Scalable Database Server

HiRDB Version 8

UAP Development Guide Part II

3020-6-356(F)

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

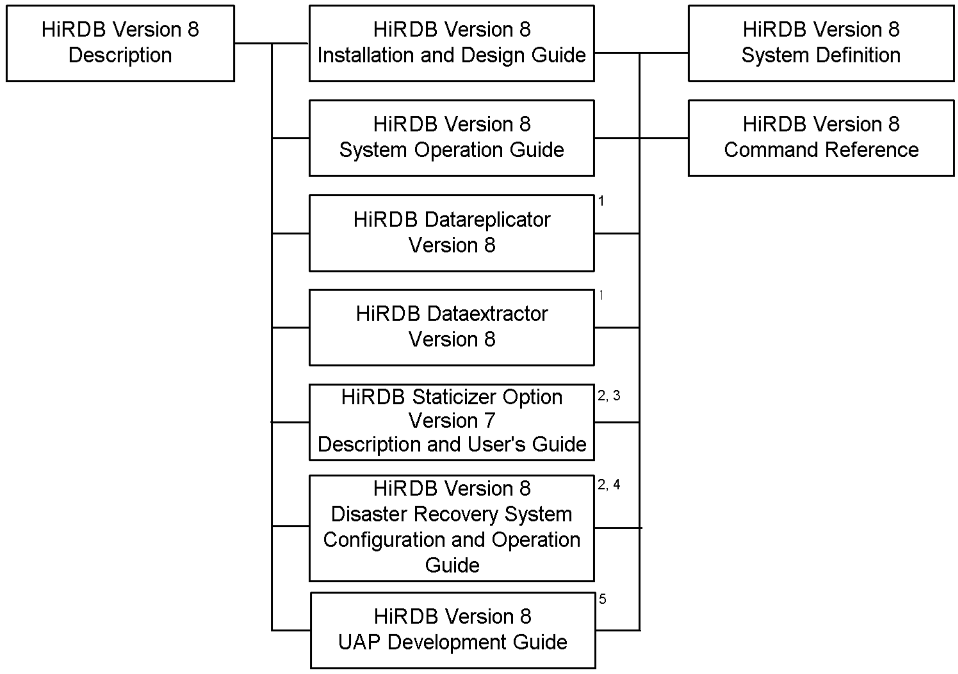
- *HiRDB External Data Access Version 7 Description and User's Guide* (3000-6-284(E))
- *Distributed Database System DF/UX* (3000-3-248(E))

- *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))

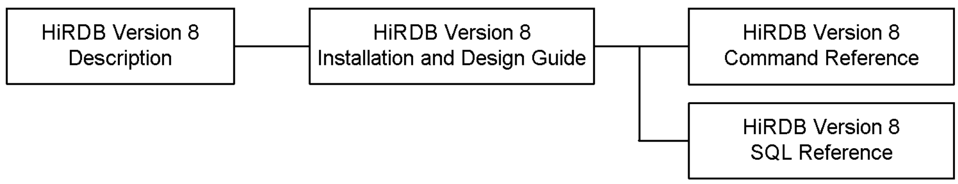
Organization of HiRDB manuals

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.

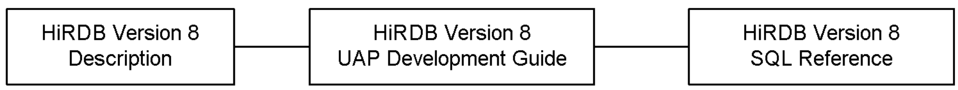
Manuals for system administrators:



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.

<table>
<tr><th>Name of product or other entity</th><th colspan="2">Representation</th></tr>
<tr><td>HiRDB/Single Server Version 8</td><td rowspan="2">HiRDB/Single Server</td><td rowspan="4">HiRDB or HiRDB Server</td></tr>
<tr><td>HiRDB/Single Server Version 8(64)</td></tr>
<tr><td>HiRDB/Parallel Server Version 8</td><td rowspan="2">HiRDB/Parallel Server</td></tr>
<tr><td>HiRDB/Parallel Server Version 8(64)</td></tr>
<tr><td>HiRDB/Developer's Kit Version 8</td><td rowspan="2">HiRDB/ Developer's Kit</td><td rowspan="4">HiRDB Client</td></tr>
<tr><td>HiRDB/Developer's Kit Version 8(64)</td></tr>
<tr><td>HiRDB/Run Time Version 8</td><td rowspan="2">HiRDB/Run Time</td></tr>
<tr><td>HiRDB/Run Time Version 8(64)</td></tr>
<tr><td>HiRDB Datareplicator Version 8</td><td colspan="2">HiRDB Datareplicator</td></tr>
<tr><td>HiRDB Dataextractor Version 8</td><td colspan="2">HiRDB Dataextractor</td></tr>
<tr><td>HiRDB Text Search Plug-in Version 7</td><td colspan="2">HiRDB Text Search Plug-in</td></tr>
<tr><td>HiRDB Spatial Search Plug-in Version 3</td><td colspan="2">HiRDB Spatial Search Plug-in</td></tr>
<tr><td>HiRDB Staticizer Option Version 8</td><td colspan="2">HiRDB Staticizer Option</td></tr>
<tr><td>HiRDB LDAP Option Version 8</td><td colspan="2">HiRDB LDAP Option</td></tr>
<tr><td>HiRDB Advanced Partitioning Option Version 8</td><td colspan="2">HiRDB Advanced Partitioning Option</td></tr>
<tr><td>HiRDB Advanced High Availability Version 8</td><td colspan="2">HiRDB Advanced High Availability</td></tr>
<tr><td>HiRDB Non Recover Front End Server Version 8</td><td colspan="2">HiRDB Non Recover FES</td></tr>
<tr><td>HiRDB Disaster Recovery Light Edition Version 8</td><td colspan="2">HiRDB Disaster Recovery Light Edition</td></tr>
<tr><td>HiRDB External Data Access Version 8</td><td colspan="2">HiRDB External Data Access</td></tr>
<tr><td>HiRDB External Data Access Adapter Version 8</td><td colspan="2">HiRDB External Data Access Adapter</td></tr>
<tr><td>HiRDB Adapter for XML - Standard Edition</td><td colspan="2" rowspan="2">HiRDB Adapter for XML</td></tr>
<tr><td>HiRDB Adapter for XML - Enterprise Edition</td></tr>
<tr><td>HiRDB Control Manager</td><td colspan="2">HiRDB CM</td></tr>
<tr><td>HiRDB Control Manager Agent</td><td colspan="2">HiRDB CM Agent</td></tr>
</table>

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 JavaTM 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	

<table>
<tr><th>Name of product or other entity</th><th colspan="2">Representation</th></tr>
<tr><td>Microsoft$_{(R)}$ Windows NT$_{(R)}$ Workstation Operating System Version 4.0</td><td colspan="2" rowspan="2">Windows NT</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows NT$_{(R)}$ Server Network Operating System Version 4.0</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ 2000 Professional Operating System</td><td colspan="2" rowspan="3">Windows 2000</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ 2000 Server Operating System</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ 2000 Datacenter Server Operating System</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ 2000 Advanced Server Operating System</td><td colspan="2">Windows 2000 or Windows 2000 Advanced Server</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003, Standard Edition</td><td colspan="2" rowspan="2">Windows Server 2003</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003, Enterprise Edition</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003 R2, Standard Edition</td><td colspan="2" rowspan="2">Windows Server 2003 R2 or Windows Server 2003</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003 R2, Enterprise Edition</td></tr>
<tr><td>64 bit Version Microsoft$_{(R)}$ Windows Server^{TM} 2003, Enterprise Edition (IPF)</td><td colspan="2">Windows Server 2003 (IPF) or Windows Server 2003</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003, Standard x64 Edition</td><td rowspan="2">Windows Server 2003 or Windows Server 2003 x64 Editions</td><td rowspan="5">Windows (x64)</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003, Enterprise x64 Edition</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003 R2, Standard x64 Edition</td><td rowspan="2">Windows Server 2003, Windows Server 2003 R2 or Windows Server 2003 x64 Editions</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows Server^{TM} 2003 R2, Enterprise x64 Edition</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ XP Professional x64 Edition</td><td>Windows XP or Windows XP x64 Edition</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ XP Professional Operating System</td><td>Windows XP Professional</td><td rowspan="2">Windows XP</td></tr>
<tr><td>Microsoft$_{(R)}$ Windows$_{(R)}$ XP Home Edition Operating System</td><td>Windows XP Home Edition</td></tr>
<tr><td>Single server</td><td colspan="2">SDS</td></tr>
</table>

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	Journaled File System
JFS2	Enhanced Journaled File System
JIS	Japanese Industrial Standard code
JP1	Job Management Partner 1
JRE	Java Runtime Environment
JTA	Java Transaction API
JTS	Java Transaction Service
KEIS	Kanji processing Extended Information System
LAN	Local Area Network
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**.

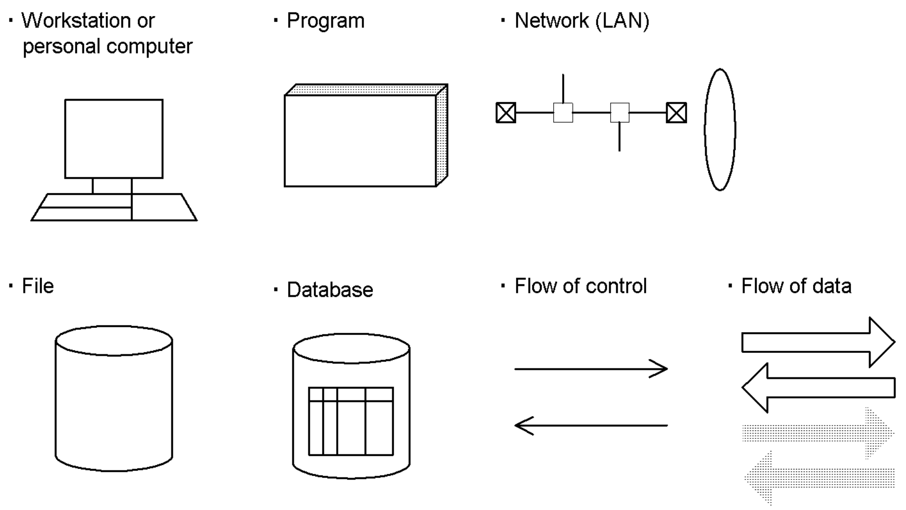
- UNIX version

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	`/var/adm/syslog/syslog.log`
Solaris	`/var/adm/messages` or `/var/log/syslog`
AIX 5L	`/var/adm/ras/syslog`
Linux	`/var/log/messages`

Conventions: Diagrams

This manual uses the following conventions in diagrams:



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: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italics	Italics are used to indicate a placeholder for some actual text provided by the user or system. Italics are also used for emphasis. For example: • Write the command as follows: `copy` *source-file target-file* • Do *not* delete the configuration file.
`Code font`	A code font indicates text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`

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:

S`tore`**D**`atabase [temp|`<u>`perm`</u>`]` `(`*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.
database-name	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
<*alphabetics-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 ↑34 ÷ 3↑ is 12.
↓ ↓	Discard digits following the decimal point. Example: The result of ↓34 ÷ 3↓ is 11.
MAX	Select the largest value as the result. Example: The result of Max(10, 2 × 4, 3 + 8) is 11.
MIN	Select the smallest value as the result. Example: The result of Min(10, 2 × 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 $1{,}024^2$ bytes.
- 1 GB (gigabyte) is $1{,}024^3$ bytes.
- 1 TB (terabyte) is $1{,}024^4$ bytes.

Conventions: Version numbers

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

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

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

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

List of figures

List of tables

Chapter

10. UAP Troubleshooting

This chapter explains the collection of historical information for UAP execution and the collection of error information to be used for troubleshooting. It also describes the types of UAP errors and the recovery methods.

This chapter contains the following sections:

10.1 Gathering error information

When an error occurs in a UAP, the troubleshooting functions can be used to investigate the cause of the error. The troubleshooting functions are as follows:

- SQL tracing
- Error logging
- Facility for output of extended SQL error information
- UAP statistical report facility
- Command trace facility
- SQL trace dynamic acquisition facility
- Reconnect trace facility
- Access path information file for HiRDB SQL Tuning Advisor

10.1.1 SQL tracing

This function collects in an SQL trace file the SQL trace information for an executed UAP.

If an SQL error occurs during UAP execution, the SQL trace information can be used to identify the SQL statement that caused the error.

When the current SQL trace file becomes full, a new file is swapped in and the previous information in that file can be overwritten.

(1) Collecting SQL trace information

SQL trace information is collected by setting values in the `PDCLTPATH` and `PDSQLTRACE` environment variables during client environment definition. For details about the client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

The two SQL trace files in which information is collected are created under a specified directory. The file names that are created depend on whether or not an API (`TX_function`) conforming to X/Open is used.

Table 10-1 shows the relationship between the `TX_function` and the files that are created.

Table 10-1: Relationship between the use of an API (TX_function) conforming to X/Open and created SQL trace files

Use of TX_function	Created SQL trace files
No	`pdsql1.trc` and `pdsql2.trc`
Yes	`pdsql`*xxxxx*`-1.trc` and `pdsql`*xxxxx*`-2.trc`

xxxxx: Process ID during UAP execution

(2) Examining SQL trace information

SQL trace information is output after the execution of SQL statements is completed. An example of output of SQL trace information is shown as follows, followed by an explanation.

