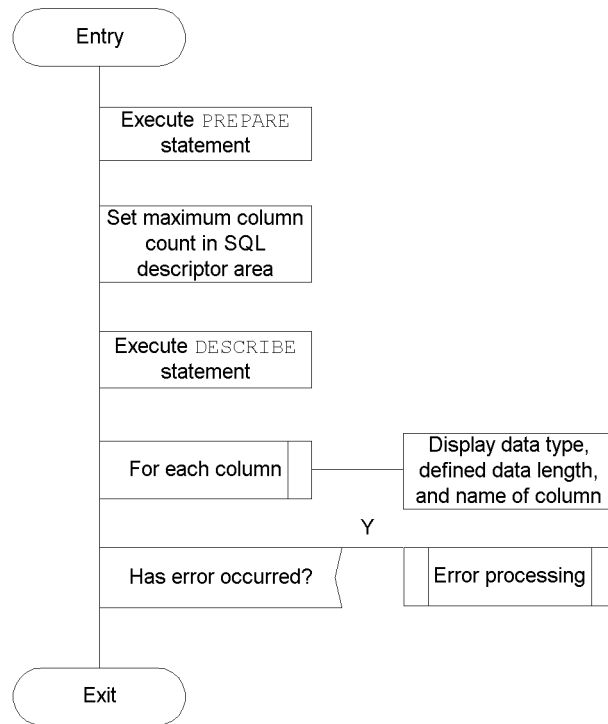


Figure 7-6: PAD chart for program example 2 (3/4)

[Customer table retrieval]

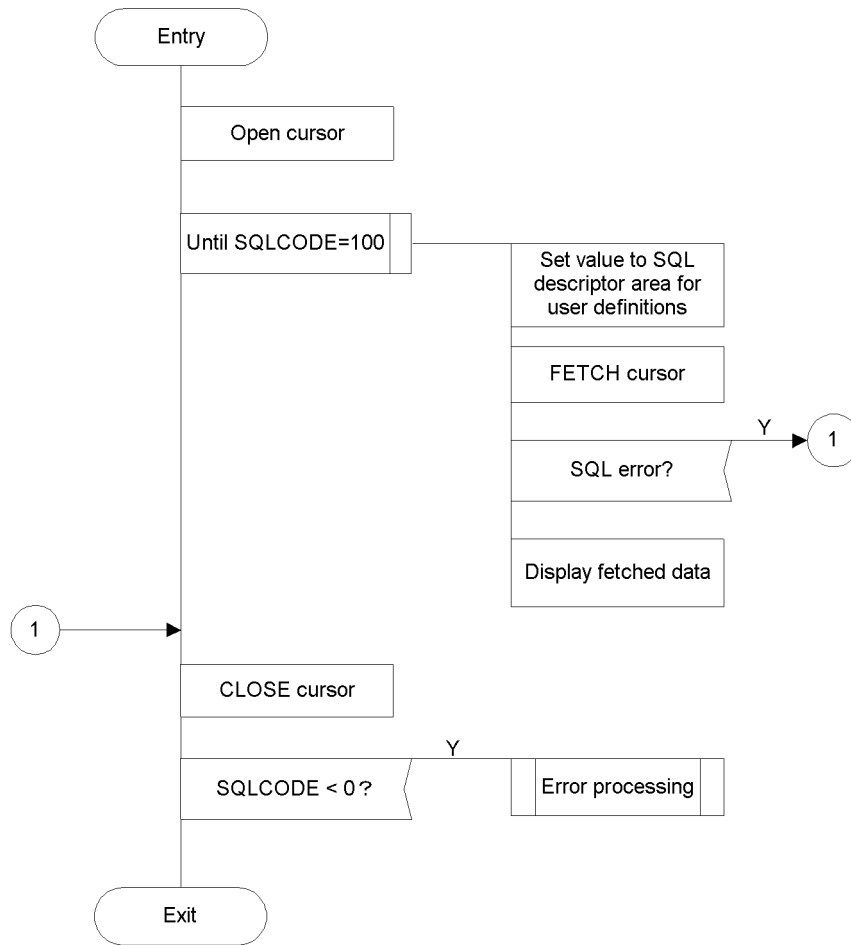
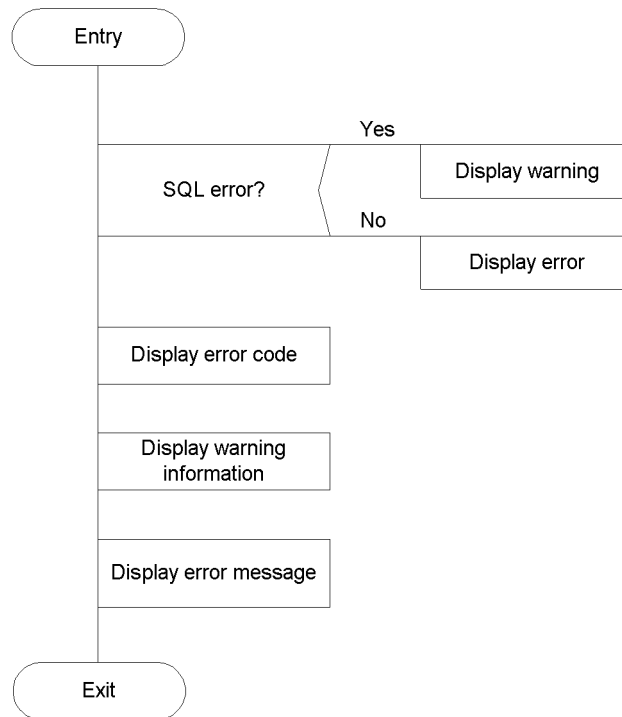


Figure 7-7: PAD chart for program example 2 (4/4)

[Error processing]

**(b) Coding example**

A coding example for program example 2 follows:

```

1  /*****
2  /*
3  /* ALL RIGHTS RESERVED, COPYRIGHT (C) 1997, HITACHI, LTD.
*/
4  /* LICENSED MATERIAL OF HITACHI, LTD. */
5  /* */
6  /* SAMPLE OF FETCH WITH SQLDA */
7  /* */
8  /*****
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include "pdbsqlda.h"

```

7. UAP Creation

```

15
16
17 static void Describe();
18 static void Fetch();
19 static void ClearSqllda(short);
20 static void errmsg();
21
22 /*****
23 /* GLOBAL VARIABLE */
24 *****/
25 short ErrFlg;
26
27 /*****
28 /* GLOBAL VARIABLE */
29 *****/
30
31 /* sqllda */
32 PDUSRSQLDA(10) xsqllda; 2
33
34 /* sqlcnda */ 3
35 struct { 3
36     short sqlnz; 3
37     struct { 3
38         short sqlnamel; 3
39         char sqlnamec[30]; 3
40     } SQLNAME[10]; 3
41 } ucnda; 3
42
43
44 /*****
45 /*
46 /* MAIN ROUTINE */
47 /*
48 *****/
49 int main(
50 int argc,
51 char *argv[])
52 {
53
54 /*****
55 /* CONNECT */
56 *****/
57 EXEC SQL
58     WHENEVER SQLERROR GOTO:ERR_EXIT
59
60     printf("***** connect start \n");
61 EXEC SQL
62     CONNECT; 4

```

```

63         printf("***** connect : END\n");
64
65     /*****
66     /*  DESCRIBE
67     /*****
68         Describe();
69         if(ErrFlg <0){
70             goto ERR_EXIT;
71         }
72
73     /*****
74     /*  FETCH
75     /*****
76         Fetch();
77         if(ErrFlg <0){
78             goto ERR_EXIT;
79         }
80
81     /*****
82     /*  END OF ALL
83     /*****
84     ERR_EXIT :
85         if(SQLCODE <0){
86             errmsg();
87             ErrFlg = -1;
88         }
89
90         EXEC SQL
91             WHENEVER SQLERROR CONTINUE;
92         EXEC SQL
93             WHENEVER NOT FOUND CONTINUE;
94         EXEC SQL
95             WHENEVER SQLWARNING CONTINUE;
96
97         EXEC SQL
98             DISCONNECT;
99
100         return (ErrFlg);
101     }
102
103
104     /*****
105     /*
106     /*  DYNAMIC CURSOR
107     /*
108     /*****
109     static void Fetch()

```

6

7. UAP Creation

```

110 {
111     EXEC SQL BEGIN DECLARE SECTION;
112     char XCUSTOM_CD[6];
113     char XCUSTOM_NAME[31];
114     char XTELNO[13];
115     char XZIPCD[4];
116     char XADDRESS[31];
117     EXEC SQL END DECLARE SECTION;
118
119     EXEC SQL
120         WHENEVER SQLERROR GOTO :Exit_Fetch;
121
122     EXEC SQL
123         DECLARE CUR2 CURSOR FOR SEL1;                                7
124
125     /*****
126     /*  OPEN CURSOR
127     /*****
128     printf("***** DYNAMIC CURSOR open start\n");
129     EXEC SQL
130         OPEN CUR2;                                                    8
131     printf("***** DYNAMIC CURSOR open : END\n");
132
133
134     /*****
135     /*  FETCH
136     /*****
137     printf("***** fetch (use sqllda) start\n");
138
139
140     EXEC SQL
141         WHENEVER NOT FOUND GOTO FETCH2_END;
142
143     for(;;) {
144         ClearSqllda(5);                                                9
145         PDSQLDATA(xsqllda, 0) = (void *)
146             XCUSTOM_CD;                                                (a) 9
147         PDSQLCOD(xsqllda, 0) = PDSQL_CHAR;                            (a) 9
148         PDSQLLEN(xsqllda, 0) = sizeof
149             (XCUSTOM_CD)-1;                                            (a) 9
150         PDSQLDATA(xsqllda, 1) = (void *)
151             XCUSTOM_NAME; (b) 9
152         PDSQLCOD(xsqllda, 1) = PDSQL_CHAR;                            (b) 9
153         PDSQLLEN(xsqllda, 1) = sizeof
154             (XCUSTOM_NAME)-1;                                          (b) 9
155         PDSQLDATA(xsqllda, 2) = (void *)XTELNO;                      (c) 9
156         PDSQLCOD(xsqllda, 2) = PDSQL_CHAR;                          (c) 9
157         PDSQLLEN(xsqllda, 2) = sizeof(XTELNO)-1;                    (c) 9

```



```

154     PDSQLDATA(xsqlda, 3) = (void *)XZIPCD;      (d) 9
155     PDSQLCOD(xsqlda, 3) = PDSQL_CHAR;          (d) 9
156     PDSQLLEN(xsqlda, 3) = sizeof(XZIPCD)-1;    (d) 9
157     PDSQLDATA(xsqlda, 4) =
158         (void *)XADDRESS;                       (e) 9
159     PDSQLCOD(xsqlda, 4) = PDSQL_CHAR;          (e) 9
160     PDSQLLEN(xsqlda, 4) =
161         sizeof(XADDRESS)-1;                     (e) 9
162
163     memset(XCUSTOM_CD, 0, sizeof(XCUSTOM_CD));
164     memset(XCUSTOM_NAME, 0, sizeof(XCUSTOM_NAME));
165     memset(XTELNO, 0, sizeof(XTELNO));
166     memset(XZIPCD, 0, sizeof(XZIPCD));
167     memset(XADDRESS, 0, sizeof(XADDRESS));
168
169     EXEC SQL FETCH CUR2
170         USING DESCRIPTOR :xsqlda;                10
171
172     printf("%s", XCUSTOM_CD);
173     printf("%s", XCUSTOM_NAME);
174     printf("%s", XTELNO);
175     printf("%s", XZIPCD);
176     printf("%s\n", XADDRESS);
177 }
178 FETCH2_END:
179     printf("***** fetch : END\n");
180
181 /*****
182 /* CLOSE CURSOR */
183 /*****
184     printf("***** close start\n");
185     EXEC SQL
186     WHENEVER NOT FOUND CONTINUE;
187     EXEC SQL
188     CLOSE CUR2;                                11
189     printf("***** close : END\n");
190
191 /*****
192 /*
193 /*****
194 Exit_Fetch:
195     if(SQLCODE <0){
196         errmsg();
197         ErrFlg = -1;
198     }
199     return;
200 }

```

7. UAP Creation

```

200
201 /*****
202 /*  DESCRIBE
203 /*****
204 static void Describe()
205 {
206     short I;
207
208     EXEC SQL
209         WHENEVER SQLERROR GOTO :Exit_Describe;
210
211 /*****
212 /*  PREPARE
213 /*****
214     printf("***** prepare start\n");
215     EXEC SQL
216         PREPARE SEL1
217             FROM 'SELECT * FROM CUSTOM'
218             WITH SQLNAME OPTION;
219     printf("***** prepare : END\n");
220
221 /*****
222 /*  DESCRIBE
223 /*****
224     PDSQLN(xsqlda) = 10;
225     printf("***** describe start\n");
226     EXEC SQL
227         DESCRIBE SEL1 INTO :xsqlda :ucnda;
228     printf("***** describe : END\n");
229
230     printf(" describe result\n");
231     printf(" NUMBER OF DATA =%d\n",
232             PDSQLD(xsqlda));
233     printf(" NUMBER OF COLUMN NAME = %d\n",
234             ucnda.sqlnzc);
235     for (i=0 ; i < ucnda.sqlnzc ; i++) {
236         printf(" [%d]), i);
237         printf(" DATA TYPE(%d)", PDSQLCOD(xsqlda,
238             i));
239         printf(" DATA LENGTH(%d)",
240             PDSQLLEN(xsqlda, i));
241         printf(" COLUMN NAME(%s)\n",
242             ucnda.SQLNAME[i].sqlnamec);
243     }
244 /*****
245 /*
246 /*****

```

```

243 Exit_Describe:
244     if(SQLCODE <0){
245         errmsg();
246         ErrFlg = -1;
247     }
248     return;
249 }
250
251
252 /*****
253  /*   Clear SQLDA                               */
254  *****/
255 static void ClearSqllda (
256 short num)
257 {
258     PDSQLN(xsqllda) = num;           14
259     PDSQLD(ssqllda) = num;          14
260     while(numzueng016.tif0){
261         PDSQLDATA(xsqllda, num) = NULL;    15
262         PDSQLIND(xsqllda, num) = NULL;    15
263         PDSQLDIM(xsqllda, num) = 0;       15
264         PDSQLXDIM(xsqllda, num) = 1;      15
265         PDSQLSYS(xsqllda, num) = 0;      15
266         PDSQLCOD(xsqllda, num) = 0;      15
267         PDSQLLEN(xsqllda, num) = 0;      15
268     }
269     return;
270 }
271
272
273 /*****
274  /*                                           */
275  /*   WARNING                               */
276  /*                                           */
277  *****/
278 static void errmsg()
279 {
280     int wsqlcode;
281
282     if(SQLCODE > 0){
283         printf(">>> warning\n");
284     }
285     if(SQLCODE <0){
286         printf(">>> error occurred\n");
287     }
288     wsqlcode = SQLCODE;
289     printf(">>> sqlcode = %d\n", SQLCODE);
290     printf(">>> sqlwarn = %c", SQLWARN0);

```

```

291     printf("%c", SQLWARN1);
292     printf("%c", SQLWARN2);
293     printf("%c", SQLWARN3);
294     printf("%c", SQLWARN4);
295     printf("%c", SQLWARN5);
296     printf("%c", SQLWARN6);
297     printf("%c", SQLWARN7);
298     printf("%c", SQLWARN8);
299     printf("%c", SQLWARN9);
300     printf("%c", SQLWARNA);
301     printf("%c", SQLWARNB);
302     printf("%c\n", SQLWARNC);
303
304     #if defined(HIUXWE2) || defined(WIN32)
305         printf(">>> message = %s\n", SQLERRMC);
306     #else
307         printf(">>> message = %Fs\n", SQLERRMC);
308     #endif
309     return;
310 }

```

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)

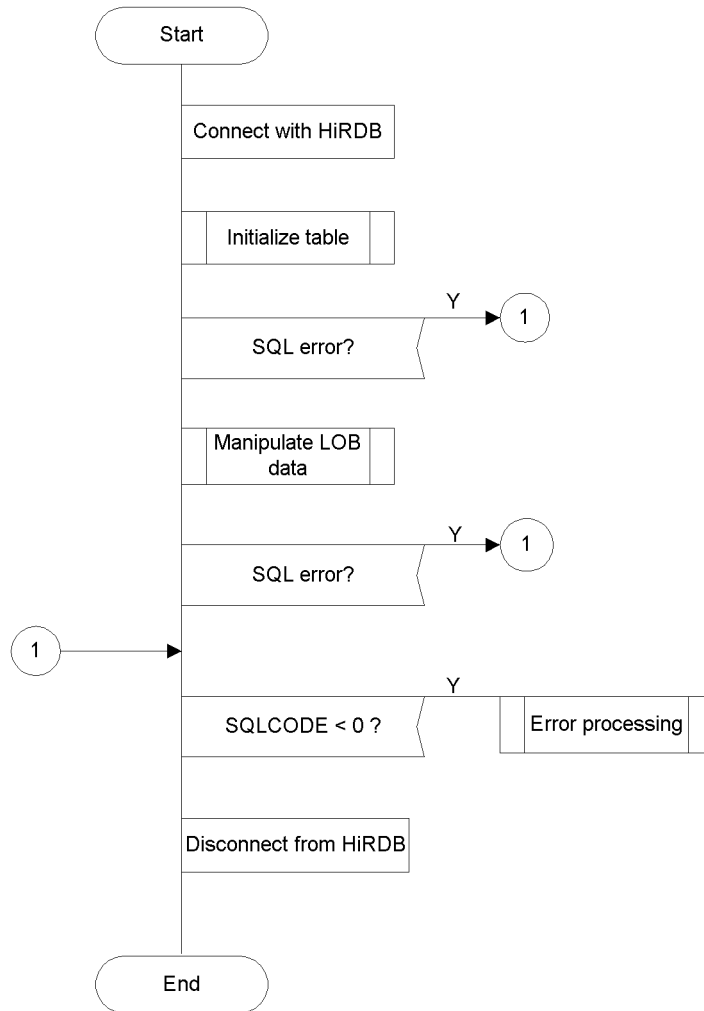


Figure 7-9: PAD chart for program example 3 (2/3)

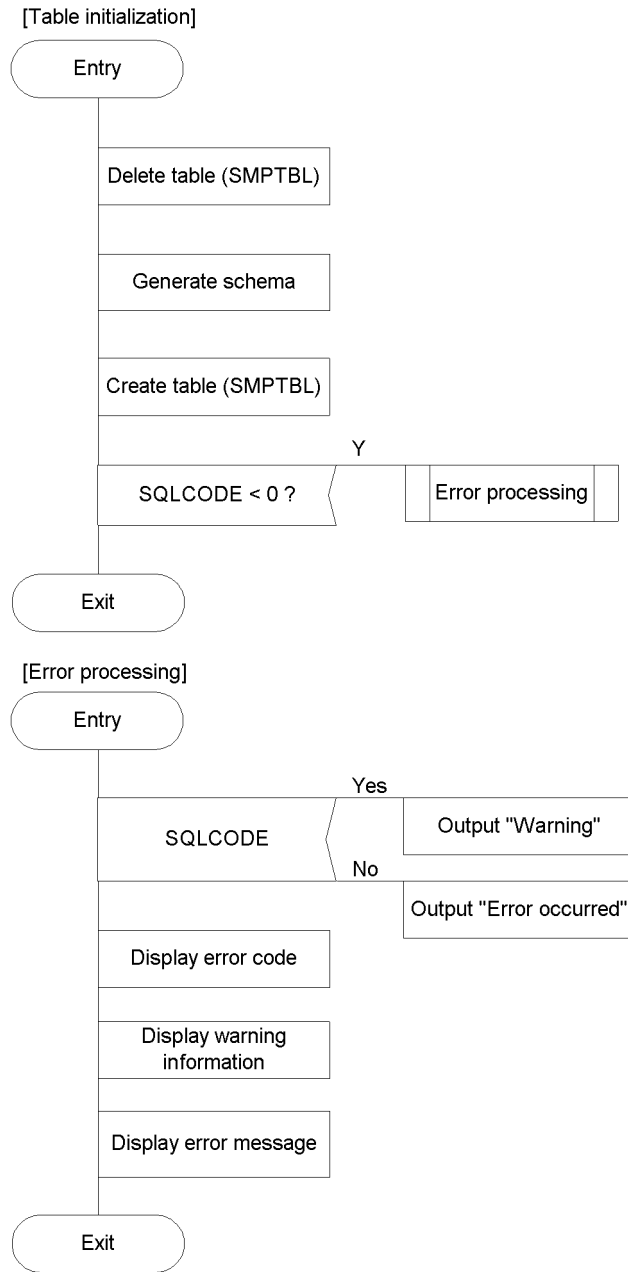
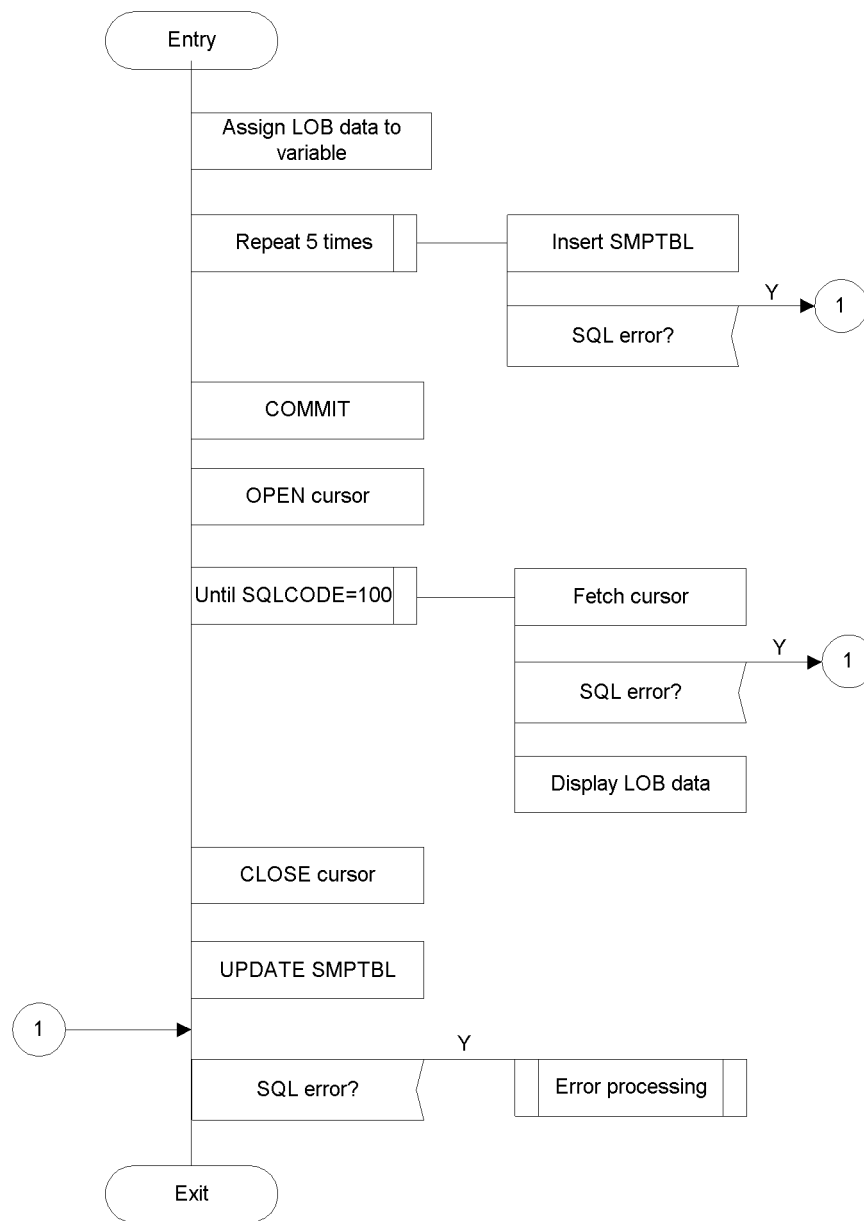


Figure 7-10: PAD chart for program example 3 (3/3)

[LOB data manipulation]



(b) Coding example

A coding example of program example 3 follows:

```

1  /*****
2  /*
3  /*  ALL RIGHTS RESERVED, COPYRIGHT (C) 1997,
      HITACHI, LTD.
4  /*  LICENSED MATERIAL OF HITACHI, LTD.
5  /*
6  /*****
7
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <stddef.h>
12 #include <ctype.h>
13 #include <string.h>
14
15 static void InitTable();
16 static void TestBlob();
17 static void warning();
18
19
20 /*****
21 /*  GLOBAL VARIABLE
22 /*****
23 short ErrFlg;
24
25 EXEC SQL BEGIN DECLARE SECTION;
26     short XSINT_IN;
27     short XSINT_OUT;
28     long  XINT_IN;
29     long  XINT_OUT;
30     SQL TYPE IS BLOB(16K) XBLOB_IN;
31     SQL TYPE IS BLOB(16K) XBLOB_OUT;
32 EXEC SQL END DECLARE SECTION;
33
34 /*
35 *  name = MAIN
36 *  func = SAMPLE
37 *  io   = argc : i :
38 *       argv : i :
39 *  return = 0,-1
40 *  note  = This program needs "RDUSER02" rdarea
           on Server.
41 *  date  = 98.04.24 by matsushiba
42 */
43 int main(

```

7. UAP Creation

```

44 int   argc,
45 char *argv[])
46 {
47     ErrFlg = 0;
48
49     /*****
50     /*
51     *****/
52 EXEC SQL
53     WHENEVER SQLERROR    goto ERR_EXIT;
54
55 EXEC SQL
56     WHENEVER SQLWARNING PERFORM :warning;
57
58 EXEC SQL CONNECT;                2
59
60
61     /*****
62     /*   INIT
63     *****/
64     InitTable();                3
65     if(ErrFlg <0){              3
66     goto ERREXIT;                3
67     }                             3
68
69     /*****
70     /*
71     *****/
72     TestBlob();                4
73
74     if(ErrFlg <0){              4
75     goto ERR_EXIT;                4
76     }                             4
77
78     /*****
79     /*
80     *****/
80 ERREXIT:
81     if(SQLCODE <0){
82         printf(":> ERROR HAPPENED!!\n");
83         warning();
84         ErrFlg = -1;
85     }
86
87 EXEC SQL
88     WHENEVER SQLERROR CONTINUE;
89 EXEC SQL
90     WHENEVER NOT FOUND CONTINUE;

```

```

91 EXEC SQL
92     WJEMEVER SQLWARNING CONTINUE;
93
94 EXEC SQL DISCONNECT;
95
96 return (ErrFlg);
97 }
98
99
100 /*****
101  /*  INIT
102  *****/
103 static void InitTable()
104 {
105
106 /*****
107  /*
108  *****/
109     EXEC SQL
110         WHENEVER SQLERROR CONTINUE;
111
112     EXEC SQL
113         DROP TABLE SMPTBL;
114
115     EXEC SQL
116         CREATE SCHEMA;
117
118     printf("## CREATE TABLE\n");
119
120     EXEC SQL
121         WHENEVER SQLERROR GOTO INIT_ERROR;
122
123     printf("## CREATE SMPTBL\n");
124     EXEC SQL
125         CREATE TABLE SMPTBL (CLM1      BLOB (30K) IN
126                                 RDUSER02,
127                                 CLM2      SMALLINT,
128                                 CLM3      INTEGER);
129
130     return;
131
132 INIT_ERROR:
133     warning();
134     ErrFlg = -1;
135     return;
136 }
137

```

7. UAP Creation

```

138
139 /*****
140 /*   TEST BLOB                               */
141 /*****
142 static void TestBlob()
143 {
144     short cnt;
145
146     EXEC SQL
147         WHENEVER SQL ERROR goto :ExitTestBlob;
148
149     EXEC SQL
150         WHENEVER SQLWARNING PERFORM :warning;
151
152 /*****
153 /*   INSERT                               */
154 /*****
155     memset(XBLOB_IN.XBLOB_IN_data,
156           0x55,
157           sizeof(XBLOB_IN.XBLOB_IN_data));
158     XBLOB_IN.XBLOB_IN_length =
159         sizeof(XBLOB_IN.XBLOB_IN_data);
160
161     printf("## INSERT \n");
162     for(cnt=1; cnt<5; cnt++){
163         XSINT_IN = cnt;
164         XINT_IN = 100+cnt;
165         EXEC SQL                               8
166             INSERT INTO SMPTBL                8
167             VALUES (:XBLOB_IN, :XSINT_IN,
168                     :XINT_IN);                8
169     }
170     EXEC SQL COMMIT;
171
172 /*****
173 /*   FETCH                               */
174 /*****
175     printf("## FETCH \n");
176
177     EXEC SQL                               9
178         DECLARE CUR_BLOB CURSOR FOR          9
179         SELECT * FROM SMPTBL;                9
180
181     EXEC SQL
182         OPEN CUR_BLOB;                        10
183
184     EXEC SQL
185         WHENEVER NOT FOUND GOTO FETCH_END;

```

```

184
185     for(;;){
186         memset (XBLOB_OUT.XBLOB_OUT_data,
187                0
188                sizeof(XBLOB_OUT.XBLOB_OUT_data));
189         XBLOB_OUT.XBLOB_OUT_length = 0;
190         EXEC SQL                                     //
191             FETCH CUR_BLOB INTO :XBLOB_OUT,         //
192                                 :XSINT_OUT,         //
193                                 :XINT_OUT;          //
194
195         printf("CLM1 XBLOB_length == %d\n",
196                XBLOB_OUT.XBLOB_OUT_length);
197         printf("CLM2 = %d\n", XSINT_OUT);
198         printf("CLM3 = %ld\n", XINT_OUT);
199     }
200 FETCH_END:
201     EXEC SQL
202         WHENEVER NOT FOUND CONTINUE;
203
204     EXEC SQL
205         CLOSE CUR_BLOB;                               //
206
207     /*****
208     /*  UPDATE                                     */
209     /*****
210     memset(XBLOB_IN.XBLOB_IN_data,
211            0x38,
212            sizeof(XBLOB_IN.XBLOB_IN_data));
213     XBLOB_IN.XBLOB_IN_length =
214         sizeof(XBLOB_IN.XBLOB_IN_data);
215
216     printf("## UPDATE\n");
217     EXEC SQL
218         UPDATE SMPTBL SET CLM1=:XBLOB_IN;             //
219
220     EXEC SQL COMMIT
221
222     /*****
223     /*
224     /*****
225 ExitTestBlob:
226     if(SQLCODE < 0){
227         warning();
228         ErrFlg = -1;
229     }
230     return;
231 )

```

```

231
232
233 /*****
234 /*    WARNING
235 /*****
236 static void warning()
237 {
238     if (SQLCODE < 0) {
239         printf(">>>ERROR\n");
240         printf(">>> sqlcode = %d\n", SQLCODE);
241 #if defined(HIUXWE2) || defined(WIN32)
242         printf(":> message = %s\n", SQLERRMC);
243 #else
244         printf(":> message = %Fs\n", SQLERRMC);
245 #endif
246     }
247     else{
248         printf(">>>WARNING\n");
249         printf(">>>sqlwarn = %c", SQLWARN0)
250         printf("%c", SQLWARN1);
251         printf("%c", SQLWARN2);
252         printf("%c", SQLWARN3);
253         printf("%c", SQLWARN4);
254         printf("%c", SQLWARN5);
255         printf("%c", SQLWARN6);
256         printf("%c", SQLWARN7);
257         printf("%c", SQLWARN8);
258         printf("%c", SQLWARN9);
259         printf("%c", SQLWARNA);
260         printf("%c", SQLWARNB);
261         printf("%c\n", SQLWARNC);
262     }
263     return
264 }

```

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 the Linux (IPF) and Linux (EM64T) versions, Windows Server 2003 (IPF), and Linux for AP8000 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
- Blanks 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)

1	6 7	8	11 12		72 73	80
Sequential number	area	Identifier area	Area A	Area B	Heading area	

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 the blank must be specified or can be specified; a description can span multiple lines.