Output example

```
                           [20]                   [19] [22]
** UAP TRACE (CLT:VV-RR(Mmm dd yyyy) SVR:VV-RR US) WIN32(WIN32) **

  USER APPLICATION PROGRAM FILE NAME : XXXXXXXX [1]
  USERID : YYYYYYYY [2]
  UAP START TIME : YYYY/MM/DD HH:MM:SS [3]
  UAP ENVIRONMENT : [4]
    LANG(ja_JP.SJIS)
    USER("YYYYYYY")
    HOST(dcm3500)
    NAMEPORT(20281)
    FESHOST( )
    SVCGRP( )  SVCPORT( )  SRVTYPE( )
    SWAIT(600)  CWAIT(0)   SWATCH(0)

    BLKF(1)  LOCKLMT(0)  ISLLVL(2)  DBLOG(ALL)  DFLNVAL(NOUSE)
    AGGR(1024)  DLKPRIO(64)  EXWARN(NO)  VWOPTMODE(0)
    LOCKSKIP(NO) CLTGRP(A)  PLGIXMK(NO)
    CLTRCVPORT(5000)  CLTRCVADDR(192.134.35.4)  PLGPFSZ(8192)
    PLGPFSZEXP(8192)  SPACELVL(-1)  STJTRNOUT( )
    OPTLVL("SELECT_APSL","RAPID_GROPING")
    ADDITIONALOPTLVL("COST_BASE_2","APPLY_HASH_JOIN")
    UAPREPLVL( )  REPPATH( )
    TRCPATH( )
```

```
      IPC(MEMORY)  SENDMEMSIZE(16)  RECVMEMSIZE(32)
      HASHTBLSIZE(128)  CMMTBFDDL(NO)  PRPCRCLS( )
      SQLTRCOPENMODE(SQL)  AUTOCONNECT(ON)  CWAITTIMEWRNPNT(-1)  TCPCONOPT(0)
      WRTLNFILSZ(-1) WRTLNCOMSZ(1024)
      WRTLNPATH( ) UAPENVFILE( )
      TP1SERVICE(NO) AUTORECONNECT(NO) RCCOUNT(0) RCINTERVAL(0)
      KALVL(0) KATIME(0) CLTCNVMODE(NOUSE)
      PRMTRC(YES) PRMTRCSIZE(256) BESCONHOLD() BESCONHTI(-1)
      BLKBUFFSIZE(0) BINARYBLKF(NO) FORUPDATEEXLOCK(NO)
      CNSTRNTNAME() SQLTEXTSIZE(4096) RCTRACE(-1)
      FESGRP()
      NBLOCKWAITTIME(0) CONNECTWAITTIME(300) DBBUFLRU(YES)
      UAPEXERLOGUSE() UAPEXERLOGPRMSZ() HJHASHINGMODE(TYPE1)
      DDLDEAPRP(NO) DELRSVWDFILE() HATRNQUEUING()
      ODBSPLITSIZE(100) CURSORLVL(0)
      TAAPINFPATH() TAAPINFMODE(0) TAAPINFSIZE(409600)
      JETCOMPATIBLE(NO) SUBSTRLEN()
    CONNECTION STATUS : [5]
      CURHOST(dcm3500)  CURPORT(4439)  SRVNAME(fes1)
      CNCTNO(1)  SVRPID(8945)  CLTPID(9155)  CLTTID( ) CLTCNCTHDL(0x0)


  [6]  [7]   [8]  [9] [10] [11] [12] [13]  [14]         [15]         [16] [23]
  CNCT CLPID CLTID NO OP   SEC  SQL  SQL   START-TIME   END-TIME     OP   EXEC-TIME
  NO                  CODE NO   CODE WARN                            TION
  ---- ----- ----- -- ---- ---  ---- ----- ------------ ------------ ---- -----------
    1  9155      0  1 CNCT  0      0 WC040 16:03:55.720 16:03:58.080 0001     2356125
    1  9155      0  2 AUI2  1      0 -0000 16:03:58.630 16:03:59.400 M000      769651

 *SQL* INSERT INTO STOCK(GNO,GNAME,PLAN,PRICE,QUANTITY,DISCOUNT) VALUES(?,?,
.........17
 ?,?,?,?)
.......................................................................17

    1  9155      0  3 SET   2      0 -0000 16:04:00.820 16:04:01.540 M000      719825

 *SQL* SELECT GNO,GNAME,PLAN,PRICE,QUANTITY,DISCOUNT FROM STOCK
......................17

    1  9155      0  4 OPEN  2      0 -0000 16:04:02.090 16:04:02.800 M000      709123
    1  9155      0  5 FETC  2   -204 -0000 16:04:03.080 16:04:03.790 M000      708902
    1  9155      0  6 SET   2      0 W8800 16:04:04.060 16:04:04.830 M000      765147

 *SQL(AUTHID)* INSERT INTO TBL01 VALUES('12345',12345)  ..........................17
```

```
   1  9155      0  7 SAUT   0     0 -0000 16:04:04.834 16:04:04.835 M000        912

 *USER* hirdb01
..................................................................18

   1  9155      0  8 AUI2   3     0 -0000 16:05:05.110 16:05:05.121 M000       9456

 *SQL* INSERT INTO TBL01 VALUES(?,100)  ......................................21
 *PARAM* NO=       1 COD=c5 XDIM=   1 SYS=     0 LEN=      15 IND=       0  .......21
         DATA=30 35 2d 30 35 00 00 00 00 00 00 00 00 00 00     *05-05.........*   ..21

   1  9155         9 DISC   0     0 -0000 16:05:55.110 16:05:56.660 M004    1547893
```

1. UAP name: Displays the name specified in the `PDCLTAPNAME` environment variable.
2. Authorization identifier: Displays the authorization identifier of the user who executed the UAP.
3. UAP start time: Displays the time at which execution of the UAP started.
4. UAP execution environment: Displays the values of the environment variables when the UAP was executed.
5. UAP execution status: Displays the status of the connection with the server when the UAP was executed:
    - `CURHOST`: Connection-destination host name
    - `CURPORT`: Connection port number
    - `SRVNAME`: Front-end server name or single-server name
    - `CNCTNO`: Connection serial number
    - `SVRPID`: Connected server process status
    - `CLTPID`: UAP process number
    - `CLTTID`: UAP thread number
    - `CLTCNCTHDL`: Connection handle

    If information cannot be obtained, an invalid value may be displayed (Windows).
6. Connection serial number: Displays the connection serial number. Connection serial numbers are assigned sequentially each time the server accepts `CONNECT`.
7. Displays the process number of the UAP.

    If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

8. UAP thread number: Displays the UAP thread number when the UAP is running in a multi-thread environment. Displays `0` if the UAP is not running in a multi-thread environment. A thread number that cannot be allocated can sometimes be displayed as an invalid value.
9. SQL counter: Displays the SQL counter values. Each time an SQL statement is accepted, the counter value is incremented (from 1 through 999999, after which the counter value returns to `1`).
10. Operation code: Displays the operation code that corresponds to each SQL statement.

    The following table shows the SQL statements that correspond to the displayed operation codes.

Operation code	Corresponding SQL statement
AUI2	DELETE statement (static SQL), INSERT statement (static SQL), UPDATE statement (static SQL), LOCK statement (static SQL), PURGE TABLE statement (static SQL), single-row SELECT statement (static SQL), FREE LOCATOR statement (static SQL)
AUI3	Assignment statement (static SQL)
AUX	EXECUTE statement
AUXI	EXECUTE IMMEDIATE statement, all definition SQL statements
AUXO	EXECUTE statement (INTO specified)
CALL	CALL statement
CLOS	CLOSE statement
CMIT	COMMIT statement
CNCT	CONNECT statement
CPRP	Commit prepare*
DESC	DESCRIBE statement (OUTPUT specified)
DEST	DESCRIBE TYPE statement
DISC	DISCONNECT statement, COMMIT statement (RELEASE specified)
DISR	ROLLBACK statement (RELEASE specified)
DIST	Disconnect + Tran Check*
DSCM	Used by the system.
DSPR	Used by the system.

Operation code	Corresponding SQL statement
DSRL	Used by the system.
FETC	FETCH statement
GETD	GET DIAGNOSTICS
HVAR	DESCRIBE statement (INPUT specified)
JARI	INSTALL JAR
JARR	REPLACE JAR
JARU	REMOVE JAR
OPEN	OPEN statement (dynamic SQL)
OPN2	OPEN statement (static SQL)
OPNR	OPEN statement (dynamic SQL (multiple cursors))
RENV	Used by the system.
RNCN	CONNECT statement (TO specified)
RNDS	DISCONNECT statement (TO specified)
RNSC	SET CONNECTION statement
ROLL	ROLLBACK statement
RSDC	DESCRIBE statement (OUTPUT and RESULT SET specified)
RSFT	FETCH statement (RESULT SET specified)
RSCL	CLOSE statement (RESULT SET specified)
SAUT	SET SESSION AUTHORIZATION statement
SET	PREPARE statement
SINF	Used by the system.
SOPT	Used by the system.
SVLS	Used by the system.
THRE	Used by the system.
THSU	Used by the system.
TRCK	Used by the system.
TRC2	Used by the system.

Operation code	Corresponding SQL statement
`TRST`	Used by the system.
`TSCM`	Used by the system.
`TSRL`	`Transfer Rollback`*
`TSPR`	`Transfer Prepare`*
`ALCR`	`ALLOCATE CURSOR` statement
`DSET`	`DEALLOCATE PREPARE` statement

* Applicable only when the XA interface is used.

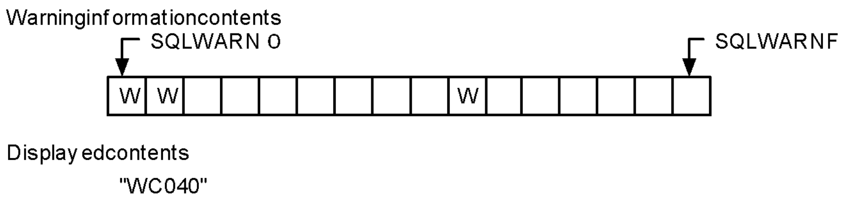
11. Section number: Displays a number for verifying SQL statement correspondence; this number is assigned automatically by the SQL preprocessor.

12. `SQLCODE`: Displays the `SQLCODE` that occurs as a result of SQL statement execution.

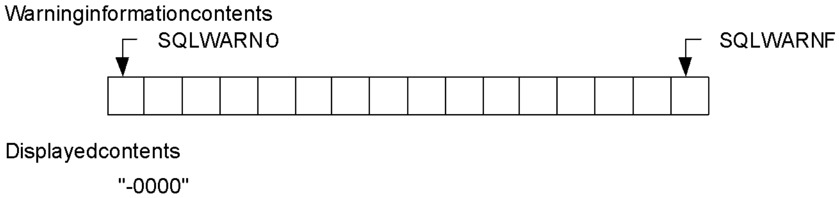
13. `SQLWARN`: Displays warning information (in hexadecimal). Starting from the left, one bit each is allocated to warning information `SQLWARN0` through `SQLWARNF`. A 16-bit value is obtained by setting each bit to `1` if the warning flag is set and to `0` if it is not set. This obtained value is displayed as a 4-digit hexadecimal number.

    `W` is displayed at the beginning if at least one warning flag is set; - is displayed if no warning flags are set.

    Example 1



    Example 2



14. SQL statements execution request receipt time: Displays the time at which the SQL execution request was received (in *HH:MM:SS:mmm* format).
15. SQL statement execution request termination time: Displays the time at which the SQL statement execution request was terminated (in *HH:MM:SS:mmm* format).
16. Information used by the system: Displays the information used by the system. If the first byte is `M`, memory is used for process-to-process communication. The other part of the information is used by the HiRDB developer for maintenance purposes.
17. SQL statement: Displays the SQL statement, but only when the operation code is `SET`, `AUXI`, `AUI2`, or `OPN2`.

    The maximum SQL statement length that can be output is 4,096 bytes; any excess is truncated. If the `-A` option was specified during preprocessing or the `/A` option was used during preprocessing to specify the authorization identifier to be assumed when the authorization identifier in the `SQL` statement is omitted, `*SQL*` is displayed as `*SQL (`*assumed-authorization-identifier*`)*`.
18. New user identifier: Displays a new user identifier if the user identifier was changed during a single connection. This information is also displayed if the user identifier change operation fails.
19. Platform for UAP:

Platform	Character string to be displayed
HP-UX 11.0	`HP32`
HP-UX 11.0 (64-bit mode)	`HP64`
Solaris	`SOL`
Solaris (64-bit mode)	`SOL64`
AIX 5L	`AIX`
AIX 5L (64-bit mode)	`AIX64`
Linux	`LINUX`
Windows	`WIN32`
HP-UX (IPF) 32-bit mode	`HPI32`
HP-UX (IPF) 64-bit mode	`HPI64`
Linux (IPF)	`LINI64`
Linux (EM64T)	`LINX64`
Windows Server 2003 (IPF)	`WINI64`

Platform	Character string to be displayed
Windows (x64) 64-bit mode	WINX64
Type4 JDBC driver	Type4

20. Library creation date: Displays the creation date of the linked library in the following format:

    Mmm: Month (first three letters of the month in English with the first letter in upper case). For example, June is displayed as Jun.

    dd: Date

    yyyy: Year

21. Parameter trace: Displays input parameter information, output parameter information, and retrieved data when PDPRMTRC=YES, IN, OUT, or INOUT is specified in the client environment definitions.

    The parameter information data is displayed up the length specified in PDPRMTRCSIZE (or 256 bytes if omitted), and any excess part is discarded. For details, see *(4) Parameter trace output examples*.

    NO

    Parameter number

    COD

    Data type code

    XDIM

    Number of array elements

    SYS

    Length of one element, including gaps

    LEN

    Data length

    IND

    Value of indicator variable

    ARRAY NUM

    Number of elements in repetition array

    ROW NUM

    Number of execution rows in SQL that uses embedded variables in an array

DATA

Data (dump format)

22. Linked library name

Library name	Displayed characters
libzclt.sl, libclt.a	UNIX, UNIX_32
libzclts.sl, libclts.a	UNIX_S, UNIX_32S
libzcltm.sl, libcltm.a	UNIX_M, UNIX_32M
libzcltk.sl, libcltk.a	UNIX_K, UNIX_32K
libzcltx.sl, libcltxa.a	UNIX_XA, UNIX_XA_32
libzcltxs.sl, libcltxas.a	UNIX_XA_S, UNIX_XA_32S
libzcltxm.sl, libcltxam.a	UNIX_XA_M, UNIX_XA_32M
libzcltxk.sl, libcltxak.a	UNIX_XA_K, UNIX_XA_32K
libzclt64.sl, libclt64.a	UNIX_64
libzcltk64.sl, libcltk64.a	UNIX_64K
libzclts64.sl	UNIX_64S
libzcltx64.sl, libzclty64.sl	UNIX_XA_64
libzcltxk64.sl, libzcltyk64.sl	UNIX_XA_64K
libzcltxs64.sl, libzcltys64.sl	UNIX_XA_64S
CLTDLL.DLL	WIN_32
PDCLTM32.DLL	WIN_M32
PDCLTM50.DLL	WIN_M50
PDCLTP32.DLL	WIN_P32
PDCLTX32.DLL	WIN_XA_32
PDCLTXM.DLL	WIN_XA_32M
PDCLTXS.DLL	WIN_XA_32S
PDCLTXM5.DLL	WIN_XA_50M
PDCLTM64.DLL	WIN_M64
PDCLTX64.DLL	WIN_XA_64

Library name	Displayed characters
PDCLTXM64.DLL	WIN_XA_64M
PDCLTXS64.DLL	WIN_XA_64S
PDJDBC2.JAR	Type4

23. SQL runtime: Displays the SQL runtime in microseconds when `PDSQLEXECTIME=YES` is specified in the client environment definitions.

(3) Making a backup of an SQL trace file

If the SQL trace file becomes full while SQL trace information is being output, HiRDB stops writing to that file and outputs SQL trace information to another SQL trace file. Any information that already exists in the switched-in SQL trace file is overwritten in chronological order by the new SQL trace information. To prevent that information from being lost, copy the contents of the SQL trace file into a backup file whenever execution of a UAP is completed.

To determine the SQL trace file that is being used currently, check the most recent update dates/times of the files. The SQL trace file that was updated most recently is the current file.

For a Windows version HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX version HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

(4) Parameter trace output examples

Output examples of representative parameter traces are shown below.