To break a line where a blank 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 blank 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 *E. SQL Data Types and Data Descriptions*.
 - SIGN, 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 *E. 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.
 - UNICODE characters cannot be used in PICTURE and VALUE clauses. If they are used, the operation is not guaranteed.
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 *E. 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 *E. 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. Table 7-3 shows the divisions 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 8 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 8 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)

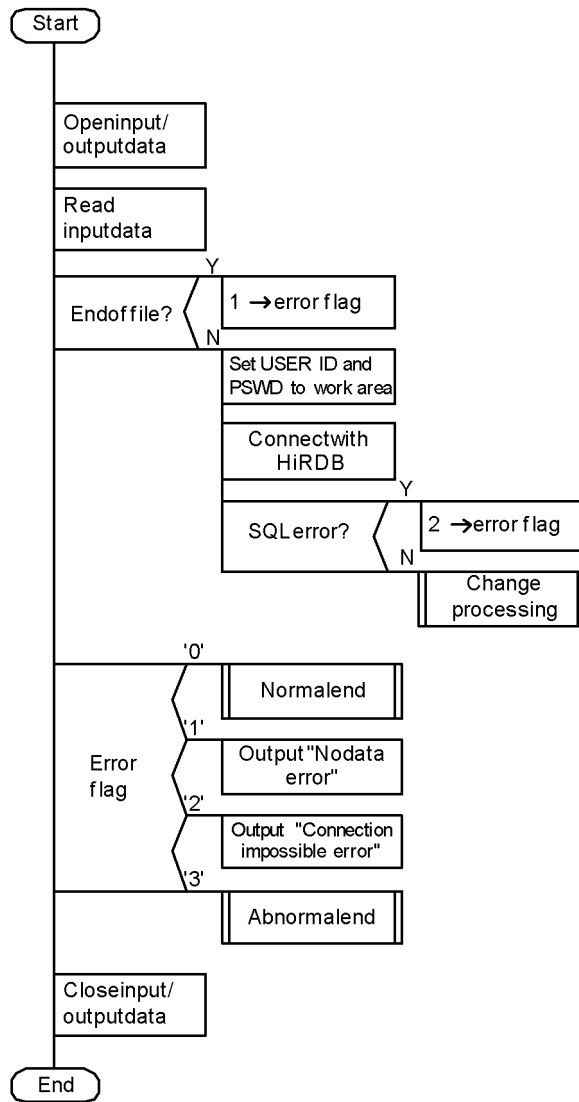


Figure 7-12: Flowchart of program example 4 (2/3)

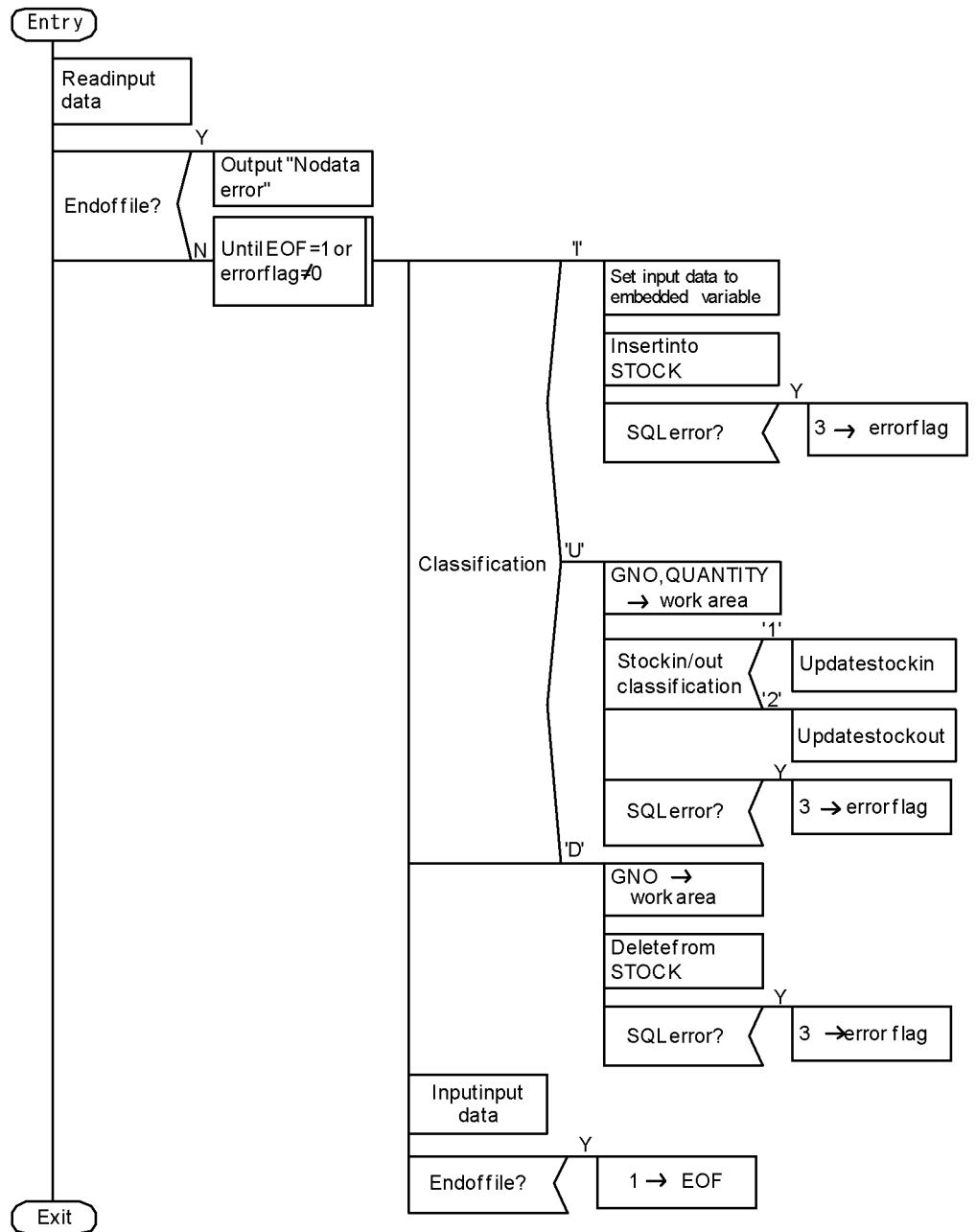
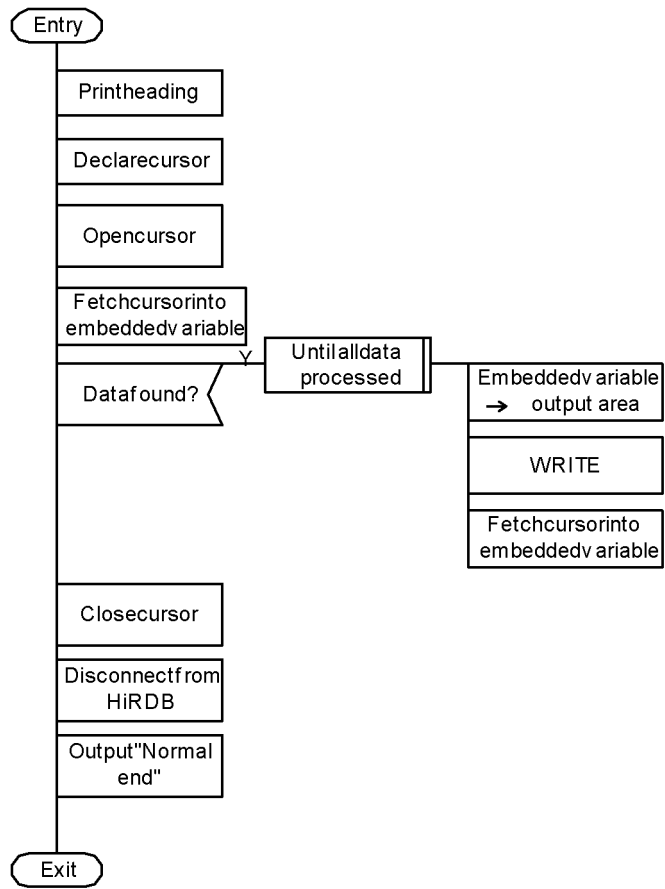
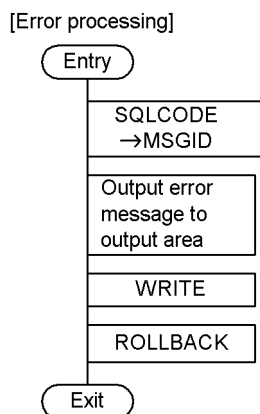


Figure 7-13: Flowchart of program example 4 (3/3)

[Normalprocessing]



**(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).
  
```

7. UAP Creation

```

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
00520          BEGIN DECLARE SECTION
00530      END-EXEC .
00540 77  XUSERID                   PIC X(30) .
00550 77  XPSWD                     PIC X(30) .
00560 77  XPCODE                     PIC X(4) VALUE '0000' .
00570 77  XPNAME                     PIC N(8) .
00580 77  XCOLOR                     PIC N(1) .
00590 77  XPRICE                     PIC S9(9) COMP .
00600 77  XSTOCK                     PIC S9(9) COMP .
00610* INDICATOR VARIABLE
00620 77  XIPCODE                     PIC S9(4) COMP VALUE 1040 .
00630 77  XIPNAME                     PIC S9(4) COMP VALUE 1050 .
00640 77  XICOLOR                     PIC S9(4) COMP VALUE 1060 .
00650 77  XIPRICE                     PIC S9(4) COMP VALUE 1070 .
00660 77  XISTOCK                     PIC S9(4) COMP VALUE 1080 .
00670*
00680*
00690      EXEC SQL
00700          END DECLARE SECTION
00710      END-EXEC .
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

```

7. UAP Creation

```

= ' .
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 'KFP1'.
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
01810          WHENEVER SQLERROR

```

(a) 2
(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

```

7. UAP Creation

```

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
02640          INSERT INTO STOCK(PCODE, PNAME, COLOR, PRICE,
SQUANTITY) 3
02650              VALUES (:XPCODE, :XPNAME, :XCOLOR, :XPRICE,
:XSTOCK) 3
02660      END-EXEC.
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'
02810              EXEC SQL
02820                  UPDATE STOCK SET SQUANTITY = SQUANTITY +
: XSTOCK          (a) 4
02830                      WHERE PCODE=:XPCODE
02840                      END-EXEC
02850                      WHEN '2'
02860                          EXEC SQL
02870                              UPDATE STOCK SET SQUANTITY = SQUANTITY -
: XSTOCK          (b) 4
02880                                  WHERE PCODE=:XPCODE
02890                                  END-EXEC
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
03050          DELETE FROM STOCK
03060              WHERE PCODE=:XPCODE
03070      END-EXEC.
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

```

7. UAP Creation

```

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 XPNAME 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

```

7. UAP Creation

```
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 (SQLEERROR) 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)

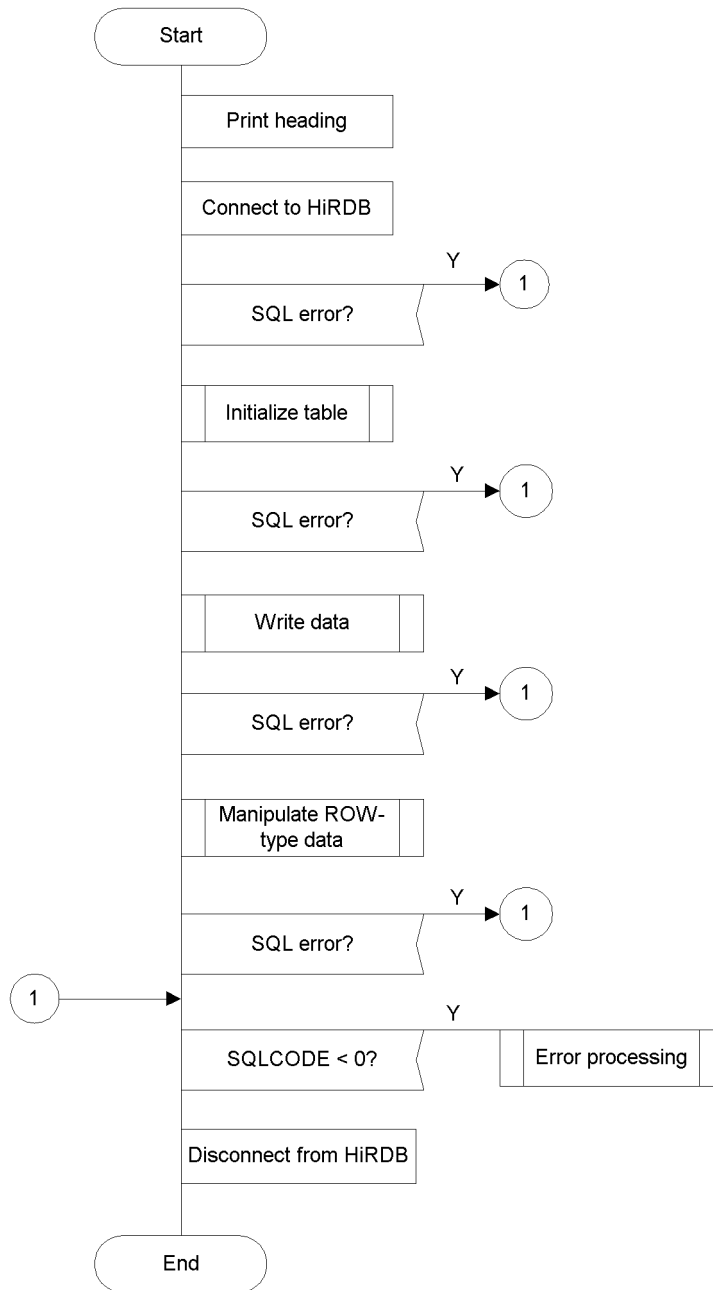
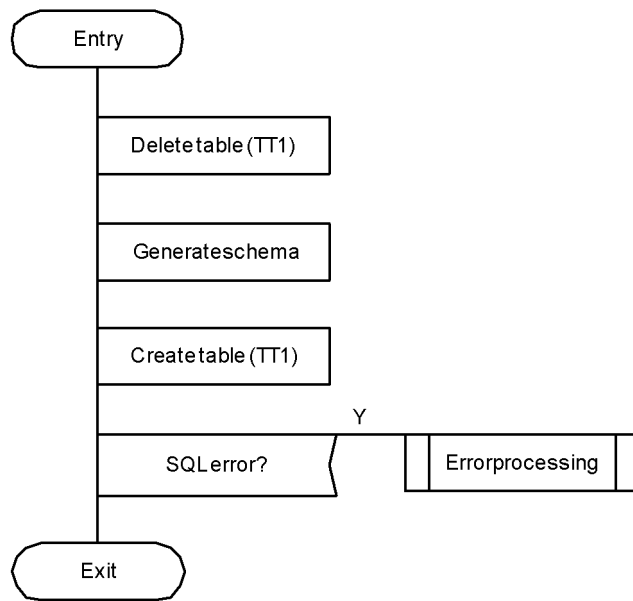


Figure 7-15: PAD chart for program example 5 (2/4)

[Tableinitialization]



[Errorprocessing]

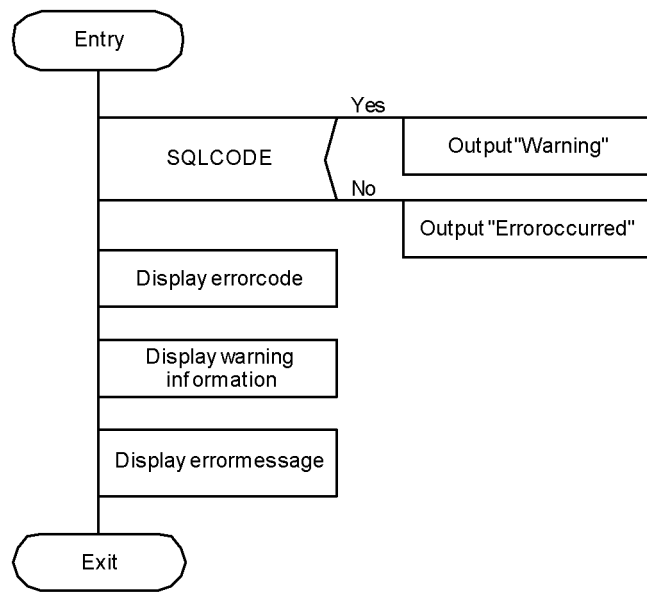


Figure 7-16: PAD chart for program example 5 (3/4)

[ROW-type data manipulation]

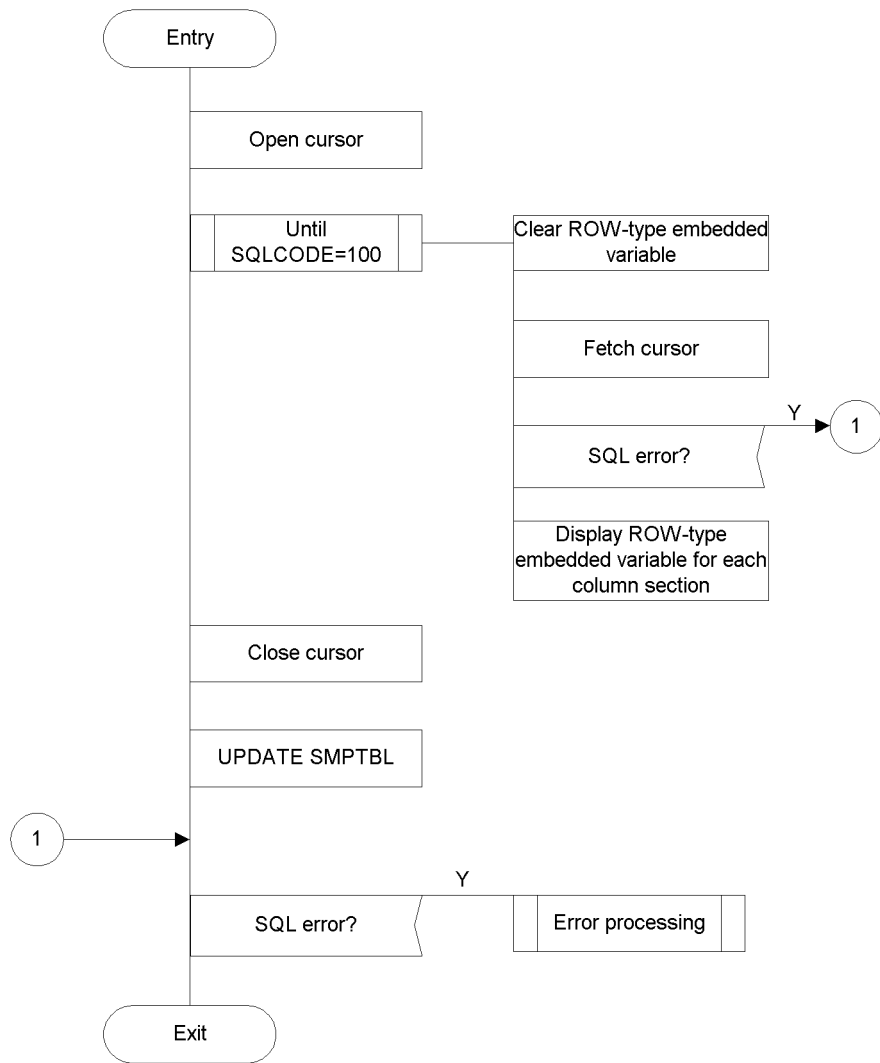
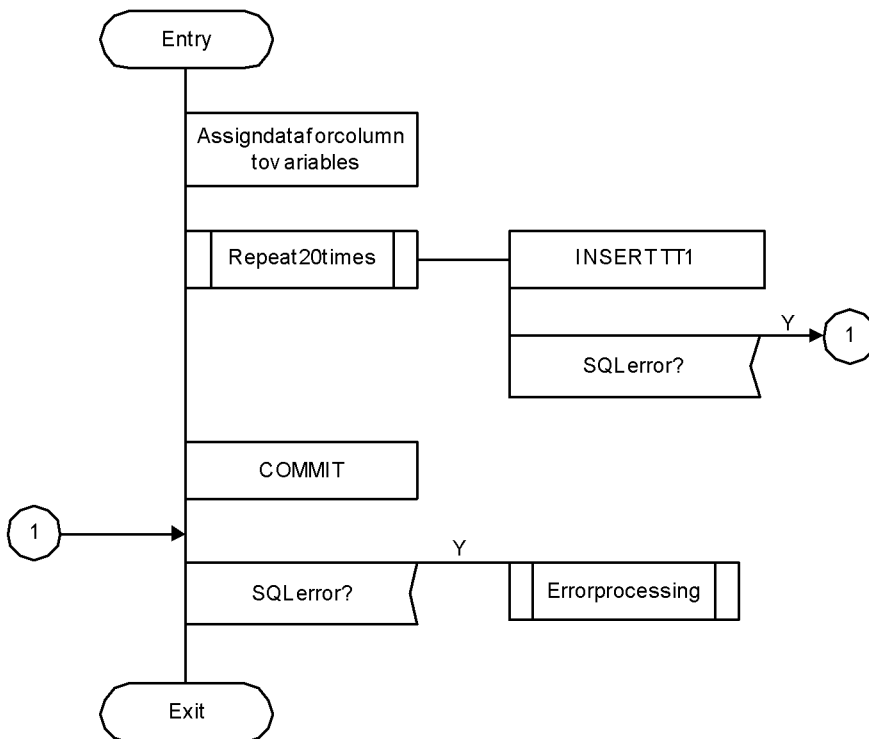


Figure 7-17: PAD chart for program example 5 (4/4)

[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.
    
```

7. UAP Creation

```

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).           /
00400     02 ROW-INT1      PIC S9(9) COMP.      /
00410     02 ROW-INT2      PIC S9(9) COMP.      /
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 *****

```

```

00650 MAIN SECTION
00660 CALL 'DISPLAY-TITLE'.
00670 MOVE ZERO TO ERRFLG.
00680
00690 *****
00700 * CONNECT
00710 *****
00720 EXEC SQL
00730 WHENEVER SQLERROR GOTO ERR-EXIT
00740 END-EXEC
00750
00760 DISPLAY '***** CONNECT '.
00770 EXEC SQL 2
00780 CONNECT 2
00790 END-EXEC. 2
00800 DISPLAY '***** CONNECT : END'.
00810
00820 *****
00830 * INIT
00840 *****
00850 DISPLAY '## TABLE WILL BE INITIALIZED'.
00860 CALL 'INIT-TABLE'.
00870 IF ERRFLG < ZERO
00880 GO TO ERR-EXIT
00890 END-IF
00900 DISPLAY '## IS NORMAL'.
00910
00920 *****
00930 * INSERT
00940 *****
00950 DISPLAY 'INSERT ## DATA'.
00960 CALL 'TEST-INSERT'.
00970 IF ERRFLG < ZERO
00980 GO TO ERR-EXIT
00990 END-IF
01000 DISPLAY '## IS NORMAL'.
01010
01020 *****
01030 * ROW
01040 *****
01050 DISPLAY '## ROW TYPE TEST WILL BE EXECUTED'.
01060 CALL 'TEST-ROW'.
01070 IF ERRFLG < ZERO
01080 GO TO ERR-EXIT
01090 END-IF
01100 DISPLAY '## IS NORMAL'.
01110
01120 *****

```

7. UAP Creation

```

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
01640          INSERT INTO TT1 (CLM1,
01650                          CLM2,
01660                          CLM3)
01670          VALUES (:IN-CHR1,
01680                  :IN-INT1,
01690                  :IN-INT2)
01700      END-EXEC
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'

```

7. UAP Creation

```

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.

```

7. UAP Creation

```

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
03180         END-EXEC                                     10
03190         DISPLAY '***** CREATE SCHEMA'.
03200         EXEC SQL                                     11
03210             CREATE SCHEMA
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.