(a) INSERT statement (with null value and repetition column)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    7  1088  2060      1 CNCT    0    0 -0000 18:47:21.435 18:47:21.755 0000
    7  1088  2060      2 AUI2    1    0 -0000 18:47:21.765 18:47:21.765 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 65                                   *...e          *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=        0
             ARRAY NUM=    5
           0 DATA(    0)=00 01 61                                     *..a              *
           0 DATA(    1)=00 07 62 62 62 62 62 62 62                   *..bbbbbbb        *
           0 DATA(    2)=00 04 63 63 63 63                            *..cccc           *
           0 DATA(    3)=00 09 64 64 64 64 64 64 64 64 64             *..ddddddddd      *
           0 DATA(    4)=00 0a 65 65 65 65 65 65 65 65 65 65          *..eeeeeeeeee     *
    7  1088  2060      3 AUI2    2    0 -0000 18:47:21.785 18:47:21.795 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 66                                   *...f          *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=        0
             ARRAY NUM=    5
           0 DATA(    0)=00 01 61                                     *..a              *
          -1 DATA(    1)=
           0 DATA(    2)=00 04 63 63 63 63                            *..cccc           *
          -1 DATA(    3)=
           0 DATA(    4)=00 4f 65 65 65 65 65 65 65 65 65 65 65 65 65 65  *.Oeeeeeeeeeeeeee*
                         65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65  *eeeeeeeeeeeeeeee*
                         --- SAME 3 LINES ---
                         65                                               *e               *
    7  1088  2060      4 AUI2    3    0 -0000 18:47:21.805 18:47:21.815 0000
     *SQL* INSERT INTO TBL01(C1,C3) VALUES(?,?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=        0
             DATA=00 00 00 67                                   *...g          *
     *INPRM* NO=    2 COD=c1 XDIM=    5 SYS=  102 LEN=       100 IND=       -1
             DATA=
    7  1088  2060      5 DISC    0    0 -0000 18:47:21.825 18:47:21.825 0000
```

Explanation

This is an output example of parameter trace information when `INTEGER` and `VARCHAR(10)` repetition column (five elements) values are inserted with the `INSERT` statement. The values are output in the sequence in which the `?` parameters are specified.

1. For input parameters, `*INPRM*` is displayed. However, when `PDPRMTRC=YES`, `*PARAM*` is displayed.
2. For a repetition column, the number of repeated elements is displayed in `ARRAY NUM`.

3. The number before each `DATA` clause is the indicator variable of each element in the repetition column.
4. The number in parentheses in each `DATA` clause is the repetition column element number.
5. For `VARCHAR`-type data, the first 2 bytes of `DATA` is the data length area (the first 4 bytes for `BINARY`-type data, and the first 8 bytes for `BLOB`-type data). When `PDPRMTRC` is `YES`, the size of the output information is the sum of the defined length and the data area length. When `PDPRMTRC` is `IN`, `OUT`, or `INOUT`, the size of the output information is the sum of the actual data length and the data area length.
6. If the indicator variable is a negative value, only the information up to `DATA=` is displayed.
7. If the data extends beyond one line, `--- SAME` *x* `LINES ---` (*x* is the number of lines) is output. However, when `PDPRMTRC=YES`, all data is output.

(b) Single-row SELECT statement

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   12  1492  2260      1 CNCT    0    0 -0000 19:18:31.914 19:18:32.135 0000
   12  1492  2260      2 AUI2    1    0 -0000 19:18:32.135 19:18:32.145 0000
    *SQL* SELECT C2,C3 FROM TBL02 WHERE C1=? AND C4=?
    *OUTPM* NO=    1 COD=c4 XDIM=    1 SYS=    0 LEN=       10 IND=         0
            DATA=41 41 41 41 41 41 41 41 41 41                    *AAAAAAAAAA      *
    *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=       10 IND=         0
            DATA=00 08 61 61 61 61 61 61 61 61                    *..aaaaaaaa      *
    *INPRM* NO=    3 COD=f0 XDIM=    1 SYS=    0 LEN=        4 IND=         0
            DATA=00 00 00 65                                      *...e            *
    *INPRM* NO=    4 COD=f4 XDIM=    1 SYS=    0 LEN=        2 IND=         0
            DATA=00 62                                            *.b              *
   12  1492  2260      3 DISC    0    0 -0000 19:18:32.155 19:18:32.155 0000
```

Explanation

This is an output example of parameter trace information when `PDPRMTRC=INOUT` is specified. The retrieval data information is output first in retrieval item sequence, and the input parameter information is output later in the specification sequence.

1. This is the retrieval data information. This information is not output when `PDPRMTRC=IN`. When `PDPRMTRC=YES`, `*PARAM*` is displayed instead of `*OUTPM*`.

2. This is the input parameter information. This information is not output when `PDPRMTRC=OUT`. When `PDPRMTRC=YES`, `*PARAM*` is displayed instead of `*INPRM*`.

(c) Stored procedure execution (CALL statement)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
   16  1456  2188      1 CNCT    0    0 -0000 19:43:00.486 19:43:00.797 0000
   16  1456  2188      2 CALL    1    0 -0000 19:43:00.797 19:43:00.807 0000
     *SQL* CALL PROC1(IN?,OUT?,INOUT?)
     *INPRM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 78                                  *...x          *
     *INPRM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 09 63 63 63 63 63 63 63 63 63             *..ccccccccc   *
     *OUTPM* NO=    1 COD=f0 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 dc                                  *....          *
     *OUTPM* NO=    2 COD=c0 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 09 63 63 63 63 63 63 63 63 63             *..ccccccccc   *
   16  1456  2188      3 DISC    0    0 -0000 19:43:00.829 19:43:00.829 0000
```

Explanation

1. This is the `IN` parameter. When `PDPRMTRC=OUT`, this information is not output.
2. This is the input parameter of the `INOUT` parameter. However, the contents of the `DATA` clause become output data.
3. This is the `OUT` parameter. This information is not output when `PDPRMTRC=IN` or `YES`.
4. This is the output parameter of the `INOUT` parameter. This information is not output when `PDPRMTRC=IN` or `YES`.

(d) Retrieval (FETCH statement)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      1 CNCT    0    0 -0000 14:49:54.326 14:49:54.696 0000
    6   668  1664      2 OPN2    1    0 -0000 14:49:54.736 14:49:54.746 0000
     *SQL* SELECT*FROM TBL03
    6   668  1664      3 FETC    1    0 -0000 14:49:54.746 14:49:54.746 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 78                                     *...x         *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=41 41 41 41 41 41 41 41 41 41                   *AAAAAAAAAA   *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 08 61 61 61 61 61 61 61 61                   *..aaaaaaaa   *
    6   668  1664      4 FETC    1    0 -0000 14:49:54.756 14:49:54.756 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 96                                     *....         *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=43 43 43 43 43 43 43 43 43 43                   *CCCCCCCCCC   *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 06 63 63 63 63 63 63                         *..cccccc     *
    6   668  1664      5 FETC    1    0 -0000 14:49:54.756 14:49:54.766 0000
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    0 LEN=         4 IND=         0
             DATA=00 00 00 b4                                     *....         *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=44 44 44 44 44 44 44 44 44 44                   *DDDDDDDDDD   *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=    0 LEN=        10 IND=         0
             DATA=00 09 64 64 64 64 64 64 64 64 64                *..ddddddddd  *
    6   668  1664      6 FETC    1  100 -0000 14:49:54.766 14:49:54.766 0000
    6   668  1664      7 CLOS    1    0 -0000 14:49:54.776 14:49:54.776 0000
    6   668  1664      8 CMIT    0    0 -0000 14:49:54.776 14:49:54.776 0000
```

Explanation

This is an output example of parameter trace information for the `FETCH` statement. A parameter trace is not output when `PDPRMTRC=IN` or `YES`.

1. If the `SQLCODE` of the `FETCH` statement is a value other than `0`, a parameter trace is not output.

(e) Retrieval (FETCH facility using arrays)

```
CNCT  CLPID CLTID NO     OP   SEC  SQL  SQL   START-TIME   END-TIME     OP
NO                       CODE NO   CODE WARN                            TION
----- ----- ----- ------ ---- ---- ---- ----- ------------ ------------ ----
    6   668  1664      9 OPN2    2    0 -0000 14:49:54.786 14:49:54.786 0000
     *SQL* SELECT*FROM TBL03
    6   668  1664     10 FETC    2    0 -0000 14:49:54.786 14:49:54.796 0002
     *ROW NUM = 2*
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=        4 IND=        0
           0 DATA(    0)=00 00 00 78                                *...x           *
           0 DATA(    1)=00 00 00 96                                *....           *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=       10 IND=        0
           0 DATA(    0)=41 41 41 41 41 41 41 41 41 41              *AAAAAAAAAA     *
           0 DATA(    1)=43 43 43 43 43 43 43 43 43 43              *CCCCCCCCCC     *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=       10 IND=        0
           0 DATA(    0)=00 08 61 61 61 61 61 61 61 61              *..aaaaaaaa     *
           0 DATA(    1)=00 06 63 63 63 63 63 63                    *..cccccc       *
    6   668  1664     11 FETC    2  100 -0000 14:49:54.796 14:49:54.806 0001
     *ROW NUM = 1*
     *OUTPM* NO=    1 COD=f1 XDIM=    1 SYS=    4 LEN=        4 IND=        0
           0 DATA(    0)=00 00 00 b4                                *....           *
     *OUTPM* NO=    2 COD=c5 XDIM=    1 SYS=   11 LEN=       10 IND=        0
           0 DATA(    0)=44 44 44 44 44 44 44 44 44 44              *DDDDDDDDDD     *
     *OUTPM* NO=    3 COD=c1 XDIM=    1 SYS=   12 LEN=       10 IND=        0
           0 DATA(    0)=00 09 64 64 64 64 64 64 64 64 64           *..ddddddddd    *
    6   668  1664     12 CLOS    2    0 -0000 14:49:54.806 14:49:54.816 0000
    6   668  1664     13 DISC    0    0 -0000 14:49:54.826 14:49:54.846 0000
```

Explanation

This is an output example of parameter trace information for the FETCH facility using arrays. A parameter trace is not output when PDPRMTRC=IN or YES.

1. ROW NUM displays the number of array elements (number of retrieval rows).
2. The number before each DATA clause is the indicator variable of each array element.
3. The number in parentheses in each DATA clause is the array element number.
4. If the SQLCODE of the FETCH statement is a value other than 0, parameter trace information is output for the number of rows returned from the server.

10.1.2 Error logging

If an error occurs during communication between a client and the HiRDB server or in the XA interface specified by X/Open, error information is collected as an error log in an error log file.

When the current error log file becomes full, a new file is swapped in and the oldest information in that file can be overwritten.

(1) Collecting error log information

An error log can be collected by setting values in the `PDCLTPATH` and `PDUAPERLOG` environment variables during client environment definition. For details about client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

The two error log files in which information is collected are created under a specified directory. The files that are created depend on whether or not an API (`TX_function`) conforming to X/Open is used.

Table 10-2 shows the relationship between the use of an API (`TX_function`) conforming to X/Open and the error log file that is created.

Table 10-2: Relationship between use of API (TX_function) conforming to X/Open and created error log files

Use of TX_function	Created error log files
No	`pderr1.trc` and `pderr2.trc`
Yes	`pderr`*xxxxx*`-1.trc` and `pderr`*xxxxx*`-2.trc`

xxxxx: Process ID during UAP execution

(2) Interpreting an error log

Error log information is output when an error occurs during SQL execution, during communication, or during execution of an XA interface function specified by X/Open. An example of output of error log information is shown as follows, followed by an explanation.

Output example

```
 1.  2.   3.    4.  5.             6.                          7.
>  672  0  5223  0  1997/12/18 22:07:46  KFPZ02444-E Communication error,fu   ect,
                                         errno=2
>  672  0  5223  1  1997/12/18 22:07:46  KFPZ02444-E Communication error,fu   ect,
                                         errno=2
                                              8.        9.    10.         11.      12.
>> 672  0  5223  2  1997/12/18 22:07:46  SQLCODE:-723 5 (205622412) cltsql.c:   AUI2
KFPA11723-E Communication error occurred, reason=NETWORK   7.
```

1. Error log leading identifier: >> is displayed for an error that occurred during SQL execution; > is displayed for any other error.
2. UAP process number: Displays the process number of the UAP where the error

occurred. If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

3. UAP thread number: Displays the UAP thread number when the UAP in which the error occurred is running in a multi-thread environment. Displays `0` if the UAP is not running in a multi-thread environment. The correct thread number cannot be assigned, and an invalid numeric value can sometimes be displayed as a result.
4. Server process number: Displays the process number at the server that is connected.
5. Error log counter: Displays the error log counter values. Each time error log information is accepted, the counter value is incremented (from `0` through `65535`).
6. Collection date and time: Displays the date and time at which the error log information was collected (in *YYYY/MM/DD HH:MM:SS* format).
7. Log data: Displays the error information (error message).
8. `SQLCODE`: Displays the `SQLCODE` when the error log corresponds to an `SQLCODE` to be returned to the UAP.
9. SQL counter: Displays the SQL counter value for the SQL statement in which the error occurred. For details about the SQL counter, see the output example explanation in Section *10.1.1 SQL tracing*.
10. Error collection time: Displays (in milliseconds) the amount of time used to collect the error log information.
11. Error detection location: Displays the name of the source file and the row number where the error was detected.
12. Operation code: Displays the operation code of the SQL statement in which the error occurred.

(3) Making a backup of an error log file

If the error log file becomes full while error log information is being output, HiRDB stops writing to that file and outputs error log information to another error log file. Any information that already exists in the switched-in error log file is overwritten in chronological order by the new error log information. To prevent that information from being lost, copy the contents of the error log file into a backup file whenever execution of a UAP is completed.

To determine the error log file that is being used currently, check the most recent update dates/times of the files. The error log file that was updated most recently is the current file.

For a Windows version HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX version HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

10.1.3 Facility for output of extended SQL error information

(1) What is the facility for output of extended SQL error information

The facility for output of extended SQL error information performs the following functions:

- Outputs the affected SQL statement and parameter information to the information of the error log facility. (The information produced when the SQL statement and parameter information are added to the information of the error log facility is called *SQL error information*.)
- Outputs SQL error information to the server as well. (The file to which SQL error information is output is called the *SQL error report file*.)

(2) Benefits

- Centralized management of SQL error information

  If an SQL error occurs, SQL error information is output on the server side, as well as on the client side. Since SQL error information for multiple clients can be output to the SQL error report file of one server, centralized management of **`SQL error`** information is possible.

- Output of the affected SQL statement and parameter information

  The SQL statement affected by the error and the related parameter information are output. The affected SQL statement can be investigated from this information.

(3) Usage method

When you use the facility for output of extended SQL error information, specify the following system definitions or client environment definitions:

- Whether or not the facility for output of extended SQL error information is to be used

  Use the `pd_uap_exerror_log_use` operand or `PDUAPEXERLOGUSE` to set whether or not the facility for output of extended SQL error information is to be used. Specify the `pd_uap_exerror_log_use` operand to set a value for the entire HiRDB system, and specify `PDUAPEXERLOGUSE` to set a value for each application.

- Output destination directory and maximum size of the SQL error report file

  Use the `pd_uap_exerror_log_dir` operand to set the output directory of the SQL error report file. Use the `pd_uap_exerror_log_size` operand to set the maximum size of the SQL error report file.

- Maximum data length of the parameter information output to the error log file or the SQL error report file

  Use the `pd_uap_exerror_log_param_size` operand or `PDUAPEXERLOGPRMSZ` to set the maximum data length of the parameter information output to the error log file or the SQL error report file. Specify the `pd_uap_exerror_log_param_size` operand to set a value for the entire HiRDB system, and specify `PDUAPEXERLOGPRMSZ` to set a value for each application.

(4) Interpreting SQL error information

(a) Output format of the SQL error report file

The output format of the SQL error report file is shown below.

Output format

```
** UAP ERROR INFORMATION aa...aa bbbbbbbbbbbbbbbbbbbbbbbbb ** [1]

* UAP INFORMATION * [2]
  UAP_NAME(cc...cc) USERID(dd...dd)
  IPADDR(ee...ee) CLTPID(ff...ff) THRDID(gg...gg)
  START_TIME(hhhhhhhhhhhhhhhhhhh)

* SERVER INFORMATION * [3]
  HOST(ii...ii) PORT(jj...jj) PLATFORM(kk...kk)
  SVRNAME(ll...ll) SVRPID(mm...mm)

* SQL INFORMATION * [4]
  OPTIMIZE_LEVEL(nn...nn) ADDITIONAL_OPTIMIZE_LEVEL(oo...oo)
  ISOLATION_LEVEL(pp...pp)

CNCTNO     SQL-       OP   SEC  SQL   SQL   OP   ERROR
           COUNTER    CODE NO   CODE  WARN  TION COUNTER
---------- ---------- ---- ---- ----- ----- ---- -----
rrrrrrrrrr  ssssssssss  tttt  uuuu  vvvvv  wwwww xxxx  yyyyy

START-TIME       END-TIME         EXEC-TIME
---------------- ---------------- -----------------
zzzzzzzzzzzzzzz  AAAAAAAAAAAAAAA  BB...BB

* SQL MESSAGE * [5]
  "CC...CC" [DD...DD]

* SQL STATEMENT * [6]
  "EE...EE"
```

```
* PARAMETER * [7]
*ELM NO= FFFFF*
*GGGGG* NO=HHHHH COD=III XDIM=JJJJJ SYS=KKKKK LEN=LLLLLLLLLLL
IND=MMMMMMMMMMM
          ARRAY NUM=NNNNN
        DATA=OO...OO
```

Explanation

1. Title of SQL error report file
2. UAP information
3. Server information
4. SQL information
5. SQL message
6. SQL statement
7. Parameter information

aa...aa

Displays the HiRDB version in the format shown below. (The maximum size of the displayed characters is 8 bytes.)

vv-*rr*-*zz*

If there is no -*zz* value, -*zz* is not output.

bbbbbbbbbbbbbbbbbbbbbbbbbb

Displays the date and time that the error information was output. The output format is shown below. (The maximum size of the displayed characters is 26 bytes.)

YYYY/*MM*/*DD* *hh*:*mm*:*ss*.*uuuuuu*

YYYY: Year

MM: Month

DD: Day

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

cc...cc

Displays the UAP name that was specified in the `PDCLTAPNAME` client environment definition. (The maximum size of the displayed characters is 30 bytes.)

dd...dd

Displays the authorization identifier of the connected user. (The maximum size of the displayed characters is 8 bytes.)

ee...ee

Displays the IP address of the UAP. (The maximum size of the displayed characters is 15 bytes.)

ff...ff

Displays the UAP process number. (The maximum size of the displayed characters is 10 bytes.)

If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

gg...gg

Displays the UAP thread number if the UAP is operating in multiple threads. (The maximum size of the displayed characters is 11 bytes.) If the UAP is not operating in multiple threads, `0` is displayed.

An incorrect number may be displayed if the correct thread number cannot be obtained. If the client version is 07-01 or earlier, `*` is displayed.

hhhhhhhhhhhhhhhhhhh

Displays the UAP execution time in the format shown below. (The maximum size of the displayed characters is 19 bytes.)

YYYY/*MM*/*DD* *hh*:*mm*:*ss*

YYYY: Year

MM: Month

DD: Day

hh: Hour

mm: Minute

ss: Second

ii...ii

Displays the name of the host in which the server process is operating. (The maximum size of the displayed characters is 30 bytes.)

jj...jj

Displays the communication port number of the server process. (The maximum size of the displayed characters is 5 bytes.)

kk...kk

Displays the platform supported by the client library. (The maximum size of the displayed characters is 6 bytes.)

For details about the output information, see the UAP operation platform in *10.1.1(2) Examining SQL trace information*. If the client version is 07-01 or earlier, `*` is output.

ll...ll

Displays the server name of the single server or the front-end server. (The maximum size of the displayed characters is 8 bytes.)

mm...mm

Displays the process number of the server process. (The maximum size of the displayed characters is 10 bytes.)

nn...nn

Displays the value of the SQL optimization option in decimal format. (The maximum size of the displayed characters is 10 bytes.)

oo...oo

Display the value of the SQL extension optimizing option in decimal format. (The maximum size of the displayed characters is 10 bytes.)

pp...pp

Displays the value of the data guarantee level. (The maximum size of the displayed characters is 10 bytes.)

rrrrrrrrrr

Displays the connection sequence number each time the server accepts `CONNECT`. (The maximum size of the displayed characters is 10 bytes.) The displayed connection sequence number is right-justified and padded with leading single-byte space characters.

ssssssssss

Displays the incremented SQL counter value each time an SQL statement is accepted. (The maximum size of the displayed characters is 10 bytes.) The displayed SQL counter value is right-justified and padded with leading single-byte space characters.

tttt

Displays the operation code for the SQL statement. (The maximum size of the displayed characters is 4 bytes.)

uuuu

Displays the section number of the SQL statement. (The maximum size of the displayed characters is 4 bytes.) The displayed section number is right-justified and padded with leading single-byte space characters. If an error occurs during execution of a control SQL, `****` is displayed.

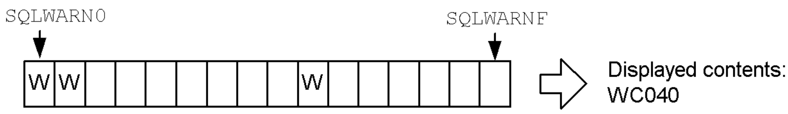
vvvvv

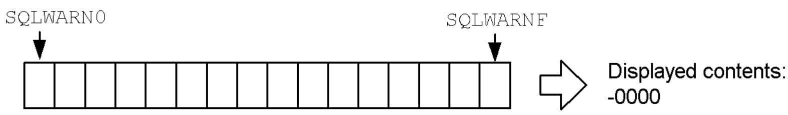
Displays the `SQLCODE` of the SQL execution result. (The maximum size of the displayed characters is 5 bytes.) The displayed `SQLCODE` is right-justified and padded with leading single-byte space characters.

wwwww

Displays warning information in hexadecimal format. (The maximum size of the displayed characters is 5 bytes.) In the warning information, one bit is assigned to each of the items `SQLWARN0` to `SQLWARNF`, starting from the left. If a warning flag is set to one of these items, the corresponding bit is set to `1`. If a warning flag is not set, the bit is set to `0`. All of these bits combined are output as a 4-digit hexadecimal value. If at least one warning flag is set, the 4-digit hexadecimal value is preceded by `W`. If no warning flag is set, the value is preceded by `-`. Examples are shown below.

Example:





xxxx

Displays information that the system uses. (The maximum size of the displayed characters is 4 bytes.)

If the first byte is `M`, it indicates that the inter-process memory communication facility is being used. The other three bytes represent maintenance information. However, if the client version is 07-01 or earlier,

**** is displayed.

yyyyy

Displays the error log number. (The maximum size of the displayed characters is 5 bytes.)

The output error log number is right-justified and padded with leading single-byte space characters. However, If the client version is 07-01 or earlier, ***** is displayed.

zzzzzzzzzzzzzzz

Displays the time that the SQL execution request was received from the client. The time is displayed in the format shown below. (The maximum size of the displayed characters is 15 bytes.)

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

AAAAAAAAAAAAAAA

Displays the time that processing of the client request ended. The time is displayed in the format shown below. (The maximum size of the displayed characters is 15 bytes.)

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

BB...BB

Displays the processing time of the client request in the format shown below. (The maximum size of the displayed characters is 17 bytes.) The displayed seconds value is right-justified and padded with leading single-byte space characters.

ssssssssss.*uuuuuu*

ssssssssss: Seconds

uuuuuu: Microseconds

CC...CC

Displays the message that was output during SQL execution. (The maximum size of the displayed characters is 254 bytes.)

DD...DD

Displays information that the system uses. (The maximum size of the displayed characters is 21 bytes.)

EE...EE

Displays the SQL statement. (The maximum size of the displayed characters is 2,000,000 bytes.)

If comments or SQL optimization specifications are described in the SQL statement, those are also displayed. If an error occurred during execution of a control SQL statement, `*` is displayed. For details about comments and SQL optimization specifications, see the manual *HiRDB Version 8 SQL Reference*.

FFFFF

Displays the affected element number if an error occurs in an SQL statement that uses an array. (The maximum size of the displayed characters is 5 bytes.)

GGGGG

Displays `INPRM` for input parameter information or `OUTRM` for output parameter information. For input/output parameter information, this variable displays `INPRM` for input information and `OUTRM` for output information. (The maximum size of the displayed characters is 5 bytes.)

HHHHH

Displays the parameter number. (The maximum size of the displayed characters is 5 bytes.)

III

Displays the data-type code. (The maximum size of the displayed characters is 3 bytes.)

JJJJJ

Displays the number of array elements. (The maximum size of the displayed characters is 5 bytes.)

KKKKK

Displays the area length of one element, including gaps. (The maximum size of the displayed characters is 5 bytes.)

LLLLLLLLLLL

Displays the data length. (The maximum size of the displayed characters is 11 bytes.)

MMMMMMMMMMM

Displays the indicator variable value. (The maximum size of the displayed characters is 11 bytes.)

NNNNN

Displays the number of elements in the repetition column if the SQL statement contains a repetition column. (The maximum size of the displayed characters is 5 bytes.) If the SQL statement does not contain a repetition column, this information is not displayed.

OO...OO

Displays parameter information. (The size of the displayed characters is the value specified for the `pd_uap_exerror_log_param_size` operand.) The types of parameter information are input parameter information, output parameter information, and input/output parameter information. The rules pertaining to parameter information are as follows:

- If an input parameter is a `BLOB`-type or `BINARY`-type locator, the value of the `BLOB`-type or `BINARY`-type locator is displayed.
- If the indicator variable is a negative value, only the portion up to `DATA=` is displayed.
- If there is information for several parameters, the parameter information is displayed in the sequence that the parameters were specified.
- If similar data extends beyond one line, `--- SAME` *x* `LINES ---` (*x* is the number of lines) is displayed.
- The size of the displayed parameter information is the sum of the actual data length and the data area length.
- For a repetition column, the number of elements in the repetition column is displayed in `ARRAY NUM`.
- For a repetition column, `DATA` is preceded by an indicator variable for each repetition element.
- For a repetition column, `DATA` is followed by the repetition column element number enclosed in parentheses.

(b) Output format of the error log file

Shown below is the output format of the error log file when the facility for output of extended SQL error information is used.

Output format

```
>  8355     0  8393     9 2005/08/12 14:06:30 KFPZ03000-I Error
information, type=CONNECT STATUS,
 inf=CLT=07-02(Aug  4 2005):WS SVR=07-02   US:WS LIBTYPE=UNIX_32
>  8355     0  8393    10 2005/08/12 14:06:30 KFPZ03000-I Error
information, type=SQL STREAM,
 inf=insert into t1 values ( ? , ? ,? )
>> 8355     0  8393    11 2005/08/12 14:06:30 SQLCODE:-404
47(140630218) sqaexp0.c   :2348 AUX
 KFPA11404-E Input data too long for column or assignment target
in variable 3
 UAP userprog1,hiuser01 [1]
 SVR host03,1146,sds,hp [2]
 SQLINF
1034,1,2,7,17,-0000,0000,14:06:30.216463,14:06:30.217765,0.001
302 [3]
 SQL INSERT INTO T1 VALUES(?,?,?) [4]
 PRM [5]
 INPRM 1,f1,1,0,4,0
       DATA=00 00 ff ff                              *....
*
 INPRM 2,c1,10,258,255,9
          ARRAY NUM=    9
     0 DATA(   0)=00 01 61                            *..a
*
     0 DATA(   1)=00 02 61 62                         *..ab
*
      0 DATA(    2)=00 03 61 62 63
*..abc           *
      0 DATA(    3)=00 04 61 62 63 64
*..abcd          *
      0 DATA(    4)=00 05 61 62 63 64 65
*..abcde         *
      0 DATA(    5)=00 06 61 62 63 64 65 66
*..abcdef        *
      0 DATA(    6)=00 07 61 62 63 64 65 66 67
*..abcdefg       *
      0 DATA(    7)=00 08 61 62 63 64 65 66 67 68
*..abcdefgh      *
      -1 DATA(    8)=
 INPRM 3,93,1,0,32002,0
        DATA=00 00 00 00 00 00 7d 02 41 41 41 41 41 41 41 41
*......}.AAAAAAAA*
             41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
*AAAAAAAAAAAAAAAA*
              --- SAME 14 LINES ---
```

Explanation

1. UAP information

UAP name

The name of the UAP that was specified in the `PDCLTAPNAME` client environment definition is displayed.

Authorization identifier

The authorization identifier of the connected user is displayed.

2. Server information

Host name

The name of the host in which the server process is operating is displayed.

Port number

The communication port number of the server process is displayed.

Server name

The server name of the single server or front-end server is displayed.

Platform

The platform supported by the client library is displayed.

For details about the displayed information, see the UAP operation platform in *10.1.1(2) Examining SQL trace information*. If the client version is 07-01 or earlier, `*` is displayed.

3. SQL information

SQL optimization option

The value of the SQL optimization option is displayed in decimal format.

SQL extension optimizing option

The value of the SQL extension optimizing option is displayed in decimal format.

Data guarantee level

The value of the data guarantee level is displayed.

Connection sequence number

The connection sequence number, which is incremented sequentially each time the server accepts `CONNECT`, is displayed.

Section number

The section number of the SQL statement is displayed.

SQLWARN

Warning information is displayed in hexadecimal format. In the warning information, one bit is assigned to each of the items `SQLWARN0` to `SQLWARNF`, starting from the left. If a warning flag is set to one of these items, the corresponding bit is set to `1`. If a warning flag is not set, the bit is set to `0`. All of these bits combined are output as a 4-digit hexadecimal value. If at least one warning flag is set, the 4-digit hexadecimal value is preceded by `W`. If no warning flag is set, the value is preceded by `-`. (For examples, see the explanation for *wwwww* in *(a) Output format of the SQL error report file*.)