```



```

03520     IF SQLCODE < ZERO THEN
03530         MOVE SQLCODE TO MSG-SQLCODE
03540         MOVE SQLERRMC TO MSG-SQLERRMC
03550         DISPLAY MSG-CODE
03560         DISPLAY MSG-MC
03570         MOVE -1 TO ERRFLG
03580     END-IF
03590     GOBACK.
03600
03610     *****
03620     *     WARNING
03630     *****
03640     INIT-TABLE-WARNING.
03650         DISPLAY 'WARNING'
03660         MOVE SQLCODE TO MSG-SQLCODE
03670         MOVE SQLERRMC TO MSG-SQLERRMC
03680         DISPLAY MSG-CODE
03690         DISPLAY MSG-MC.
03700     END PROGRAM INIT-TABLE.
03710
03720     *****
03730     *     DISPLAY
03740     *****
03750     IDENTIFICATION DIVISION.
03760     PROGRAM-ID. DISPLAY-TITLE.
03770     DATA DIVISION.
03780     WORKING-STORAGE SECTION.
03790     PROCEDURE DIVISION.
03800         DISPLAY '#####'
03810         DISPLAY '#                                     #'
03820         DISPLAY '# THIS PROGRAM IS A SAMPLE          #'
03830         DISPLAY '# PROGRAM FOR THE ROW-TYPE          #'
03840         DISPLAY '# INTERFACE                          #'
03850         DISPLAY '#####'
03860     END PROGRAM DISPLAY-TITLE.
03870     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 p \bar{d}

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
- Blanks 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*.

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 on UAP execution

8.1 UAP execution procedure

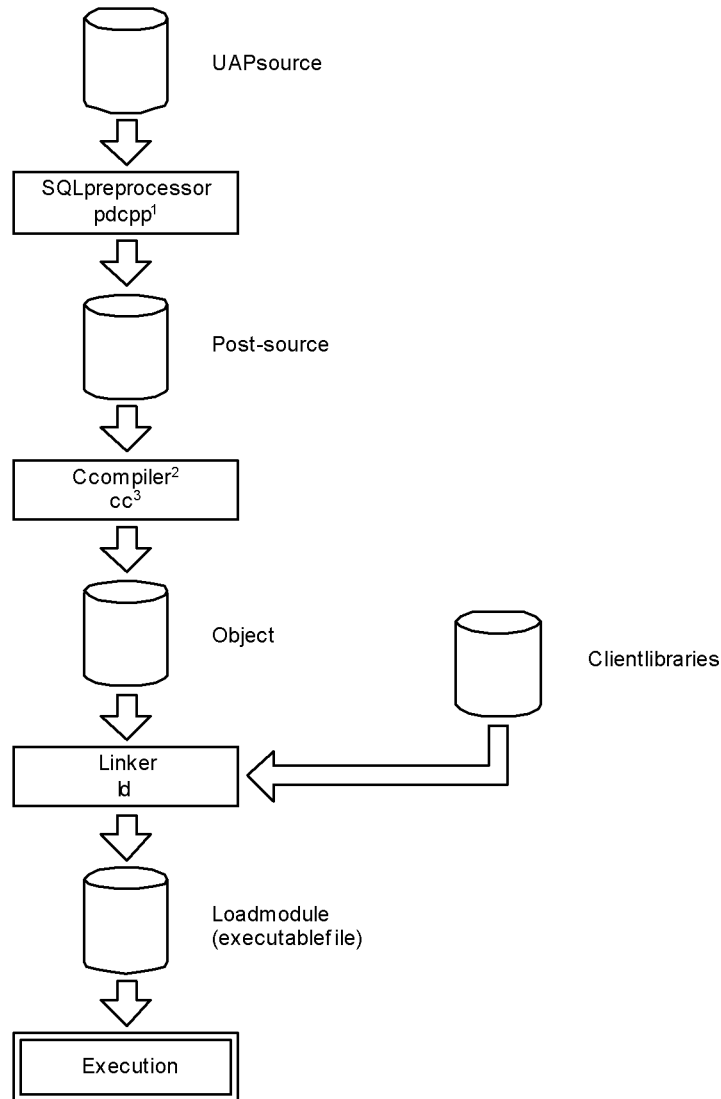
A UAP in which SQL statements are embedded cannot be executed directly. This section explains the procedure for executing such a UAP.

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.

Figure 8-1 shows the procedure for executing a UAP written in C.

Figure 8-1: Execution procedure for UAP written in C



¹For C++, the SQLpreprocessor is pdocc.

²For C++, the C++ compiler is used.

³For C++, the compiler is CC.

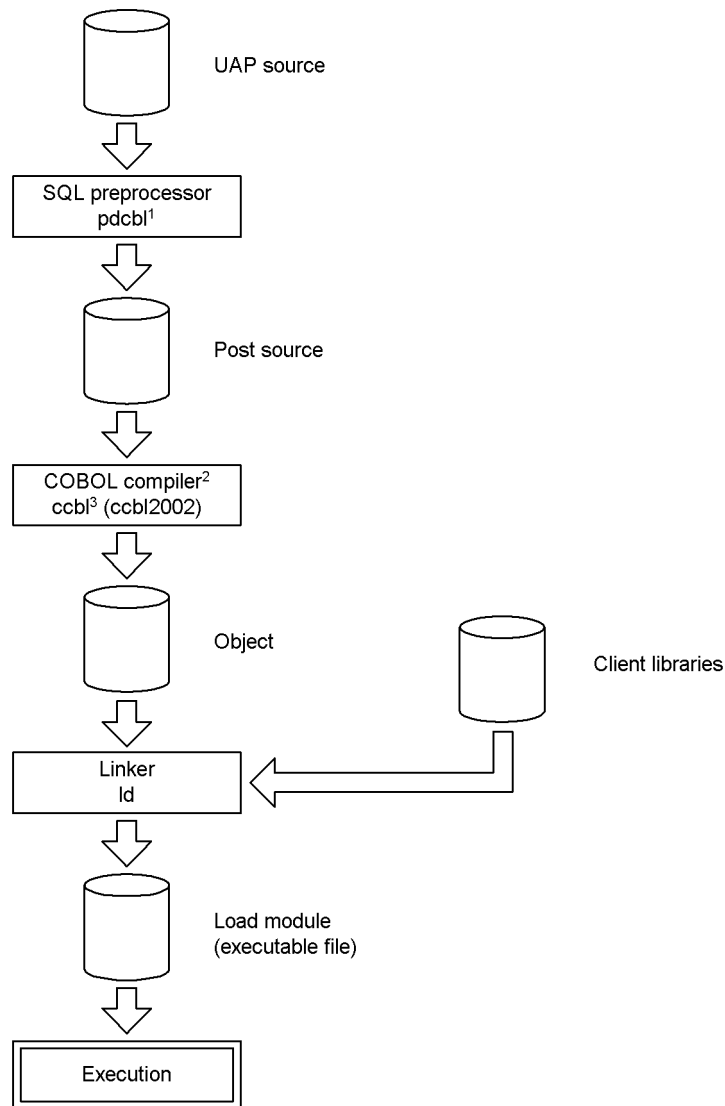
8.1.2 Executing a UAP written in COBOL

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.

Figure 8-2 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 5L 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. Table 8-1 shows the character codes that can be specified for 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 5L	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	Any value ²			
No specification	ja_JP.SJIS	ja	Ja_JP	ja_JP.eucJP, ia_JP, or Ja_JP.ujis

Legend:

—: Cannot be specified.

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 8 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. This specification is invalid when the distributed database facility is used and when remote database access is involved.

If the authorization identifier is omitted, the user identifier used during `CONNECT` is assumed.

options

Specifies, as necessary, the options shown in Table 8-2. 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
<code>-s</code>	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.
<code>-o file-name</code>	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 <code>.c</code> is used as the output filename.
<code>-A authorization-identifier</code>	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 <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code> , single-row <code>SELECT</code> , <code>OPEN (format 1)</code> , <code>CALL</code> , <code>LOCK</code> , or <code>PURGE TABLE</code> statement.
<code>-h64</code>	Specifies that a post source for 64-bit mode is to be created. If an embedded variable that uses the <code>long</code> type is declared, an error occurs.
<code>-P</code>	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: <ul style="list-style-type: none"> • 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	<p>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.</p> <ul style="list-style-type: none"> • 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}	<p>Specifies whether the cursor for PREPARE statement execution is to be closed automatically.</p> <p>y: Creates a post source that closes the cursor automatically. n: Creates a post source that does not close the cursor automatically.</p> <p>If this option is omitted, the preprocessor creates a post source according to the specification value in the PDRPCRCLS client environment definition.</p>
-Xv	<p>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.</p>

Preprocessing option	Description
<code>-E{1 2 3} ["option-character-string"]</code>	<p>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.</p> <p>-E1: Specifies that preprocessor declaration statements are to be validated.</p> <p>-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.</p> <p>-E3: Specifies that both -E1 and -E2 apply.</p> <p>"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 <code>-I</code> option specified in the C compiler. When specifying multiple options in <i>option-character-string</i>, use semicolons to separate the options. You can also specify any C compiler. When the <code>-E2</code> option is specified, this value ignored.</p> <p>When the <code>-E1</code> option is specified, the path name to the compiler must be specified in the <code>PATH</code> environment variable because the preprocessor calls the C compiler internally.</p>

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 <code>#define</code> .	N	Y	N	Y
Validate the header file that was included with <code>#include</code> .	N	Y	N	Y
Enable conditional compilation with <code>#if</code> , <code>#ifdef</code> , 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: If the C compiler is not 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"
```

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.

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).

Table 8-3 lists the return codes.

Table 8-3: SQL preprocessor return codes (for C programs in a UNIX environment)

Return code	Explanation
0	Normal termination

Return code	Explanation
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).

Table 8-4 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. This specification is invalid when the distributed database facility is used and when remote database access is involved.

If the authorization identifier is omitted, the user identifier used during `CONNECT` is assumed.

options

Specifies, as necessary, the options shown in Table 8-5. 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
<code>-s</code>	Specifies that only syntax is checked and that no post source is output; when this option is omitted, the post source is output.
<code>-o file-name</code>	Specifies a filename for the post source; when this option is omitted, the input filename with its file identifier changed to <code>.cbl</code> is used as the output filename. If the input file identifier is <code>.cbl</code> , this option must be specified to change the post source filename to an identifier other than <code>.cbl</code> .

8. Preparation for UAP Execution

Preprocessing option	Description
-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 <i>authorization-identifier</i>	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.
-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: <ul style="list-style-type: none"> • 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.
-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. <ul style="list-style-type: none"> • 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

Preprocessing option	Description
-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 PDRPCRCLS client environment definition.
-E2	Specifies that embedded variables are to be used without being declared in the embedded SQL declare section.

1. Examples of command specification in COBOL

Example 1

The UAP source filename is `sample` and no post source will be output.

```
pdcb1 sample.ecb -s
```

Example 2

The UAP source filename is `sample` and the filename of the post source to be output is `main`.

```
pdcb1 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).

Table 8-6 lists 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).

Table 8-7 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:

```
PDCPP.EXE 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. This specification is invalid when the distributed database facility is used. If the authorization identifier is omitted, the user identifier used during CONNECT is assumed.

options

Specifies, as necessary, the options shown in Table 8-8. 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
<code>/S</code>	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 <code>/xp</code> is also specified.
<code>/O file-name</code>	Specifies a filename for the post source that is output; when this option is omitted, the input filename with its file identifier changed to <code>.C</code> is used as the output filename.
<code>/A authorization-identifier</code>	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 <code>INSERT</code> , <code>UPDATE</code> , <code>DELETE</code> , single-row <code>SELECT</code> , <code>OPEN (format 1)</code> , <code>CALL</code> , <code>LOCK</code> , or <code>PURGE TABLE</code> statement.
<code>/h64</code>	Specifies that a post source for 64-bit mode is to be created. If an embedded variable that uses the <code>long</code> type is declared, an error occurs.
<code>/Xe{y n}</code>	Specifies whether the cursor for <code>PREPARE</code> statement execution is to be closed automatically. <code>y</code> : Creates a post source that closes the cursor automatically. <code>n</code> : 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 <code>PDRPCRCLS</code> client environment definition.
<code>/Xv</code>	Specifies that <code>VARCHAR</code> - and <code>BINARY</code> -type structures are to be analyzed as normal structures when the <code>/E2</code> option is specified. To declare <code>VARCHAR</code> - and <code>BINARY</code> -type embedded variables, use the <code>SQL TYPE IS</code> -type variable declaration. This option must be specified together with the <code>/E2</code> option. Do not specify this option if the UAP uses macros for repetition columns.
<code>/XA</code>	Specifies that an X/Open-compliant API is to be used to create the UAP.

Preprocessing option	Description
/Xo	<p>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.</p> <ul style="list-style-type: none"> • 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.
/E{1 2 3} ["option-character-string"]	<p>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.</p> <p>/E1: Specifies that preprocessor declaration statements are to be validated.</p> <p>/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.</p> <p>/E3: Specifies that both /E1 and /E2 apply.</p> <p><i>"option-character-string"</i>: 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 <i>option-character-string</i>, use semicolons to separate the options. You can also specify any C compiler. When the /E2 option is specified, this value ignored.</p> <p>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.</p>
/Xp	<p>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.</p>

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 <code>#define</code> .	N	Y	N	Y
Validate the header file that was included with <code>#include</code> .	N	Y	N	Y
Enable conditional compilation with <code>#if</code> , <code>#ifdef</code> , 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.

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.

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`.

Table 8-9 lists 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)

Return code	Explanation
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).

Table 8-10 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)
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 `PDOCBL.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. This specification is invalid when the distributed database facility is used. If the authorization identifier is omitted, the user identifier used during CONNECT is assumed.

options

Specifies, as necessary, the options shown in Table 8-11. 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 <i>file-name</i>	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 <u>. CBL</u> (for COBOL language) or <u>. OCB</u> (for OOCOBOL language) and is used as the output filename. If the input file identifier is <u>. CBL</u> (for COBOL language) or <u>. OCB</u> (for OOCOBOL language), this option must be specified to change the post source filename to an identifier other than <u>. CBL</u> (for COBOL language) or <u>. OCB</u> (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.

Preprocessing option	Description
/A <i>authorization-identifier</i>	<p>Specifies that the default authorization identifier, which is used when no authorization identifier is specified in a static SQL statement, is to be changed.</p> <p>A static SQL statement refers to the INSERT, UPDATE, DELETE, single-row SELECT, OPEN (format 1), CALL, LOCK, or PURGE TABLE statement.</p>
/XD	<p>Specifies that a DLL is to be created.</p> <p>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.</p>
/Xe{y n}	<p>Specifies whether the cursor for PREPARE statement execution is to be closed automatically.</p> <p>y: Creates a post source that closes the cursor automatically.</p> <p>n: Creates a post source that does not close the cursor automatically.</p> <p>If this option is omitted, the preprocessor creates a post source according to the specification value in the PDRPCRCLS client environment definition.</p>
/XAD	<p>Specifies that a UAP that used an X/Open-compliant API is to be created as a DLL.</p>
/XA	<p>Specifies that the UAP is to be created by using an X/Open-compliant API.</p>

Preprocessing option	Description
/Xo	<p>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.</p> <ul style="list-style-type: none"> • 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	<p>Specifies that embedded variables are to be used without being declared in the embedded SQL declare section.</p>
/Xp	<p>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.</p>

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`.

Table 8-12 lists 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).

Table 8-13 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];
```

8. Preparation for UAP Execution

```
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 :xpcod, :xpnam, :xcol, :xstock;
        printf("    %4s    %-16s %2s %8d %8d\n",
            xpcod, xpnam, xcol, 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 FETCH CUR1 INTO :xprice, :xstock, :xpname;
```

(2) Rules

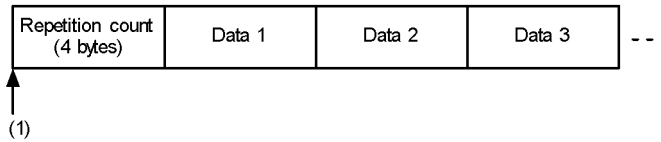
1. When specifying a pointer variable, add a colon before the variable name in the SQL statement. An asterisk cannot be used.
2. The size of the value to be referenced becomes the size of the data type specified in the declaration. However, this does not apply to the fixed-length character string type (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

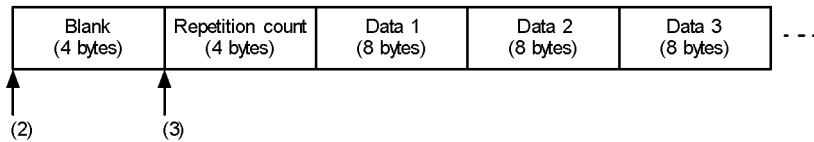
(1), which coincides with a word boundary, must be specified in the pointer.



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*.

- 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).

- 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

Table 8-14 shows whether or not pointers, structures, and pointer qualifiers can be used

when you specify the preprocessor's /E2 or /E3 option (-E2 or -E3 option in the UNIX version).

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

SQL statement that specifies embedded variable or indicator variable		Pointer	Structure	Structure qualifier
	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
	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 CONNECTION statement	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.

8. Preparation for UAP Execution

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	Used	For a single thread: libzclts.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 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
HP-UX 11i V2 (IPF)	Used	For a single thread: libzclts.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	—

8. Preparation for UAP Execution

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file
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
AIX 5L	Used	For a single thread: libzclts.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 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 (IPF)	Used	For a single thread: libzclts64.so For multiple threads (POSIX threads): libzcltk64.so	—
	Not used	libzclt64.so	—

Platform	Multi-connection facility	Library name	
		Shared library file	Archive file
Linux (EM64T)	Used	For a single thread: libzclts.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	—
Windows	Used	PDCLTM32.LIB	—
	Not used	CLTDLL.LIB	—
Windows Server 2003 (IPF)	Used	PDCLTM64.LIB	—
	Not used	PDCLTM64.LIB	—
Windows (x64)	Used	For 32-bit mode: PDCLTM32.LIB For 64-bit mode: PDCLTM64.LIB	—
	Not used	For 32-bit mode: PDCLTM32.LIB For 64-bit mode: PDCLTM64.LIB	—

Legend:

— : Not applicable

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	Dynamic registration	For a single thread: libzcltx.sl libzcltxs.sl (for OTS) For multiple threads (kernel threads): libzcltxk.sl	For a single thread: libcltxa.a libzcltxas.a For multiple threads (kernel threads): libcltxak.a
	Dynamic or static registration	For a single thread: libzclty.sl libzcltys.sl (for OTS) For multiple threads (kernel threads): libzcltyk.sl	For a single thread: libcltya.a libzcltyas.a For multiple threads (kernel thread): libcltyak.a
HP-UX 11i V2 (IPF)	Dynamic registration	For a single thread: libzcltx.so libzcltxs.so (for OTS) For multiple threads (kernel threads): libzcltxk.so For 64-bit mode multiple threads (kernel threads): libzcltxk64.so	—
	Dynamic registration or static registration	For a single thread: libzclty.so libzcltys.so (for OTS) For a 64-bit mode single thread (kernel threads): libzclty64.so libzcltys.so (for OTS) For multiple threads (kernel threads): libzcltyk.so 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 OTS) For multiple threads (Solaris threads): libzcltxk.so	For a single thread: libcltxa.a libcltxas.a (for OTS) For multiple threads (Solaris threads): libcltxak.a
	Dynamic or static registration	For a single thread: libzclty.so libzcltys.so (for OTS) For multiple threads (Solaris threads): libzcltyk.so	For a single thread: libcltya.a libcltyas.a (for OTS) For multiple threads (Solaris threads): libcltyak.a
AIX 5L	Dynamic registration	For a single thread: libzcltx.a libzcltxs.a (for OTS) For multiple threads (POSIX threads): libzcltxk.a	For a single thread: libcltxa.a libcltxas.a (for OTS) For multiple threads (POSIX threads): libcltxak.a
	Dynamic or static registration	libzclty.a libzcltys.a (for OTS)	libcltya.a libcltyas.a (for OTS)
Linux	Dynamic registration	libzcltx.so libzcltxs.so (for OTS)	libcltxa.a libcltxas.a (for OTS)
	Dynamic or static registration	libzclty.so libzcltys.so (for OTS)	libcltya.a libcltyas.a (for OTS)
Linux (IPF)	Dynamic registration	For a 64-bit mode single thread (kernel threads): libzcltx64.so libzcltxs64.so (for OTS) For 64-bit mode multiple threads (POSIX threads): libzcltxk64.so	—
	Dynamic registration or static registration	For a 64-bit mode single thread (kernel threads): libzclty64.so libzcltys64.so For 64-bit mode multiple threads (POSIX threads): libzcltyk64.so	—

8. Preparation for UAP Execution

Platform	Transaction registration method	Library name	
		Shared library file	Archive file
Linux (EM64T)	Dynamic registration	For a single thread: libzcltx.so libzcltxs.so (for OTS) For multiple threads (POSIX threads): libzcltxk.so	—
	Dynamic registration or static registration	For a single thread: libzclty.so libzcltys.so (for OTS) For multiple threads (POSIX threads): libzcltyk.so	—
Windows	Dynamic registration	—	—
	Dynamic or static registration	For a single thread: PDCLTX32.LIB PDCLTXS.LIB (for OTS) For multiple threads: PDCLTXM.LIB	—
Windows Server 2003 (IPF)	Dynamic registration	—	—
	Dynamic registration or static registration	For a single thread: PDCLTX64.LIB PDCLTXS64.LIB (for OTS) 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 8 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. This option can be specified only when the shared library is used.

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 5L