System information

Information used by the system is displayed.

If the first byte is `M`, it indicates that the inter-process memory communication facility is being used. The other three bytes represent maintenance information. However, if the client version is 07-01 or earlier, `****` is displayed.

SQL start time

The time when the SQL execution request from the client was received is displayed in the following format:

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

SQL end time

The time when the process requested by the client ended is displayed in the following format:

hh:*mm*:*ss*.*uuuuuu*

hh: Hour

mm: Minute

ss: Second

uuuuuu: Microsecond

SQL runtime

The processing time of the client request is output in the following format. The displayed seconds value is right-justified and padded with leading single-byte space characters.

ssssssssss.*uuuuuu*

ssssssssss: Second

uuuuuu: Microsecond

4. SQL statement

SQL statement

The SQL statement is displayed.

If comments or SQL optimization specifications are described in the SQL statement, those are also displayed. The size of the displayed SQL statement is the value specified for `PDSQLTEXTSIZE` in the client environment definitions.

If an error occurs during execution of a control SQL statement, that SQL statement cannot be obtained and `*` is displayed instead.

For details about comments and SQL optimization specifications, see the manual *HiRDB Version 8 SQL Reference*.

5. Parameter information

ELM NO

If an error occurs in an SQL statement that uses an error, the number of that element is displayed.

Parameter information type

`INPRM` is displayed for input parameter information and `OUTRM` for output parameter information. For input/output parameter information, `INPRM` is displayed for input information and `OUTRM` for output information.

NO

The parameter number is displayed.

COD

The data type code is displayed.

XDIM

The number of array elements is displayed.

SYS

The area length of one element, including gaps, is displayed.

LEN

The data length is displayed.

IND

The value of the indicator variable is displayed.

ARRAY NUM

If the SQL statement contains a repetition column, the number of elements in the repetition column is displayed. If the SQL statement does not contain a repetition column, this information is not displayed.

DATA

Parameter information is displayed.

The types of parameter information are input parameter information, output parameter information, and input/output parameter information.

The rules pertaining to parameter information are as follows:

- If an input parameter is a `BLOB`-type or `BINARY`-type locator, the value of the `BLOB`-type or `BINARY`-type locator is displayed.
- If the indicator variable is a negative value, only the portion up to `DATA=` is displayed.
- If there is information for several parameters, the parameter information is displayed in the sequence that the parameters were specified.
- If similar data extends beyond one line, `--- SAME` *x* `LINES ---` (*x* is the number of lines) is displayed.
- The size of the displayed parameter information is sum of the actual data length and the data area length.
- For a repetition column, the number of elements in the repetition column is displayed in `ARRAY NUM`.
- For a repetition column, `DATA` is preceded by an indicator variable for each repetition element.
- For a repetition column, `DATA` is followed by the repetition column element number enclosed in parentheses.

(5) Rules for SQL error report files

The rules pertaining to the SQL error report files are described below. To view an SQL error report file, use a text editor or similar software.

1. HiRDB executes SQL statements, and each time it detects an error, it opens the SQL error report file, writes SQL error information, and then closes the file. Since

the SQL error information is appended to the final position of the SQL error report file, SQL error information accumulates in the file in chronological order.

2. Two SQL error report files are created (pduaperrlog1 and pduaperrlog2). If the size of the file to which data is currently being written exceeds the specified value of the pd_uap_exerror_log_size operand in the system definition, the output destination is switched to the other file. The system uses the two files alternately by performing this switching process on the other file as well. (After switching takes place, the contents of the previous file are deleted.) After HiRDB is started, the file that was most recently updated becomes the output destination.
3. When SQL processing ends, the SQL error report files are closed. Therefore, when an SQL statement is not being executed, you can use an OS command to back up or view the files. Even while an SQL statement is being executed, you can back up or view the file that is not the output destination.
4. To determine the SQL error report file that is being used currently, use the OS's dir command (ls -l command in UNIX) to check the most recent update dates of the files. The SQL error report file that was updated most recently is the current file.

(6) Notes

1. When the facility for output of extended SQL error information is used, time is required for executing a system call that retrieves the SQL start time and runtime, even if SQL error information is not output.
2. If the OS detects an error (such as a file system failure or an inappropriate file write privilege) while information is being output to an error log file or an SQL error report file, SQL error information is not output to the SQL error report file.
3. When the facility for output of extended SQL error information is used, memory space becomes necessary because parameter information is output.

10.1.4 UAP statistical report facility

The UAP statistical report facility outputs UAP statistical information during UAP execution to a UAP statistical report file.

(1) How to obtain the UAP statistical report

To obtain UAP statistical reports, specify values to PDCLTPATH, PDSQLTRACE, and PDUAPREPLVL in the client environment definitions.

The UAP statistical report facility creates two UAP statistical report files in the directory specified by PDCLTPATH.

To create the report files in a different directory from the one specified by PDCLTPATH, specify a value to PDREPPATH. Also, to open and close SQL trace files in CONNECT and DISCONNECT units, specify CNCT in PDSQLTRCOPENMODE.

For details about the individual client environment definitions, see *6.6 Client environment definitions (setting environment variables)*.

You can specify the information to be obtained using `PDUAPREPLVL` in the client environment definitions. Table 10-3 shows the relationship between the value of `PDUAPREPLVL` and the information to be obtained.

Table 10-3: Relationship between the value of PDUAPREPLVL and information to be obtained

Value of PDUAPREPLVL	**Information to be obtained**			
	By SQL	**By UAP**	**Access path information**[2]	**SQL runtime interim results**[2]
`s`[1]	Y	N	N	N
`u`	N	Y	N	N
`p`	N	N	Y	N
`r`	N	N	N	Y
`su`[1]	Y	Y	N	N
`sp`[1]	Y	N	Y	N
`sr`[1]	Y	N	N	Y
`up`	N	Y	Y	N
`ur`	N	N	Y	Y
`pr`	N	N	Y	Y
`sur`[1]	Y	Y	N	Y
`spr`[1]	Y	N	Y	Y
`upr`	N	Y	Y	Y
`a` or `sup`[1]	Y	Y	Y	Y

Y: Information is obtained.

N: Information is not obtained.

[1] If `s` is specified, SQL trace information is also obtained.

[2] When access path information or SQL runtime interim results are obtained, the

server's workload may increase because the system re-creates an SQL object even if the SQL object is found in the buffer.

■ **Size of an SQL trace file**

You can determine the size of an SQL trace file using the following formula:

Size of SQL trace file = 3208 + *A* + 80 × number of operations + total length of SQL statements (maximum of 4096) (bytes)

A: Total length of character strings specified in `PDHOST`, `PDFESHOST`, `PDSQLOPTLVL`, `PDADDITIONALOPTLVL`, `PDREPPATH`, and `PDTRCPATH` in the client environment definitions

To output information by SQL, information by UAP, access path information, and SQL runtime interim results, add the following sizes (bytes):

Information by SQL: 83* × *number of SQL statements*

Information by UAP: 2740* × *number of DISCONNECTs*

Access path information: See *(2)(b) Access path information*.

SQL runtime interim results: See *(2)(c) SQL runtime interim results*.

* This is the maximum value. The value changes according to the number of digits to be displayed.

■ **Notes**

1. If a program uses an API conforming to X/Open under OLTP, the facility does not output the information by the UAP.
2. The facility does not display the access path information if its size exceeds one gigabyte.
3. SQL runtime interim results are not output if the interim results during SQL execution exceed 1 gigabyte.
4. For a HiRDB/Parallel Server, the information by the UAP does not include the privilege checking executed at the connected dictionary server.
5. If you specify values that enable both the output of access path information and the inter-process memory facility (if you specify `PDIPC=MEMORY` in the client environment definitions), the `PDIPC` specification becomes invalid.
6. If you specify output of access path information or SQL runtime interim results, and use the inter-process memory communication facility (if you specify `PDIPC=MEMORY` in the client environment definitions), the `PDIPC` specification becomes invalid.

(2) Interpreting a UAP statistical report

The following shows a sample UAP statistical report, followed by explanations (a) through (d):

Output example

```
 CNCT  CLPID  CLTID  NO  OP    SEC  SQL   SQL    START-TIME    END-TIME      OP
 NO                      CODE  NO   CODE  WARN                               TION
 ----  -----  -----  --  ----  ---  ----  -----  ------------  ------------  ----
    1   9155      0   1  CNCT    0     0  WC040  16:03:55.720  16:03:58.080  0001
    1   9155      0   2  AUI2    1     0  -0000  16:03:58.630  16:03:59.400  0000

*SQL*  INSERT INTO T1(C1,C2,C3,C4,C5,C6) VALUES(?,?,?,?,?,?)
00:00:00.770    00:00:00.430000    340    1    0    0    0    0    0 ........(a)
    [1]               [2]           [3]   [4]  [5]  [6]  [7]  [8]  [9]

    1   9155      0   3  SET     2     0  -0000  16:04:00.820  16:04:01.540  0000


*SQL*  SELECT * from T1, T2, T3 where ((T1.C1='a' and T1.C2='A')
 or (T1.C1='a' and T1.C2='B')) and T1.C1=T2.C1 and T1.C2=T2.C2 and T2.C3>=1995
 and T1.C1=T3.C1 and T1.C2=T3.C2 order by T1.C1

00:00:00.720    00:00:00.240000    480    1     0    0    0

Result of SQL Optimizer :   .........................................(b)
 Connect No     :   1

 --------------------------------------------------------------------------------
 Section No     : 2
 UAP Source     :XXXXXXXX.ec
 Optimize Mode  : COST_BASE_2
 SQL Opt Level  : 0x00000420(1056) = "PRIOR_NEST_JOIN"(32),"RAPID_GROUPING"(1024)
 Add Opt Level  : 0x00000003(3) = "COST_BASE_2"(1),"APPLY_HASH_JOIN"(2)
 Work Table     : 0
 Table Cost     : 12672.66944
 ----- QUERY EXPRESSION BODY ID : 1 -----
              :
 ----- QUERY ID : 1 -----
              :
 JOIN
   :
 SCAN
   :

 --------------------------------------------------------------------------------

    1   9155      0   4  OPEN    2     0  -0000  16:04:02.090  16:04:02.800  0000
```

```
Result of SQL Execution :   ........................................(c)
--------------------------------------------------------------------------------
Connect No     :    1
UAP Source     : XXXXXXXX.ec
Section No     : 2
----- QUERY EXPRESSION BODY ID : 1 -----
   :
----- QUERY ID : 1 -----
   :
JOIN
   :
SCAN
   :
--------------------------------------------------------------------------------


    1   9155      0   9  DISC    0     0  -0000  16:05:55.110  16:05:56.660  0004

UAP INFORMATION:   ................................................(d)
 [1]UAPNAME()
 [2]SVHOST(dcm3500) [3]SVPORT(4439)  [4]SVNAME(fes1)   [5]CNCTNO(1)
 [6]SVPID(8945)     [7]CLPID(9155)     [8]CLTTID(0)
 [9]WAITT(0)       [10]CTIME(0)
 [11]ROREQ(0)      [12]ROHITS(0)

 [13]SOREQ(10)     [14]SOHITS(3)   [15]SOCRT(0)       [16]SOMAX(0)
 [17]COMT(0)       [18]ROLB(0)     [19]FROW(0)    [20]DROW(0)     [21]IROW(3)
 [22]UROW(0)       [23]SET(1)      [24]OPEN(2)    [25]FETC(1)     [26]CLOS(0)
 [27]DESC(0)       [28]SEL(1)      [29]INS(3)     [30]UPD(0)      [31]DEL(0)
 [32]LOCK(0)       [33]CRTT(0)     [34]DRPT(0)    [35]ALTT(0)     [36]CRTI(0)
 [37]DRPI(0)      [38]CMTT(0)      [39]CMTC(0)    [40]CRTS(0)     [41]DRPS(0)
 [42]GRTR(0)      [43]GRTS(0)      [44]GRTA(0)    [45]GRTC(0)     [46]GRTD(0)
 [47]RVKR(0)      [48]RVKS(0)      [49]RVKA(0)    [50]RVKC(0)     [51]RVKD(0)
 [52]CRTV(0)      [53]DRPV(0)      [54]PRGT(0)    [55]CRTP(0)     [56]DRPP(0)
 [57]ALTP(0)      [58]CALL(0)      [59]DESI(0)    [60]MISC(0)
 [61]MAXIO(0)     [62]MAXIOM(0)    [63]MINIO(0)   [64]MINIOM(0)
 [65]IOTIM(0)     [66]IOTIMM(0)

 [67]DIDRC(0)     [68]DIDUC(0)     [69]DIDHC(0)   [70]DIDRD(0)    [71]DIDWT(0)
 [72]LBRFC(0)     [73]LBUPC(0)     [74]LBRHC(0)   [75]LBUHC(0)    [76]LBRDC(0)
 [77]LBWTC(0)     [78]BFSHC(2320)  [79]BRDWC(0)   [80]BWTWC(50)
 [81]BLKWC(2)     [82]MWFN(0)      [83]MWFEC(0)   [84]MWFVL(0)
 [85]WFRDC(0)     [86]WFWTC(0)     [87]WBFOC(0)
 [88]MWHTS(0)     [89]MBSL1(0)     [90]MBSL2(0)   [91]MBSL3(0)
 [92]SCHSKD(0)    [93]SCHCHG(0)
 [94]CINSM(0)     [95]CAFLS(0)     [96]CAFWR(0)   [97]CFMAX(0)    [98]CFAVG(0)
 [99]LDIRC(0)    [100]LDIUC(0)    [101]LDIHC(0)  [102]LDIRD(0)
[103]LDIWT(0)    [104]LBFSHC(0)
[105]ARREQ(0)    [106]ARWC(0)     [107]ARWT(0)   [108]ARWTM(0)
[109]ARWTA(0)    [110]ARWTMA(0)   [111]ARSTA(0)  [112]ARSTMA(0)
[113]HJMAX(0)    [114]HJCMC(0)    [115]HJHTC(0)
```

(a) Information by the SQL

1. SQL execution time (milliseconds)

Displays the SQL execution time in the format *HH*:*MM*:*SS*.*mmm*. If `YES` is specified in the `PDSQLEXECTIME` client environment definition, the unit becomes microseconds.

2. SQL execution time at server (microseconds)

   Displays the SQL execution time at the server in the format *HH*:*MM*:*SS*.*mmmmmm*.

3. Difference between 1 and 2 (milliseconds)

   Provides a guideline for communication time. If `YES` is specified in the `PDSQLEXECTIME` client environment definition, the unit becomes microseconds.

4. Number of processed rows

   Displays the number of rows processed by the SQL statements that were issued during the session.

5. Work table creations count

   Displays the number of times a work table was created during internal processing for the SQL statements that were issued during the session.