```
xlc -q64 -I/HiRDB/include sample.c -Wl,-L/HiRDB/client/lib -lzclt64
```

Linux (IPF)

```
gcc -I /HiRDB/include sample.c -L/HiRDB/client/lib -lzclt64
```

Multiple threads in Linux (IPF)

8. Preparation for UAP Execution

```
gcc -I /HiRDB/include sample.c -D_REENTRANT -L/HiRDB/client/lib -lzcltk64
```

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 5L

```
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 5L

```
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 5L

```
xlc -q64 -I/HiRDB/include sample.C -Wl,-L/HiRDB/client/lib,-lclt64
```

(2) COBOL

Post-source programs in COBOL must be compiled 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. This option can be specified only when the shared library is being used.

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 `-K1` 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) version of COBOL2002, `ccbl` becomes `ccbl2002`.

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
```

(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 version 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.)

Table 8-17 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 Server 2003 (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: \HiRDB\INCLUDE
- Linkage library: \HiRDB\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 `-K1`, `-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	<u>\HiRDB</u> \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	<u>\HiRDB</u> \lib\cltdll.lib

Note

The directory where HiRDB is installed is underlined.

COBOL85 has an option to be set for **Compilation Environment**. Table 8-20 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	<u>\HiRDB</u> \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:

Platform	32-bit mode (Win32)	64-bit mode (x64)
Alignment of structure members	Default (8 bytes)	Default (8 bytes)
Runtime library to be used	Multi-thread DLL	Multi-thread DLL
Include file directory	\ <u>HiRDB</u> \INCLUDE	\ <u>HiRDB</u> \INCLUDE
Linkage library*	\ <u>HiRDB</u> \LIB\PDCLTM32.LIB	\ <u>HiRDB</u> \LIB\PDCLTM64.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 the following UAPs cannot be created for the 64-bit mode:

- UAPs that use the XA interface
- UAPs written in COBOL or OOCOBOL.

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 5L, and Linux, link the `libcltk.a` and `libzcltk.sl` libraries. Table 8-21 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	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
HP-UX 11i V2 (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 5L	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 (IPF)	POSIX thread	libzcltk64.so	—
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 5L

- 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 5L

- 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 5L

- 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`. The multi-connection facility cannot be used for a UAP that uses an X/Open-base API in OLTP.

■ **C**

This explanation assumes that Microsoft Visual C++ Version 4.2 is used. Select **Set** from the **Project** menu, and set the individual items. Table 8-22 shows the items that can be set with **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	\ <u>HIRDE</u> \INCLUDE

Item	Category	Category setting	Setting value
Linker	Input	Library	\ <u>HIRDB</u> \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*.

Table 8-23 shows 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 <code>-Mt</code> item.
		Environment variable setting	<code>CBLLIB=C:\<u>HIRDB</u>\INCLUDE</code>
Linker	Linker Option Setting	Import/user-specified library	<code>C:\<u>HIRDB</u>\LIB\PDCLTM32</code>

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`

8. Preparation for UAP Execution

- -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 on UAP execution

8.4.1 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 Windows versions.

(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 -I$DCDIR/include -I/HiRDB/include filename.c
/usr/bin/cc -o UAP-executable-form-filename UAP-filename.o
$DCDIR/spool/trnmcmd/userobj/control-object-filename.o
-L/HiRDB/client/lib -lzclty -L$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. The `-Wl,+s` option must be specified if different HiRDB distribution library directories are used for UAP creation and for 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 8 Installation and Design Guide*.

- To create a UAP that uses an X/Open-compatible API in AIX 5L, 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 -I$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

- The directory for installing HiRDB is underlined.
- The archive library provided by HiRDB (`-lcltxa`) must be specified before the library provided by OpenTP1 (`-lbetran`).
- 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 8 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. The `-Wl, +s` option must be specified if different HiRDB distribution library directories are used for UAP creation and for 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 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.

-lzcltx (dynamic registration)

-lzclty (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 8 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
      $DCDIR/spool/trnrmcmd/userobj/control.o
      -L/HiRDB/client/lib -lzclty -L$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 OpenTPI (-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 8 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

UAP compilation and linkage when TPBroker is used are explained here. The UAP is assumed to use the multi-thread XA interface. For details about compilation and linkage using TPBroker, 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 is used to create a UAP that accesses HiRDB, it is necessary to create a transaction control object file with the TPBroker `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=1995061filename.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. The `-Wl,+s` option must be specified if different HiRDB distribution library directories are used for UAP creation and for 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 8 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 $DCDIR/lib/ee_main.o control.o
-brt1 -bdynamic -L/HiRDB/lib -L/HiRDB/client/lib -L$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 `eetrmkobj` command is used for this purpose.

The following is specified to create an object file for resource manager linking:

```
eetrmkobj -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.

```
eetrmkobj -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*.

8.4.2 Creating UAPs that support the 64-bit mode

This section explains how to use a HiRDB client to create UAPs that support the 64-bit mode.

(1) Languages and facilities that UAPs that support the 64-bit mode can use

(a) Languages

UAPs that support the 64-bit mode can be created in C or C++. COBOL and OOCOBOL cannot be used.

(b) Facilities

UAPs that support the 64-bit mode cannot use the XA interface. However, they can basically use all other facilities. In 64-bit mode, the multi-connection facility provides real threads instead of pseudo threads.

(2) Converting a HiRDB client from 32-bit mode to 64-bit mode

Before a HiRDB client can be converted from 32-bit mode to 64-bit mode, the HiRDB client must be upgraded to a 64-bit mode version. (Install the 64-bit mode HiRDB client and set up the client environment.) For details about how to set up the client environment, see 6. *Client Environment Setup*.

When the 64-bit mode HiRDB client is installed, files for 64-bit mode are created. For details about the files that are created when the 64-bit mode HiRDB client is installed, see 6.4 *Organization of directories and files for a HiRDB client*.

After the client environment setup is completed, convert the UAP to 64-bit mode support according to the following procedure.

Procedure

1. If `long` type is used in the declaration of an embedded variable, change the location to `int` type
2. Execute UAP preprocessing. When executing UAP preprocessing, specify the `-h64` option for creating a 64-bit mode post source.
3. Execute UAP compilation. When executing UAP compilation, specify the option for creating 64-bit mode objects.
4. Execute UAP linkage. When executing UAP linkage, specify a 64-bit mode client library as the client library to be linked.

Note

For details about UAP preprocessing, compilation, and linkage, see 8.2 *Preprocessing* and 8.3 *Compiling and linking*.

8.4.3 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 8 SQL Reference* manual.

Table 8-24 shows UAP compatibility from XDM/RD or UNIFY2000.

Table 8-24: 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

8.4.4 Notes on UAP execution

Notes on UAP execution are described below.

- To execute a UAP, specify the LANG environment variable, or the PDLANG or PDCLTLANG client environment definition as appropriate to the character code classification of the HiRDB server (the character code classification specified by the pdsetup command for UNIX or the pdntenv command for Windows). If the HiRDB server and HiRDB client use different character code classifications, an error occurs when the UAP executes. Table 8-25 lists the LANG and PDLANG settings for each platform.

Table 8-25: LANG and PDLANG settings for each platform

Character code classification ^{#1}		HP-UX	Solaris	AIX 5L	Linux
lang-c	LANG	C	C	C	C
	PDLANG	—	—	—	—
sjis	LANG	ja_JP.SJIS	ja_JP.PCK	Ja_JP	Optional ^{#2}
	PDLANG	—	—	—	SJIS
ujis	LANG	ja_JP.eucJP	ja	ja_JP	ja_JP.eucJP ^{#3}
	PDLANG	—	—	—	—

Character code classification ^{#1}		HP-UX	Solaris	AIX 5L	Linux
chinese	LANG	chinese-s	Optional ^{#2}	Optional ^{#2}	Optional ^{#2}
	PDLANG	CHINESE	CHINESE	CHINESE	CHINESE
utf-8	LANG	Optional ^{#2}	Optional ^{#2}	Optional ^{#2}	Optional ^{#2}
	PDLANG	UTF-8	UTF-8	UTF-8	UTF-8

Legend:

—: Not applicable

#1

Character code classification of the HiRDB server, as specified in `pdsetup` or `pdntenv`.

#2

If a character code supported by the platform being used is supported by the UAP, specify that character code in the `LANG` environment variable. If the platform does not support one of these character codes, specify `C`.

#3

`ja_JP` is handled in the same way as `ja_JP.eucJP`.

- When executing a UAP, add `$PDDIR/client/lib` to `SHLIB_PATH`. When working with a different platform, replace `SHLIB_PATH` with the corresponding environment variable of that platform.
- If the HiRDB system has a recovery-unnecessary front-end server and a UAP that uses the X/Open XA interface to connect to the recovery-unnecessary front-end server, SQL statements cannot be executed from that UAP. In this case, specify the `PDFESHOST` and `PDSERVICEGRP` client environment definitions and connect to a front-end server that is not a recovery-unnecessary front-end server.

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 Java stored routine creation to execution
- 9.3 Sample programs of Java stored routine
- 9.4 Notes about Java program creation
- 9.5 Notes about testing and debugging
- 9.6 Notes about JAR file creation

9.1 Overview

Java stored procedures and *Java stored functions* are the stored procedures and stored functions coded in Java.

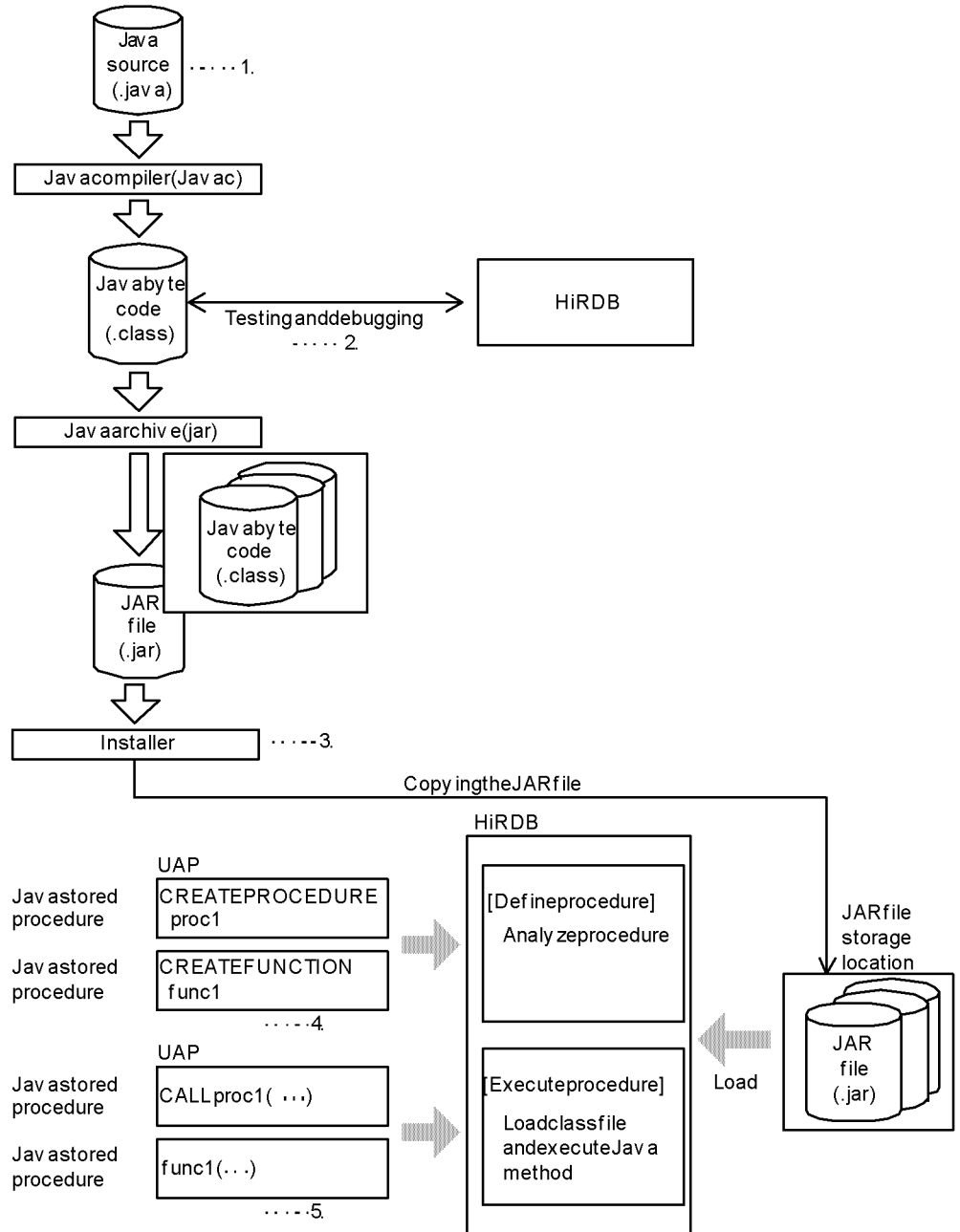
In this chapter, Java stored procedures and Java stored functions are collectively referred to as *Java stored routines*.

Java stored routines cannot be used in 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 8 System Operation Guide*.

Procedures for an SQL stored procedure or an SQL stored function are coded during definition. Procedures for a Java stored routine are not coded during definition; instead, the Java program registered at the server is specified. The specified Java program then functions as a stored procedure or stored function.

Figure 9-1 shows the procedure from Java stored routine creation to execution.