6. Work table deletions count

   Displays the number of times a work table was deleted during internal processing for the SQL statements that were issued during the session.

7. SQL object size (bytes)

   Displays the size of the SQL object created by the SQL statements that were issued during the session.

8. Total comparison count during hash table search processing by hash join, subquery hash execution

   Displays the total number of comparisons that the SQL statements issued during this connection perform on the data having the same hash value during the hash table search.

9. Total hash join search count during hash join, subquery hash execution

   Displays the number of times that the hash table was searched by the SQL statements issued during this connection.

(b) Access path information

A UAP statistical report displays access path information. `Connect No` displays the connect number. By executing an upward search based on the connect number, you can identify the SQL statements displayed in the SQL trace information. You can also use the connect number to find out the execution request start and end times of the SQL statements displayed in the SQL trace information. For dynamic SQL, execute a downward search based on the connect number, while for static SQL, execute an upward search. If you specify information acquisition in SQL units, the SQL execution times are also displayed. If you find an SQL statement that has a long SQL execution time, tune the UAP.

The UAP statistical report facility does not include the following information in the access path information: HiRDB version, number of back-end servers, UAP name, authorization identifier, SQL optimization processing time, and SQL statements. However, if the routine contains data manipulation SQL statements, the facility displays them as the SQL statements.

If the access path is `SELECT-APSL` for a HiRDB/Single Server (access path is to be selected from multiple candidates by the boundary value during execution), the facility displays the boundary value at the beginning, followed by multiple candidates separated by `Section No`.

For details about the access path information, see the access path display utility in the manual *HiRDB Version 8 Command Reference*.

■ **Notes**

1. The facility does not display the access path information for a Java routine.
2. For an SQL routine, the facility displays the access path information if the SQL object's index information becomes invalid due to an index addition or deletion made to the table used within the routine.
3. The access path information increases the size of the SQL trace file. You can determine this increase in size using the following formula. This is just a guideline; the actual size of the access path information depends on the table definitions, index definitions, and SQL statement used.

$$\text{Size of access path information} = 1 + 0.1 \times \text{Number of set operations} + 4 \times \sum_{i=1}^{n}(S_i) \quad \text{(KB)}$$

n: Number of query specifications in SQL statement
Si: Number of tables in query specification i
Note
For a query in the routine, add the length of the SQL statement to the result of this formula.

(c) SQL runtime interim results

A UAP statistical report displays SQL runtime interim results.

When SQL runtime interim results are displayed, the information listed below can be checked. (The number of rows displayed in the results is the number of rows that HiRDB actually processed at the stage that the interim results are displayed.)

- Number of rows fetched from the table

- Number of rows narrowed by the index
- Number of rows in the results for each join
- Number of input/output rows for any duplicate exclusion, `GROUP BY`, `ORDER BY`, or `LIMIT` specified in the query and number of rows in the query results
- Number of rows in the results for each set operation

Use the SQL runtime interim results and the access path information to carry out SQL tuning. For details about using access path information for SQL tuning, see the description of the access path display utility in the manual *HiRDB Version 8 Command Reference*.

Output format

```
 -------------------------------------------------------
Connect No     : aa...a
UAP Source     : bb...b
Section No     : cc...c
 ----- QUERY EXPRESSION BODY ID : ... -----   .............1
  :
 ----- QUERY ID : ... -----   ............................2
  :
 JOIN   ..................................................3
  :
 SCAN   ..................................................4
  :
```

Explanation

1. Set operation process information

   For details about set operation process information, see *Set operation process information*.

2. Query process information

   For details about query process information, see *Query process information*.

3. Join process information

   For details about join process information, see *Join process information*.

4. Base table search process information

   For details about base table search process information, see *Base table search process information*.

aa...a

Displays the connection sequence number.

bb...b

Displays the UAP source file name.

cc...c

Displays the section number (number for checking the SQL correspondence).

The information after `Connect No` is repeated for each SQL statement. By conducting a search using a connection sequence number and a section number, you can identify correspondences with the SQL statements displayed in SQL trace information and the access path information.

- Set operation process information

```
----- QUERY EXPRESSION BODY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
Order by       : ee...e ROWS
SetOpe Process : ff...f = gg...g ROWS <-- hh...h ii...i hh...h
                      :
```

Explanation

aa...a

Displays the query express body ID.

An ID number is assigned to each query expression body that includes a set operation. If the SQL statement consists of multiple query expression bodies, this line is used to separate the information displayed for each query expression body.

When *(b) Access path information* is being displayed, this value corresponds to the query expression body ID displayed in the access path information.

bb...b

Displays the number of rows in the results of the query expression.

cc...c `ROWS <--` *dd...d* `ROWS`

Displays the final number of rows for the process (`LIMIT` process) that gets search results for the maximum number of rows to return.

If `LIMIT` clause is not specified, this line is not displayed.

cc...c

Displays the number of output rows in the `LIMIT` process.

dd...d

Displays the number of input rows in the `LIMIT` process.

ee...e

Displays the number of rows of the sort process (`ORDER BY` process).

This line is not displayed if any one of the following conditions applies:

- An `ORDER BY` clause is not specified.
- The sort processing specified in the `ORDER BY` clause is omitted.
- A `LIMIT` clause is specified.

ff...f = *gg...g* `ROWS <--` *hh...h* *ii...i* *hh...h*

Displays the number of rows in the results of the set operation.

If multiple set operations are specified, the information is displayed over several lines.

If the facility that executes partitioned scanning of `UNION ALL` is applied (this facility returns the search results of each query in succession without creating a work table), this line is not displayed.

ff...f

Displays the set operation number of the set operation results in the format `LID(`*set-operation-number*`)`.

If access path information is being displayed, this corresponds to the set operation number displayed in the access path information.

gg...g

Displays the number of rows in the set operation results.

hh...h

If the query expression body to be operated is a query specification, this information is displayed in the format `QID(`*query-ID*`)`. If the query expression body to be operated is the joined result of multiple query specifications, `LID(`*set-operation-number*`)` is displayed.

ii...i

Displays the set operation type (`UNION`, `UNION ALL`, `EXCEPT`, or `EXCEPT ALL`). The *hh...h* values before and after this value form the query expression body.

- Query process information

```
----- QUERY ID : aa...a -----
Query          : bb...b ROWS
Limit          : cc...c ROWS <-- dd...d ROWS
Order by       : ee...e ROWS
Distinct       : ff...f ROWS <-- gg...g ROWS
Having         : hh...h ROWS
Group by       : ii...i ROWS <-- jj...j ROWS
```

Explanation

aa...a

Displays the query ID.

A number is assigned to each query specification. If the SQL statement consists of multiple query specifications, this line is used to separate the information displayed for each specification.

If access path information is being displayed, this value corresponds to the query ID displayed in the access path information.

bb...b

Displays the number of rows in the query results.

cc...c `ROWS <--` *dd...d* `ROWS`

Displays the final number of rows for the process (`LIMIT` process) that gets the search results for the maximum number of rows to return.

If `LIMIT` is not specified, this line is not displayed.

cc...c

Displays the number of output rows in the `LIMIT` process.

dd...d

Displays the number of input rows in the `LIMIT` process.

ee...e

The number of rows in sort processing (`ORDER BY` processing) is displayed. Note that `ORDER BY` processing may be executed implicitly even if an `ORDER BY` clause is not specified.

This line is not displayed if any one of the following conditions applies:

- An `ORDER BY` clause is not specified.
- The sort processing specified in the `ORDER BY` clause is omitted.

- `ORDER BY` processing is not executed implicitly.
- A `LIMIT` clause is specified.

ff...f `ROWS <--` *gg...g* `ROWS`

Displays the number of rows processed by duplicate exclusion. Note that duplicate exclusion may be executed implicitly even if duplicate exclusion is not specified.

This line is not displayed if any one of the following conditions applies:

- Duplicate exclusion is not specified.
- Duplicate exclusion is not executed implicitly.
- A `LIMIT` clause is specified.

ff...f

The number of output rows in duplicate exclusion processing is displayed.

gg...g

The number of input rows in duplicate exclusion processing is displayed.

hh...h

Displays the number of rows after the `HAVING` clause is evaluated.

If a `HAVING` clause is not specified, this line is not displayed.

ii...i `ROWS <--` *jj...j* `ROWS`

Displays the number of rows processed by grouping (including implicit grouping).

If grouping is not executed, this line is not displayed.

ii...i

Displays the number of output rows in grouping.

jj...j

Displays the number of input rows in grouping.

■ Join process information

```
JOIN
 # Join ID      : aa...a
   Row Count    : bb...b ROWS
   Left         : cc...c ROWS
   Right        : dd...d ROWS
   Join Type    : ee...e (ff...f)
```

Explanation

aa...a

Displays the join process ID.

An ID number is assigned to each join process unit, and if there are multiple join processes, the processes are separated with this line.

If access path information is being displayed, this value corresponds to the join process ID displayed in the access path information.

bb...b

Displays the number of rows in the join process results.

cc...c

Displays the number of rows that were fetched from the join partner on the left side.

dd...d

Displays the number of rows that were fetched from the join partner on the right side.

ee...e

- For HiRDB/Single Server and for HiRDB/Parallel Server when the join method is not determined dynamically during SQL execution

  Displays the join process type (`MERGE JOIN`, `NESTED LOOPS JOIN`, `CROSS JOIN`, or `HASH JOIN`).

- For HiRDB/Parallel Server when the join method is determined dynamically during SQL execution

  Displays `SELECT-APSL` as the join process type.

ff...f

Displays the execution type of the join process (`INNER`, `LEFT OUTER`, `EXIST`, `NOT EXIST`, `ALL`, or `VALUE`).

■ Base table search process information

- When no index or only one index is used in the search process

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
   Index Name  : dd...d 0xeeeeeeee(ee...e)
                  Search    : ff...f gg...g
                  Key       : hh...h gg...g
```

Explanation

aa...a(*aa...a*)

Displays the name of the table to be searched and the correlation name (in parentheses). If a correlation name is not being used, the correlation name (in parentheses) is not displayed. If there are several search processes, this line is used to separate the information displayed for each search.

0x*bbbbbbbb*(*bb...b*)

Displays the ID of the table to be searched in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows fetched from the base table.

dd...d

Displays the index name to be used in the search.

This line is not displayed in the following cases:

- The search is performed without the use of an index.
- HiRDB/Parallel Server dynamically determines the search method during SQL execution.

0x*eeeeeeee*(*ee...e*)

Displays the ID of the index used in the search. The ID is displayed in hexadecimal and decimal (in parentheses) formats.

ff...f

Displays the number of rows in the results narrowed by the search condition.

When an index is used in the search, the number of rows that make up the index is displayed, even if there is no search condition.

This line is not displayed when the surrogate facility for plug-in indexes is used to determine the results of a set function.

gg...g

Displays `ELEMENTS` for an index that contains a repetition column and `ROWS` for all other cases.

hh...h

Displays the number of rows in the results narrowed by the key condition.

If there is no key condition, this line is not displayed.

- When multiple indexes are used in the search process

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
   Index Name  : dd...d = ee...e 0xffffffff(ff...f)
                       Search    : gg...g hh...h
                       Key       : ii...i hh...h
                       Row Count : jj...j ROWS
                 dd...d = ee...e 0xffffffff(ff...f)
                       Search    : gg...g hh...h
                       Key       : ii...i hh...h
                       Row Count : jj...j ROWS
                 dd...d = kk...k ROWS <-- ll...l mm...m ll...l
```

Explanation

aa...a(*aa...a*)

Displays the name of the table to be searched and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed. If there are multiple search processes, this line is used to separate the information displayed for each process.

`0x`*bbbbbbbb*(*bb...b*)

Displays the ID of the table to be searched in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows fetched from the base table.

dd...d

Displays the number of the work table created when `AND PLURAL INDEXES SCAN`* is executed. The work table number is displayed in the `LID(`*work-table-number*`)` format.

If access path information is being displayed, this value corresponds to the work table number displayed in the access path information.

ee...e

Displays the name of the index used to create the work table when `AND PLURAL INDEXES SCAN`* or `OR PLURAL INDEXES SCAN`* is executed. The index name is displayed in multiple lines. However, if a work table is created without the use of an index, `(NO USE)` is displayed as the index name.

`0x`*ffffffff* `(`*ff...f*`)`

Displays the index IDs used in the search. The IDs are displayed in hexadecimal and decimal (in parentheses) formats.

gg...g

Displays the number of rows in the results narrowed by the search condition.

Even if there is no search condition, the number of rows that make up the index is displayed when a search using an index is executed.

hh...h

Displays `ELEMENTS` for an index that contains a repetition column and `ROWS` for all other cases.

ii...i

Displays the number of rows in the results narrowed by the key condition.

If there is no key condition, this line is not displayed.

jj...j

Displays the number of rows fetched from the base table.

dd...d `=` *kk...k* `ROWS <--` *ll...l* *mm...m* *ll...l*

Displays the creation sequence of the work tables created when `AND PLURAL INDEXES SCAN`* is executed. When three or more indexes are used in the search process, this information is displayed in multiple lines.

kk...k

Displays the number of rows in the operation results.

ll...l

Displays the work table that becomes the input for the operation. The work table is displayed in the `LID(`*work-table-number*`)` format.

mm...m

Displays the operation type (`AND`, `OR`, or `ANDNOT`) performed on the work tables.

* For details about `AND PLURAL INDEXES SCAN` and `OR PLURAL INDEXES SCAN`, see the description of the access path display utility in the manual *HiRDB Version 8 Command Reference*.

- When a work table is created for retrieving the results of a view table

```
SCAN
 # Table Name  : aa...a(aa...a) 0xbbbbbbbb(bb...b)
   Row Count   : cc...c ROWS
```

Explanation

aa...a(*aa...a*)

Displays the view name and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed.

0x*bbbbbbbb*(*bb...b*)

Displays the view ID in hexadecimal and decimal (in parentheses) formats.

cc...c

Displays the number of rows that were fetched from the table.

- When a work table is created for the `WITH` clause

```
SCAN
 # Table Name  : aa...a(aa...a)
   Row Count   : bb...b ROWS
```

Explanation

aa...a(*aa...a*)

Displays the `WITH` clause query name and the correlation name (in parentheses).

If a correlation name is not being used, the correlation name (in parentheses) is not displayed.

bb...b

Displays the number of rows that were fetched from the table.

- When a work table is created for the derived table specified in the `FROM` clause

```
SCAN
 # Table Name  : aa...a(aa...a)
   Row Count   : bb...b ROWS
```

Explanation

aa...a`(`*aa...a*`)`

Displays `(NO NAME)` or `(NO NAME)(`*correlation-name*`)`.

bb...b

Displays the number of rows that were fetched from the table.

- When a work table that HiRDB creates internally is searched

```
SCAN
 # Table Name  : aa...a
   Row Count   : bb...b ROWS
```

Explanation

aa...a

Displays the name of the work table that HiRDB created internally.

The name of the work table that HiRDB created internally is displayed in `(DUMMY` *work-table-number*`)` format.

The work table number is a three-digit integer.

bb...b

Displays the number of rows fetched from the work table that HiRDB created internally.

- When the query results for an external server are retrieved

```
SCAN
 # Table Name  : aa...a
   Row Count   : bb...b ROWS
```

Explanation

aa...a

Displays the name of the table identifier that is created internally so that the local HiRDB can access results fetched from an external server. The table identifier name is displayed in `(FOREIGNSQL` *table-number*`)` format.

bb...b

Displays the number of rows that were fetched from the external server.

Notes

1. SQL runtime interim results are displayed when of one of the following SQL statements is executed:

   - Definition SQL[1]
   - `ASSIGN LIST` statement[5]
   - `CLOSE` statement
   - `DELETE` statement[6]
   - `EXECUTE` statement[1]
   - `EXECUTE IMMEDIATE` statement [2]
   - `INSERT` statement[3, 6]
   - `PREPARE` statement[4]
   - `PURGE TABLE` statement[1]
   - Single-row `SELECT` statement
   - `UPDATE` statement[6]
   - `COMMIT` statement[1]
   - `DISCONNECT` statement[1]
   - `ROLLBACK` statement[1]
   - If an error that has implicit rollback occurs[1]

   [1] SQL runtime interim results are displayed if there is a cursor that has not been closed.

   [2] SQL runtime interim results are displayed for the following SQL statements:

   - `ASSIGN LIST` statement
   - `DELETE` statement
   - `INSERT` statement

- UPDATE statement

[3] SQL runtime interim results are displayed when a scalar subquery or a query specification is specified in the VALUES clause.

[4] If YES is specified in the PDPRPCRCLS client environment definition and an SQL identifier being used by an open cursor is reused by a PREPARE statement, the SQL runtime interim results of the open cursor are displayed.

[5] SQL runtime interim results are not displayed when FOR ALTER LIST is specified.

[6] SQL runtime interim results are not displayed when a foreign table is specified as the table target.

2. SQL runtime interim results are not displayed for an SQL statement described in a stored procedure, even if the CALL statement is executed.
3. SQL runtime interim results are not displayed for a trigger SQL statement described in a trigger, even if the trigger is executed.
4. When HiRDB/Parallel Server is used, the total number of rows of all servers is displayed.
5. The displayed number of rows may not be an accurate value.
6. When SQL runtime interim results are displayed, the size of the SQL trace file increases by the size shown in the expression below. Note this increase when estimating the size of the SQL trace file. However, the size of the interim results varies significantly depending on the table definitions, the index definitions, and the SQL statements. The value estimated with the following expression should be used only as a rough guideline.

Size of SQL runtime interim results

$$= 0.8 + 0.1 \times \text{set-operation-count} + 0.9 \times \sum_{i=1}^{n} (Si) \quad \text{(kilobytes)}$$

(d) Information by the UAP

1. UAP name

   This is the name of the UAP for which statistical information was edited.

2. Host name

   This is the name of the host at the connected server.

3. Port number

This is the port number at the connected server.

4. Connected server name

   This is the name of the front-end server or single server that was connected.

5. Connection sequence number

   This is the sequence number assigned by the server each time `CONNECT` is accepted.

6. Server process number

   This is the connected server's process number.

7. Client process number

   This is the UAP's process number.

8. Client's thread number

   This is the thread number of the UAP that is running in multi-thread.

9. Lock release wait time (milliseconds)[1]

   This is the length of time during which a lock acquisition request in the server was placed on lock release wait status because another user locked the requested resource.

10. CPU time (milliseconds)[1]

    This is the CPU time at the server that was used by transaction during UAP execution.

11. Stored procedure's SQL object acquisition requests count

    This is the number of times a stored procedure's SQL object acquisition request was issued for the SQL object buffer at the single server or front-end server.

12. Stored procedure object buffer hits count

    This is the number of times requested information was found in the SQL object buffer at the single server or front-end server.

13. SQL object acquisition requests count

    This is the number of times an SQL object acquisition request was issued for the SQL statements issued during the session.

14. SQL object buffer hits count

    This is the number of times requested information was found in the SQL object buffer for the SQL statements issued during the session.

15. SQL object creations count

This is the number of times an SQL object was created for the SQL statements issued during the session.

16. Maximum size of SQL object created (bytes)

    This is the maximum size of the SQL object created with the SQL statements issued during the session.

17. `COMMIT` statement executions count during the session.
18. `ROLLBACK` statement executions count during the session.
19. Number of retrieval rows passed to UAP by the `FETCH` and `SELECT` statements during the session.
20. Number of rows deleted by the `DELETE` statements during the session.
21. Number of rows inserted by the `INSERT` statements during the session.
22. Number of rows updated by the `UPDATE` statements during the session.
23. Preprocessing time during the session.
24. `OPEN` statement executions count during the session.
25. `FETCH` statement executions count during the session.
26. `CLOSE` statement executions count during the session.
27. `DESCRIBE` statement executions count during the session.
28. `SELECT` statement executions count during the session.
29. `INSERT` statement executions count during the session.
30. `UPDATE` statement executions count during the session.
31. `DELETE` statement executions count during the session.
32. `LOCK` statement executions count during the session.
33. `CREATE TABLE` executions count during the session.
34. `DROP TABLE` executions count during the session.
35. `ALTER TABLE` executions count during the session.
36. `CREATE INDEX` executions count during the session.
37. `DROP INDEX` executions count during the session.
38. `COMMENT (TABLE)` executions count during the session.
39. `COMMENT (COLUMN)` executions count during the session.
40. `CREATE SCHEMA` executions count during the session.
41. `DROP SCHEMA` executions count during the session.

42. `GRANT RDAREA` executions count during the session.
43. `GRANT SCHEMA` executions count during the session.
44. `GRANT` access privilege executions count during the session.
45. `GRANT CONNECT` executions count during the session.
46. `GRANT DBA` executions count during the session.
47. `REVOKE RDAREA` executions count during the session.
48. `REVOKE SCHEMA` executions count during the session.
49. `REVOKE` access privilege executions count during the session.
50. `REVOKE CONNECT` executions count during the session.
51. `REVOKE DBA` executions count during the session.
52. `CREATE VIEW` executions count during the session.
53. `DROP VIEW` executions count during the session.
54. `PURGE TABLE` statement executions count during the session.
55. `CREATE PROCEDURE` executions count during the session.
56. `DROP PROCEDURE` executions count during the session.
57. `ALTER PROCEDURE` executions count during the session.
58. `CALL` statement executions count during the session.
59. `DESCRIBE` statement (`INPUT`) executions count during the session.
60. Other SQL executions count during the session.
61. Maximum input/output time (seconds).
62. Maximum input/output time (microseconds).
63. Maximum input/output time (seconds).
64. Maximum input/output time (microseconds)

    Check whether the input and output times are appropriate. If input/output processing takes longer than necessary, obtain and check the hardware log for any hardware errors.

    If you used the asynchronous READ facility, the input and output times for batch look-ahead reading by the asynchronous READ process are not included.
65. Cumulative input/output time for database (seconds).
66. Cumulative input/output time for database (microseconds)

    Use this information to determine whether the cause is input/output or CPU.

If you used the asynchronous READ facility, the input and output times for batch look-ahead reading by the asynchronous READ process are not included.

67. Data, index, and directory page references count

    This is the number of times a data, index, or directory page was referenced from this UAP.

68. Data, index, and directory page updates count

    This is the number of times a data, index, or directory page was updated from this UAP.

69. Data, index, and directory page buffer hits count

    This is the number of times a requested data, index, or directory page was found in the buffer. If the hit rate ((item 69 ÷ item 67) × 100) is low, obtain the global buffer statistical information and tune the global buffer with a low hit rate. In this case, all global buffers are subject to tuning except for the LOB global buffer.

70. Data, index, and directory page real READs count

    This is the number of times a data, index, or directory page was actually read by this UAP.

    If you are using the prefetch facility, the number of look-ahead READs by the prefetch facility is included. If you used the asynchronous READ facility, the number of look-ahead READs by the asynchronous READs process is also included.

    If the buffer hit rate is low, the READs count becomes high.

71. Data, index, and directory page real WRITEs count

    This is the number of times a data, index, or directory page was actually written by this UAP. If the commit output facility is used, this count includes the number of outputs to the database during commit processing.

72. LOB page references count

    This is the number of times a LOB page was referenced by this UAP. This count includes the LOB data and plug-in retrieval operations.

73. LOB page updates count

    This is the number of times a LOB page was updated by this UAP. This count includes the LOB data and plug-in update operations.

74. LOB page reference buffer hits count

    This is the reference buffer hits count. This information is applicable if the LOB global buffer is used. If the hit rate ((item 74 ÷ item 72) × 100) is low, obtain the global buffer statistical information and tune the global buffer with a low hit rate.

In this case, the LOB global buffer is subject to tuning. If the LOB global buffer is not used, the hit rate is 0.

75. LOB page update buffer hits count

    This is the update buffer hits count. This information is applicable if the LOB global buffer is used. If the hit rate (item 75 ÷ item 73 × 100) is low for LOB data updating or plug-in index updating, obtain the global buffer statistical information and tune the global buffer with a low hit rate. In this case, the LOB global buffer is subject to tuning. If the LOB global buffer is not used, the hit rate is 0. Update buffer hits are not applicable to addition of new LOB data.

76. LOB page real READs count

    This is the number of times a LOB page was actually read by this UAP. If the LOB global buffer is used and the READ buffer hit rate is low, the READs count becomes high.

77. LOB page real WRITEs count

    This is the number of times a LOB page was actually written by this UAP. When updating a plug-in index, you can reduce the real WRITEs count by using the LOB global buffer.

78. Global buffer flushes count

    This is the number of times the buffer was flushed to create space for a new page. This indicates the number of times a page was swept out of memory because the buffer was full.[2]

79. Global buffer READ waits count

    This is the number of times the UAP was placed on wait status because a page in the global buffer was being read from a HiRDB file by another user. This indicates the number of times the UAP was placed on wait status until a READ operation was completed because the page to be referenced was under READ operation by another user.[2]

80. Global buffer WRITE waits count

    This is the number of times the UAP was placed on wait status because a page in the global buffer was being output to a HiRDB file by another user. This indicates the number of times the UAP was placed on wait status until a WRITE operation was completed because the page to be updated was under WRITE operation by another user.[2]

81. Global buffer lock release waits count

    This is the number of times the UAP was placed on wait status because a page in the global buffer was in use by another user. This indicates the number of times the UAP was placed in wait status until update processing was completed because the page to be referenced or updated was under update processing by another UAP.[2]

82. Maximum work table files count

    This is the maximum number of work table files used by this UAP.[3] You can determine the validity of the `-l` option value (maximum number of files) specified in the `pdfmkfs` command. The value of the `-l` option must satisfy the following condition:[4]

    Value of `-l` option ≥ total number of work table files for all UAPs that are executed concurrently + 20

83. Maximum work table file extensions count

    This is the maximum number of work table file extensions for this UAP. You can determine the validity of the `-e` option value (maximum number of extensions) specified in the `pdfmkfs` command. The value of the `-e` option must satisfy the following condition:[4]

    Value of `-e` option ≥ total number of work table file extensions for all UAPs that are executed concurrently

84. Maximum size of work table file (MB)

    This is the maximum size of a work table file for this UAP. You can determine the validity of the `-n` option value (maximum number of extensions) specified in the `pdfmkfs` command. The value of the `-n` option must satisfy the following condition:[4]

    Value of `-n` option ≥ total size of work table files for all UAPs that are executed concurrently + management area size for HiRDB file system area

85. Work table file READs count

    This is the number of times work table data was input from file to buffer.[1]

86. Work table file WRITEs count

    This is the number of times work table data was output from buffer to file.[1]

87. Forced outputs count for the work table buffer

    This is the number of times buffer contents in use were forcibly output to a file due to a shortage of the work table buffer.[1] If this value is not 0, increase the value of the `pd_work_buff_size` operand (size of the work table buffer) in the system definitions.

88. Estimated value for expanding hash table in batch mode (KB)

This is the estimated size of the hash table required to expand the processed hash data in batch mode during hash join or subquery hash execution.[3]

If the size of the hash table is greater than this value, batch hash join is assumed, which does not involve any packet division.[5] If this value exceeds the specified range of the hash table size, batch hash join is not possible. If this value is 0, hash join or subquery hash execution has not taken place.

89. Maximum packet size at level 1 (KB)

    This is the maximum packet size after level 1 packet division during hash join or subquery hash execution.[3]

    If the size of the hash table is at least this value, packet division was completed at level 1. If the packet division level is 2 or more, you can complete the packet division at level 1 by specifying this value as the hash table size.[6] For a batch hash join that does not involve any packet division, this value is 0.

90. Maximum packet size at level 2 (KB)

    This is the maximum packet size after level 2 packet division during hash join or subquery hash execution.[3]

    If the size of the hash table is at least this value, packet division was completed at level 2. If the packet division level is 3 or more, you can complete the packet division at level 2 by specifying this value as the hash table size.[6] If level 2 packet division did not take place, this value is 0.

91. Maximum packet size at level 3 (KB)

    This is the maximum packet size after level 3 packet division during hash join or subquery hash execution.[3]

    If the size of the hash table is at least this value, data was processed in packets with a maximum level of 3. If the hash table size is not greater than this value, a packet was partially expanded in the hash table, thereby adversely affecting the processing efficiency. In this case, specify at least this value as the hash table size.[6] Alternatively, performance may improve by avoiding the hash join or subquery hash execution. If level 3 packet division did not take place, this value is 0.

92. Unsuccessful page searches count during free space reusage execution

    This is the number of times that the mode was returned to new page allocation mode because the free space reusage facility was unable to find reusable free space when the mode was switched from new page allocation mode to free page reuse mode. If this value is a value other than 0, an inefficient page search process may have occurred during an update or insertion process executed by the UAP.

For details about the free space reusage facility, see the *HiRDB Version 8 Installation and Design Guide*.

93. Mode switches count from new page allocation mode to free page reuse mode

    This is the number of times that the mode was switched from new page allocation mode to free page reuse mode when the free area reusage facility was executed. If this value is close to the number of update and insertion processes executed by the UAP, an inefficient page search process may have occurred.

94. Cache buffer shortage occurrences count

    This is internal information used by the system.

95. Cache buffer allocation flushes count

    This is internal information used by the system.

96. WRITEs count during cache buffer area allocation flushing

    This is internal information used by the system.

97. Maximum cache buffer allocation flushes count

    This is internal information used by the system.

98. Average cache buffer allocation flushes count

    This is internal information used by the system.

99. Data and index page references count when local buffer used

    This is the number of times a data or index page was referenced from this UAP.

100. Data and index page updates count when local buffer used

    This is the number of times a data or index page was updated from this UAP.

101. Data and index page buffer hits count in local buffer

    This is the buffer hits count for data pages and index pages.

    If the buffer hit rate (101 $\div$ 99 $\times$ 100) is low for a UAP that performs random access, tune the buffer.