Figure 9-1: Procedure from Java stored routine creation to execution



Explanation

1. Code a Java stored routine. For details, see *9.2.1 Coding a Java stored routine*.
2. Test and debug the Java stored routine as a client AP. For details, see *9.2.1 Coding a Java stored routine*.
3. Register the JAR file in HiRDB. For details, see *9.2.2 Registering the JAR file in HiRDB*.
4. Define the Java stored routine. For details, see *9.2.3 Defining the Java stored routine*.
5. Execute the Java stored routine. For details, see *9.2.4 Executing the Java stored routine*.

■ **Features of Java stored routines**

1. **There is no overhead between the server and a client.**

Java stored routines are processed at the server in the same manner as for SQL stored procedures and SQL stored functions. Therefore, 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 SQL.

3. **Java stored routines are supported by different types of DBMS.**

Java is a programming language independent of platform. Therefore, a program created in Java can be run in different types of DBMS that support 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. On the other hand, a Java stored routine can be debugged at the client, including database accesses, as long as the Java debugger is installed at the client.

■ **Preparations for Java stored routine execution**

To execute a Java stored routine, you need to install the JDBC driver beforehand. For details about installing the JDBC driver, see *16.1 Installation and environment setup*.

9.2 Procedure from Java stored routine creation to execution

This section describes the procedure for creating and executing a Java stored routine:

1. Coding a Java stored routine
2. Registering the JAR file in HiRDB
3. Defining the Java stored routine
4. Executing the Java stored routine

9.2.1 Coding a Java stored routine

To code a Java stored routine, use the following procedure:

1. Coding a Java program (creating a Java file)
2. Compiling (creating a Class file)
3. Testing and debugging
4. Archiving in the JAR format (creating the JAR file)

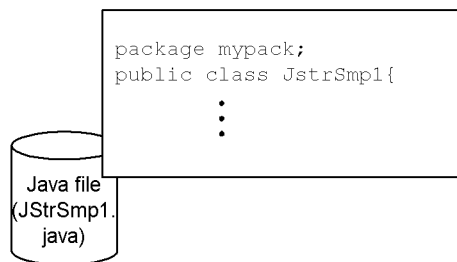
(1) Coding a Java program (creating a Java file)

Code the program that is to be registered as a Java stored routine.

For notes about Java program coding, see *9.4 Notes about Java program creation*.

Figure 9-2 shows an example of Java program coding.

Figure 9-2: Example of Java program coding



(2) Compiling (creating a Class file)

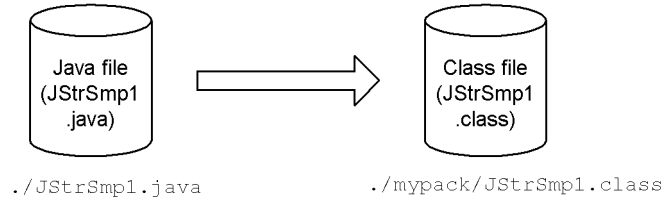
Create a Class file from the Java file using the `javac` command.

Figure 9-3 shows an example of compilation.

Figure 9-3: Example of compilation

Command entry

```
% javac -d ./ JStrSmp1.java
```



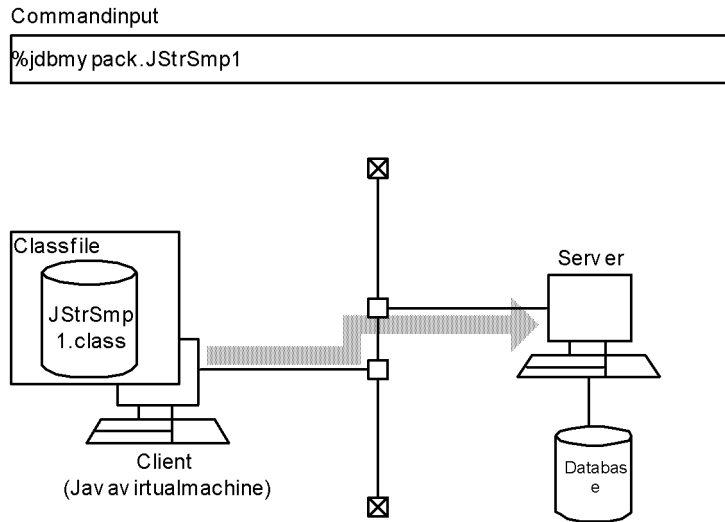
Explanation

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*. Figure 9-4 shows an overview of testing and debugging.

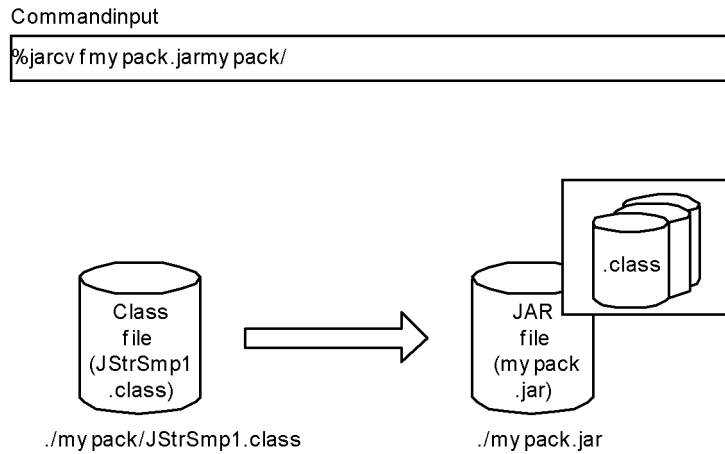
Figure 9-4: Overview of testing and debugging



(4) Archiving in the 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*. Figure 9-5 shows an example of archiving in the JAR format.

Figure 9-5: Example of archiving in the JAR format



9.2.2 Registering the JAR file in HiRDB

Register (copy) the created JAR file into 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`.

For details about `INSTALL JAR`, see the *HiRDB Version 8 SQL Reference* manual.

- **Executing the `pdjarsync` command**

Execute the `pdjarsync` command (specifying the `-i` option).

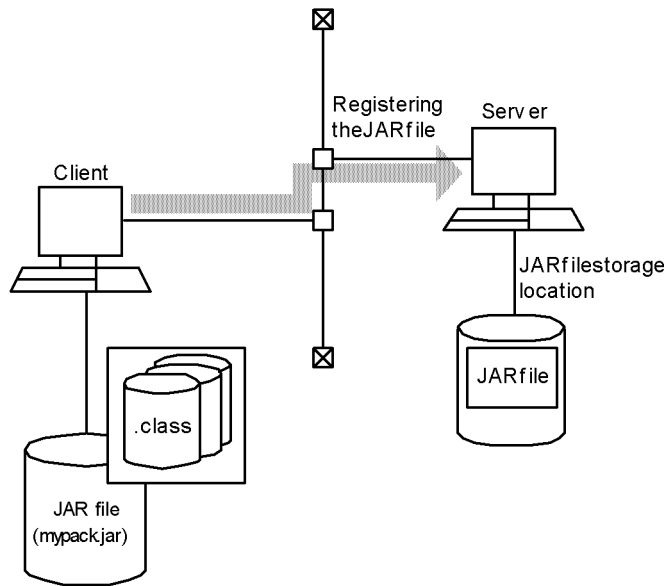
For details about the `pdjarsync` command, see the *HiRDB Version 8 Command Reference* manual.

- **Calling a Java method for installation**

Call a Java method for installation, which is `Jdbh_JARReInstall` in the `Jdbh_JARAccess` class, to register the JAR file.

Figure 9-6 shows an overview of JAR file registration.

Figure 9-6: Overview of JAR file registration



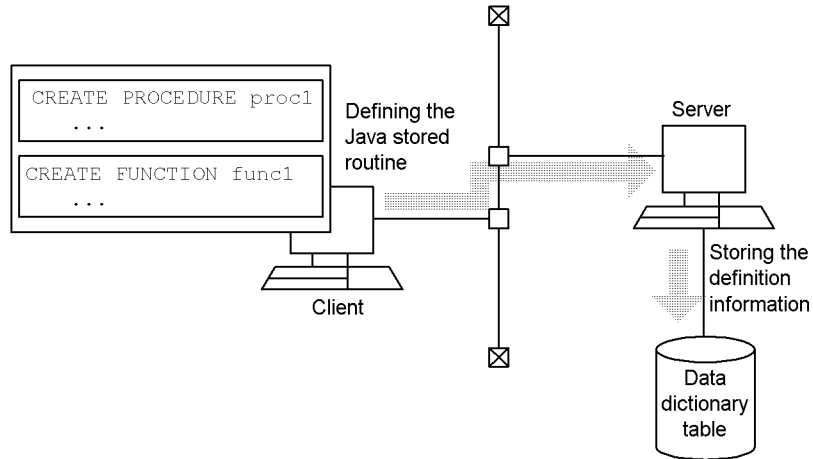
9.2.3 Defining the Java stored routine

To define a Java stored routine, use `CREATE PROCEDURE` or `CREATE FUNCTION`. `CREATE PROCEDURE` or `CREATE FUNCTION` defines association of a Java method with a procedure name or function name.

For details about `CREATE PROCEDURE` or `CREATE FUNCTION`, see the *HiRDB Version 8 SQL Reference* manual.

Figure 9-7 shows an example of Java stored routine definition.

Figure 9-7: Example of a Java stored routine definition



(1) Java stored procedure

Use `CREATE PROCEDURE` to register a Java method as a Java stored procedure.

(2) Java stored function

Use `CREATE FUNCTION` to register a Java method as a Java stored function.

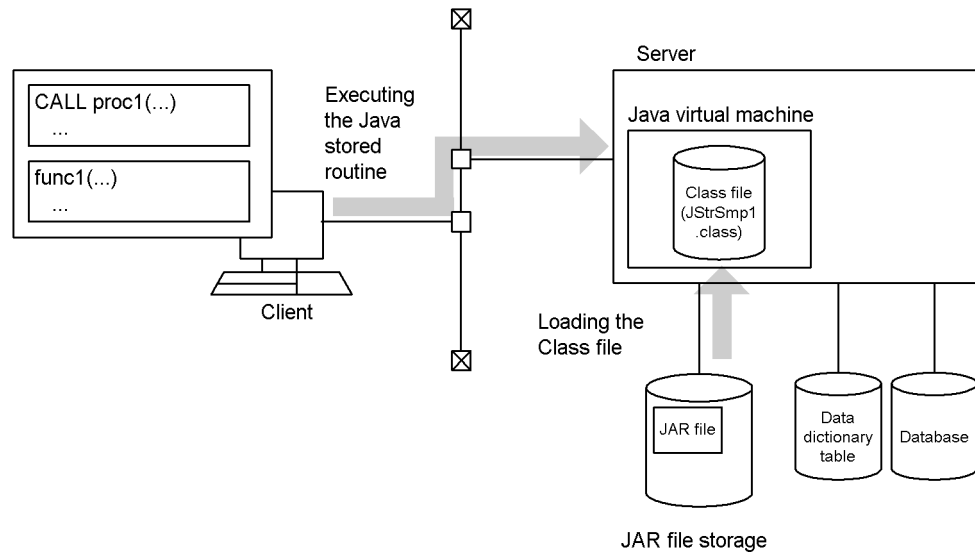
9.2.4 Executing the Java stored routine

To execute a Java stored routine, use the `CALL` statement or a function call. When the `CALL` statement or an SQL statement specifying the function call is executed, the Java method is called as the Java stored routine and executed at the server's Java virtual machine.

For details about the `CALL` statement and function calls, see the *HiRDB Version 8 SQL Reference* manual.

Figure 9-8 shows an example of executing a Java stored routine.

Figure 9-8: Example of Java stored routine execution



(1) Java stored procedure

Use the `CALL` statement to execute a Java method as a Java stored procedure.

(2) Java stored function

Use an SQL statement specifying a function call to execute a Java method as a Java stored function.

9.3 Sample programs of Java stored routine

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(IN pic_num INTEGER
                        , OUT pic_data BLOB(1M) ....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:PrdbDrive","USER1","PASS1"); .....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.
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:PrdbDrive","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

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.
5. Preprocesses the `CALL` statement.
6. Executes the `CALL` statement.
7. Obtains the `OUT` parameter of the `byte` array type.

9.3.2 Sample Java stored routines provided with HiRDB

(1) Sample 1

This sample Java stored procedure obtains a calendar of the specified year and month.

■ 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 shows an example of defining and executing the Java stored procedure using the previous Java procedure:

■ **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)

```

■ 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.PrdbDriver");
    } catch (ClassNotFoundException ex) {
        System.out.println("\n*** ClassNotFoundException caught ***\n");
        ex.printStackTrace();
        System.out.println("");
        System.out.println("\n*****\n");
        return;
    }

    jprocl(fromdate, todate);
}

/*=====*/
/* name = sample_2 method */
/*=====*/
public static void jprocl(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:PrdbDrive",
                                   "\"USER1\"", "\"PASS1\"");
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 shows an example of defining and executing the Java stored procedure using the previous Java procedure:

■ 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 jprocl(IN ?,IN ?)
```

(3) Sample 3

This example compresses and decompresses BLOB data using `gzip` and `ungzip`.

■ 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 shows an example of defining and executing the Java stored procedure using the previous Java function:

■ 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.

■ **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.PrdbDriver");
    } 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:PrdbDrive"
                                       , "\"USER1\"", "\"PASS1\"");
        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.PrdbDriver");
    } 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:PrdbDrive", "\"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 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.

Figure 9-9 shows method execution control using the security policy.

Figure 9-9: Method execution control using security policy



9.4.2 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.

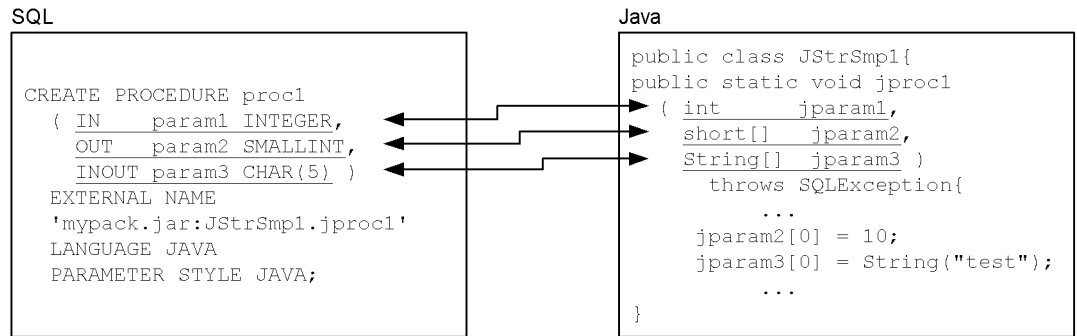
9.4.3 Parameter input/output mode mapping (Java stored procedures only)

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 8 SQL Reference* manual.

Figure 9-10 shows an example of parameter input/output mode mapping.

Figure 9-10: Example of parameter input/output mod mapping

**(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 INOUT 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.lang.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).

9.4.4 Results-set return facility (Java stored procedures only)

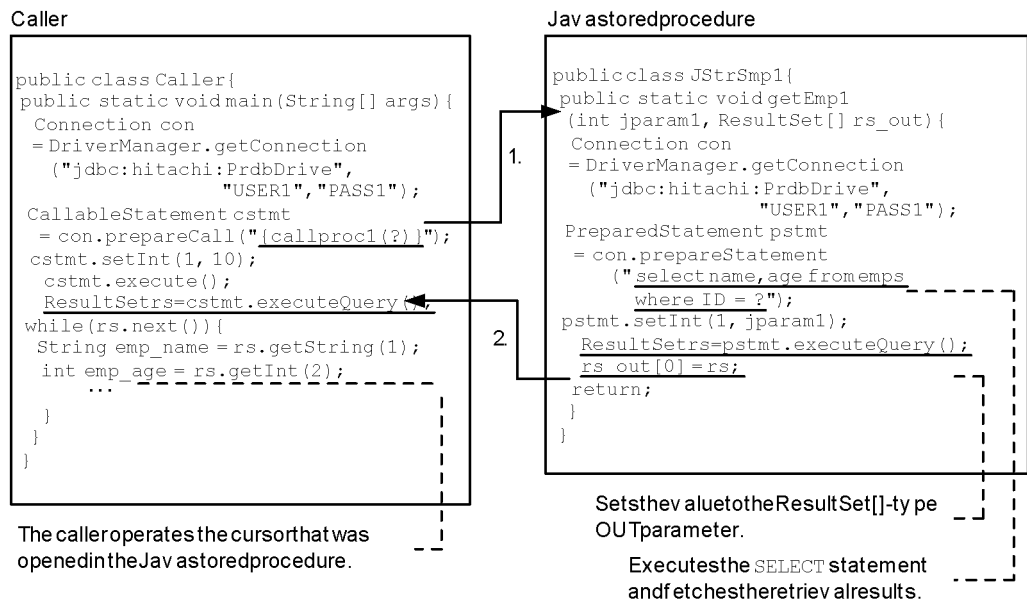
You can use the results-set return facility by specifying 1 or a greater value in the DYNAMIC RESULT SETS clause in CREATE PROCEDURE during Java stored procedure definition. The results-set return facility is not supported for Java stored functions.

(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.

Figure 9-11 shows an overview of the results-set return facility.

Figure 9-11: Overview of the results-set return facility (for a Java stored procedure)



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:PrdbDrive","USER1"
            ,"PASS1"); .....5

        PreparedStatement pstmt1 = con.prepareStatement ..6
            ("select rank,name,age from emps_1 where
                rank < ? .....6
            order by rank"); .....6

        pstmt1.setInt(1, jparam1); .....6

        ResultSet rs1 = pstmt1.executeQuery(); .....7

```

```

rs1_out[0] = rs1; .....8
PreparedStatement pstmt2 = con.prepareStatement ..9
    ("select rank,name,age from emps_2 where
        rank < ? .....9
        order by rank"); .....9
pstmt2.setInt(1, jparam1); .....9
ResultSet rs2 = pstmt2.executeQuery(); .....10
rs2_out[0] = rs2; .....11
return; .....12
}
}

```

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. Preprocesses the `SELECT` statement.
7. Executes the `SELECT` statement.
8. Sets the obtained result set `rs1` in the second argument of the `ResultSet[]` type.
9. Preprocesses the `SELECT` statement.
10. Executes the `SELECT` statement.
11. Sets the obtained result set `rs1` in the third argument of the `ResultSet[]` type.
12. 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:PrdbDrive","USER1"
            ,"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.

9.4.5 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.

9.4.6 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

Java stored routines are based on the architecture of running a normal Java program on a DBMS server; therefore, their testing and debugging methods 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.

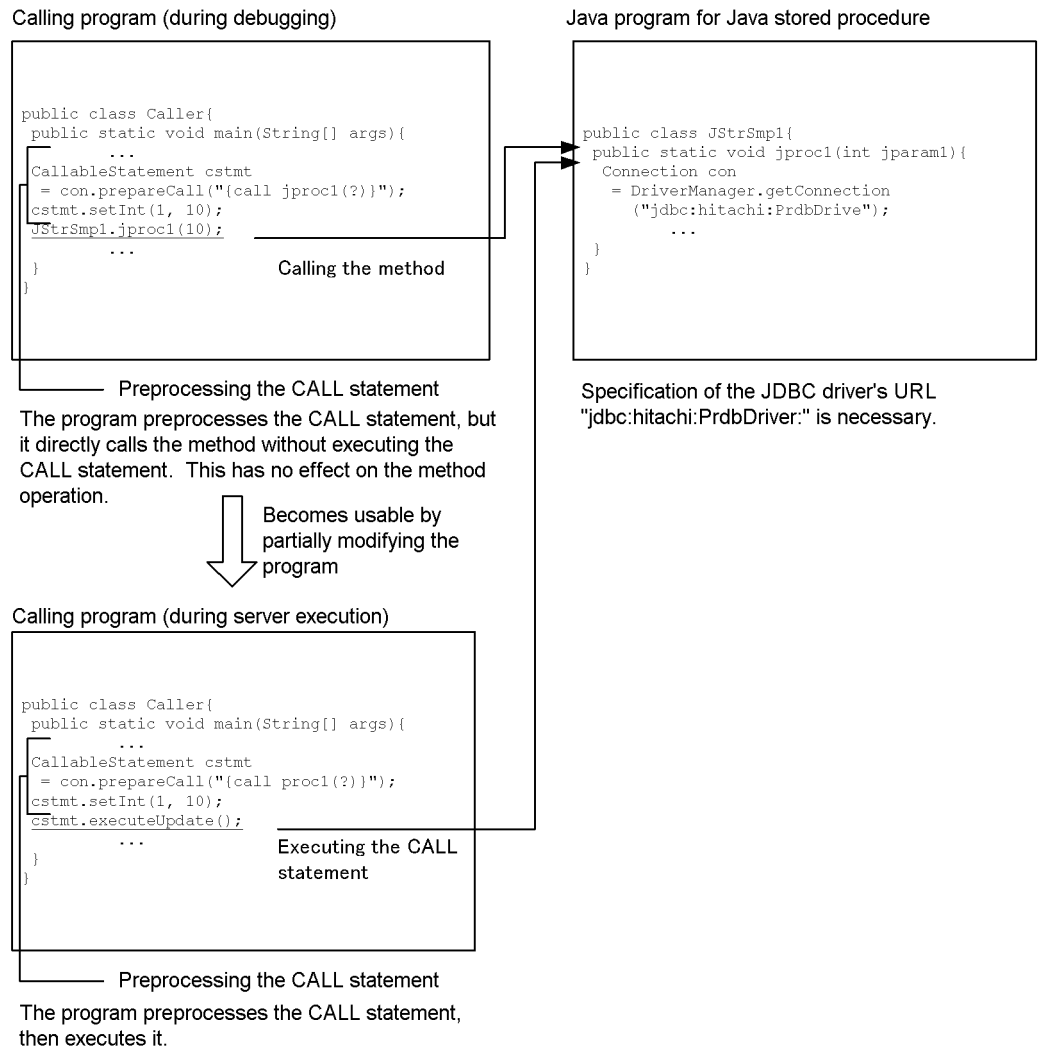
9.5.1 Java program for a Java stored procedure

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 *9.4.1 Unsupported methods*.

Figure 9-12 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



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:

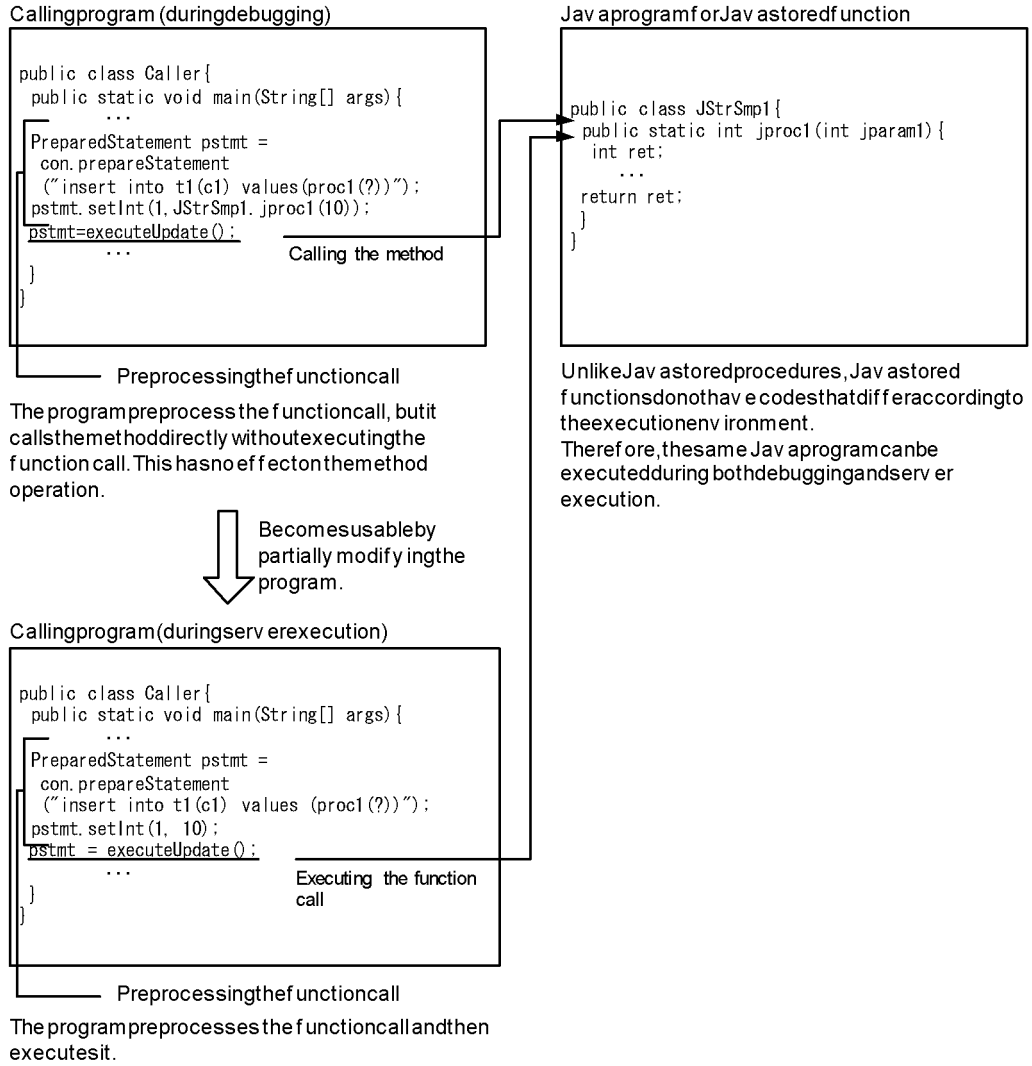
1. You can use a Java program for a Java stored function during server execution without having to modify the debugged program.
2. A caller Java program directly calls the method of the Java program for a Java

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 *9.4.1 Unsupported methods*.

Figure 9-13 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



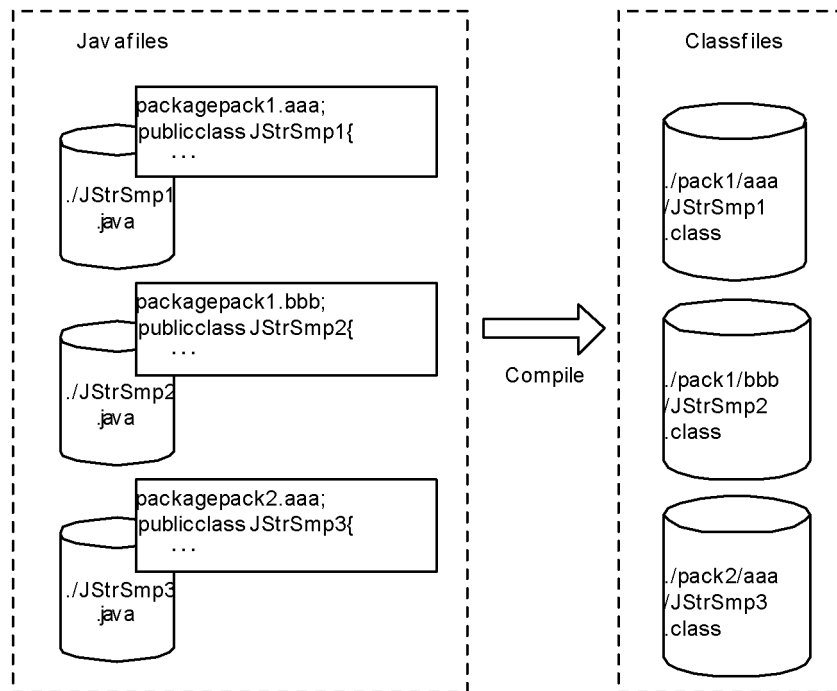
9.6 Notes about JAR file creation

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.

Figure 9-14 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

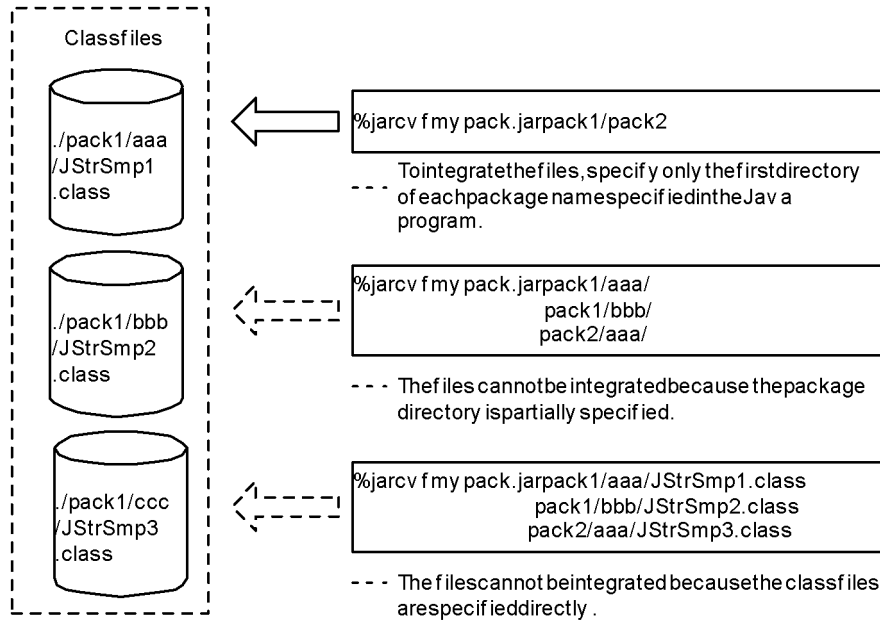
`GET JAVA STORED ROUTINE SOURCE`, you need to integrate the Java files at the same time. For details about the `GET JAVA STORED ROUTINE SOURCE` specification, see the *HiRDB Version 8 SQL Reference* manual.

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.

Figure 9-15 shows an example of integrating Class files in JAR files.

Figure 9-15: Example of integrating Class files in JAR files



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`.

Reader's Comment Form

We would appreciate your comments and suggestions on this manual. We will use these comments to improve our manuals. When you send a comment or suggestion, please include the manual name and manual number. You can send your comments by any of the following methods:

- Send email to your local Hitachi representative.
- Send email to the following address:
WWW-mk@itg.hitachi.co.jp
- If you do not have access to email, please fill out the following information and submit this form to your Hitachi representative:

Manual name:	
Manual number:	
Your name:	
Company or organization:	
Street address:	
Comment:	

(For Hitachi use)