102. Data and index page real READs count when local buffer is used

    This is the number of times a data or index page was actually read by this UAP.

    If the prefetch facility is being used, the number of look-ahead READs by the prefetch facility is also included. If the buffer hit rate is low, the READs count becomes high.

103. Data and index page real WRITEs count when local buffer is used

    This is the number of times a data or index page was actually written by this UAP.

104. Local buffer flush count

     This is the number of times the buffer was flushed to create space for a new page. This indicates the number of times a page was swept out of memory because the buffer was full.

105. Asynchronous READ request count

     This is the number of times the asynchronous READ process requested a batch look-ahead read processing when the asynchronous READ facility was used.

106. Synchronization wait count during asynchronous READ

     This is the number of times a synchronization wait occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

107. Cumulative synchronization wait time during asynchronous READ

     This is the cumulative wait time (seconds) of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

108. Cumulative synchronization wait time during asynchronous READ

     This is the cumulative wait time (microseconds) of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

109. Average synchronization wait time during asynchronous READ

     This is the average wait time (seconds) of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

110. Average synchronization wait time during asynchronous READ

     This is the average wait time (microseconds) of the synchronization waits that occurred while the asynchronous READ process performed a batch look-ahead read when the asynchronous READ facility was used.

111. Average synchronous input/output time during asynchronous READ

     This is the average synchronous READ time (seconds) for initial batch reads of the first page when the asynchronous READ facility was used.

112. Average synchronous input/output time during asynchronous READ

     This is the average synchronous READ time (microseconds) for initial batch reads of the first page when the asynchronous READ facility was used.

113. Maximum comparison count[3] during hash table search processing in hash join, subquery hash execution

This is the maximum number of comparisons for data items that have the same hash value in one hash table search.

114. Total comparison count[1] during hash table search processing in hash join, subquery hash execution

     This is the total number of comparisons for data items that have the same hash value during hash table search processing.

115. Total hash table search count[1] in hash join, subquery hash execution

     This is the number of times the hash table is searched.

[1] For HiRDB/Parallel Server, this is the total of all servers.

[2] This is the sum of all global buffers.

[3] For HiRDB/Parallel Server, this is the maximum value of each back-end server.

[4] More resources than the value obtained from the formula may be required due to temporary fragmentation. Therefore, specify a sufficient value.

[5] If the hash table size increases, the number of packet divisions may increase; therefore, a bigger hash table may be required than when the tuning information was obtained. If you have increased the hash table size on the basis of this tuning information, obtain the tuning information again. If an expected result is not obtained, you need to increase the hash table size again on the basis of the obtained tuning information.

[6] If the hash table size increases, the number of packet divisions may increase; therefore, a smaller hash table may be enough to complete packet division at an intended level than when the tuning information was obtained. On the other hand, if you reduce the hash table size, the number of packet divisions may decrease; therefore, packet division may not be completed at the same level as when the tuning information was obtained. Therefore, use the tuning information for the purpose of increasing the hash table size.

10.1.5 Command trace facility

The command trace facility outputs a client's trace information to the command trace file when a command is executed by a UAP (during the execution of the `COMMAND EXECUTE` SQL statement).

When the command trace file becomes full, the facility overwrites the oldest information.

(1) How to obtain command trace information

You can obtain command trace information by specifying appropriate values in `PDCLTPATH` and `PDCMDTRACE` in the client environment definitions. For details about each client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

Two command trace files named `pdccmd1.trc` and `pdccmd2.trc` are output to the specified directory.

(2) Interpreting command trace information

Command trace information is output when a command is executed by a UAP. The following shows sample command trace information and explains each item:

Output example

```
** COMMAND TRACE (CLT:06-00:Jan 11 2001) HP32 **  [1]

  USER APPLICATION PROGRAM FILE NAME : TESTAP  [2]
  COMMAND START TIME : 2001/01/11 10:55:27  [3]
  COMMAND EXECUTE ENVIRONMENT & STATUS :  [4]
    PDASTHOST(dcm3500)
    PDASTPORT(20266)
    PDSYSTEMID("HRD1")
    PDUSER("hirdb")
    PDASTUSER("hirdb ")
    PDCMDWAITTIME(0)
    ENVGROUP("")
    CLTPID(9155) CLTTID(0)
  [5]  [6]    [7]        [8]    [9]
  9155  0  2001/01/11 10:55:27   0  pdhold -r RDDATA01
  9155  0  2001/01/11 10:55:27   1  KFPZ02444-E Communication
error,
                                    func=connect, errno=2
```

Explanation

1. Command trace header

   The header contains the following information:

   - Version of the linked library
   - Library creation date (in the format *Mmm dd yyyy*)
   - Platform in use (For details about the character strings that are displayed for the platforms, see the *Explanation* section in *10.1.1(2) Examining SQL trace information*.)

2. UAP name

   This is the value of `PDCLTAPNAME` specified in the client environment definition.

3. Command start date and time

   This is the date and time the command execution began.

4. Command execution environment and status

   This is the value of the client environment definition and status during command execution.

5. UAP process number

   This is the UAP process number. If the correct process number cannot be obtained, an invalid value may be displayed (Windows).

6. UAP thread number

   If the UAP is running with multi-thread, this indicates the UAP thread number; otherwise, a value of `0` is displayed. Note that the facility may display an invalid numeric value if it is unable to obtain the correct thread number.

7. Command trace acquisition date and time

   This is the date and time the command trace information was acquired.

8. Command trace counter

   This is the count that was incremented each time a command trace was accepted. The value range is from `0` to `65535`.

9. Trace data

   This is the trace data.

(3) Backing up the command trace file

If the command trace file becomes full while writing command trace information, HiRDB continues output using another command trace file. In this case, existing contents of the command trace file are overwritten, beginning with the oldest information. Therefore, you should make a backup copy of a command trace file when the UAP is terminated.

To determine the command trace file that is being used currently, check the most recent update dates/times of the files. The command trace file that was updated most recently is the current file.

For a Windows version HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX version HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

10.1.6 SQL trace dynamic acquisition facility

The SQL trace dynamic acquisition facility lets you dynamically obtain SQL trace information using a command during UAP execution. Acquisition of SQL trace information begins at the next `CONNECT`.

(1) Specifying the client environment definitions

Specify the SQL trace file storage directory in `PDTRCPATH` beforehand. The facility creates two SQL trace files: `pdc`*HHMMSSmmm*_*XXX*`_1.trc` and `pdc`*HHMMSSmmm*_*XXX*`_2.trc`, where *HHMMSSmmm* indicates the `CONNECT` time (*HH*:*MM*:*SS*:*mmm*) and *XXX* indicates the connection sequence number.

(2) Trace acquisition command (pdtrcmgr)

If the directory specified with the `-d` option is the same as the directory specified in the `PDTRCPATH` client environment definition variable during UAP execution, the `pdtrcmgr` command issues the trace acquisition start and end requests.

(a) Format

```
pdtrcmgr  -d directory-name-specified-in-PDTRCPATH
           [{-b| -e}]
           [-k{[s] [u] [p] [r]| a}]
           [-n PDCLTAPNAME]
           [-s SQL-trace-file-size]
           [-o]
```

(b) Options

- **-d *directory-name-specified-in-*PDTRCPATH**

~ <path name>

Specifies the absolute path name of the value (directory name) specified in the `PDTRCPATH` client environment definition variable to start or stop the acquisition of trace information for the UAP.

The facility issues a trace acquisition start or stop request for all UAPs for which the specified directory matches the directory in `PDTRCPATH`.

- **{-b| -e}**

Specifies whether to start or stop the acquisition of the SQL trace:

`-b`: Starts the acquisition of SQL trace.

`-e`: Stops the acquisition of SQL trace.

- **-k{[s] [u] [p] [r] | a}**

Specifies the information to be output. When this option is omitted, the facility outputs only the SQL trace information.

`s`: Outputs information by the SQL.

`u`: Outputs information by the UAP.

`p`: Outputs access path information.

r: Outputs SQL runtime interim results.

a: Outputs all information.

s, u, p, and r can be specified in different combinations (such as su, spr, or spr). Specifying sup is the same as specifying a. If u, p, r, ur, pr, or upr is specified, SQL trace information is not output.

When the -e option is specified, the specification of the -k option becomes invalid.

For details about the UAP statistical report, see *10.1.4 UAP statistical report facility*.

- **-n PDCLTAPNAME**

Specifies that only the UAP specified in the PDCLTAPNAME client environment definition variable is to be subject to acquisition of an SQL trace. The facility ignores this option if the -e option is specified.

- **-s *SQL-trace-file-size***

~ <unsigned integer> ((0 or 32,768 to 2,000,000,000)) <<32,768>>

Specifies the size of the SQL trace file in bytes.

If 0 is specified, the maximum file size is assumed. If a value in the range from 32,768 to 2,000,000,000 is specified, the specified size of file is used.

The facility ignores this option if the -e option is specified.

- **-o**

Specifies that SQL trace files are to opened and closed in CONNECT and DISCONNECT units. The facility ignores this option when the -e option is specified.

When SQL trace files are opened and closed in CONNECT and DISCONNECT units instead of operation units (SQL units), the SQL trace output time can be shortened because the overhead is reduced.

If you omit this option, the SQL trace dynamic acquisition facility opens and closes SQL trace files in operation units.

This facility continues to write information as long as the SQL trace file is open. Therefore, if you specify this option, some SQL trace information may be discarded if DISCONNECT cannot be executed properly.

10.1.7 Reconnect trace facility

When the automatic reconnect facility executes reconnection, reconnect trace information, which consists of the connection handle value managed internally by HiRDB, the connection information before and after reconnect, and the reconnect time, is output to the reconnect trace file. This information is used for tracking connection information in the trace output by the PRF trace facility of Cosminexus.

(1) How to obtain the reconnect trace information

Reconnect trace information can be obtained by setting a value in the `PDRCTRACE` client environment definition.

HiRDB creates two reconnect trace files in the directory specified in the `PDCLTPATH` client environment definition. The names of the created files are `pdrcnct1.trc` and `pdrcnct2.trc`.

(2) Interpreting reconnect trace information

Reconnect trace information is output when the automatic reconnect facility establishes a connection automatically.

An output example of a reconnect trace is shown below.

```
   [1]  [2]                          [3]                                [4]
40004250 S 2004/04/12 11:10:36.766 - 2004/04/12 11:10:41.846 sds:9:23763 =>
sds:10:23750
40004250 S 2004/04/12 11:11:07.491 - 2004/04/12 11:11:12.547 sds:10:23750 =>
sds:11:23765
40004850 F 2004/04/12 11:17:58.285 - 2004/04/12 11:18:23.395 sds:14:23751 =>
40005050 S 2004/04/12 11:27:35.098 - 2004/04/12 11:27:40.152 sds:1:24414 =>
sds:2:24418
```

Explanation

1. Connection handle value

   The connection handle value that HiRDB manages internally is output in hexadecimal format.

   The value is 8 digits if the client is operating in 32-bit mode or 16 digits if the client is operating in 64-bit mode. The UAP views traces that have the same connection handle value as the same connection.

   In the output example above, `40004250` is output twice as the connection handle value. When viewed from the UAP that uses this connection handle, this information indicates that reconnect processing was executed twice.

2. Reconnect result

   The reconnection result is displayed.

   `S`: Success

   `F`: Failure

3. Reconnect start and end dates and times

   After a disconnection is detected, the dates and times when the reconnection was started and when it ended normally are displayed in milliseconds. If reconnect processing fails, the date and time immediately before control is returned to the UAP is displayed.

4. Connection information before and after reconnect

   Connection information for both before reconnect and after reconnect is displayed. The connection information displays the connection server name, the connection sequence number, and the process ID of the connection server, with the items separated by colons.

   If reconnect processing fails, the connection information for after the reconnect is not displayed (becomes blank).

(3) Matching trace information with PRF trace information of Cosminexus

The connection information shown under 4 of the output example is output to the PRF trace information of Cosminexus. If the automatic reconnect facility subsequently executes reconnect processing, match the trace information with PRF trace information as follows.

To match trace information with the PRF trace information of Cosminexus:

1. Get the HiRDB connection information in the PRF trace information.
2. In 4 of the reconnect trace file, search for the connection information obtained in Step 1, and get the corresponding connection handle value.
3. From 1 of the reconnect trace file, track the trace information that has the same connection handle value obtained in Step 2. If the same connection handle value is found, and the connection information before reconnect is the same as the connection information after reconnect for the previous instance of the same connection handle, the connection handle can be used for tracking. If the connection information is different, the connection handle cannot be used for tracking because a new connection (`DISCONNECT`-`CONNECT`) was established with the connection handle.

(4) Backing up reconnect trace information

If the reconnect log file becomes full while reconnect trace information is being output, the reconnect log is output to the other reconnect trace file. In this case, the old reconnect trace information stored in the takeover reconnect trace file is erased and overwritten by new reconnect trace information. Therefore, if the system is being operated for a long period of time, copy the contents of the reconnect trace file and back up the information, as necessary.

To determine the reconnect trace file that is being used currently, check the most recent update dates/times of the files. The reconnect trace file that was updated most recently is the current file.

For a Windows version HiRDB client, you use the `dir` command or the Explorer to check the file update dates/times.

For a UNIX version HiRDB client, you use the OS's `ls -l` command to check the file update dates/times.

10.1.8 HiRDB SQL Tuning Advisor access path information file

An access path information file used by the HiRDB SQL Tuning Advisor is output to the HiRDB client side. You can use HiRDB SQL Tuning Advisor to perform analyses by referencing this access path information file and SQL trace information. This makes it easy to identify how the SQLs are affecting performance. For details about the HiRDB SQL Tuning Advisor, see the Help for *HiRDB SQL Tuning Advisor*.

(1) Setting method

To output the HiRDB SQL Tuning Advisor access path information file, you must set the client environment definitions explained below.

- `PDTAAPINFPATH`

  Specifies the access path information file output directory. If an error occurs during the output processing (for example, because the output directory does not exist, or because the user does not have write privilege), the access path information is not output. Even when an error occurs in the output processing, no error occurs in the executing SQL.

- `PDTAAPINFMODE`

  Specifies the file name format for the access path information files.

- `PDTAAPINFSIZE`

  Specifies the file size of the access path information files. Two access path information files are created. When the specified file size is reached in the current file, the output destination is switched to the other file.

(2) Notes

- This function can be used only when the HiRDB server is version 06-00 or later.
- SQL objects with SQL code in a buffer may increase the workload of the HiRDB server due to the need to re-create the SQL objects.
- When this function is used, the inter-process memory communication facility cannot be used. Even if you specify `MEMORY` in the `PDIPC` operand of the client environment definition, the operation when `DEFAULT` is specified takes precedence.
- When you use the dynamic browsing function of the HiRDB SQL Tuning Advisor, the specification of `PDTAAPINFPATH` in the client environment definition is ignored.

- If you use this function with an HiRDB server earlier than version 07-03, the output path information for the UAP statistics report is not output, even if you have specified that access path information is to be obtained by the UAP statistics report facility (by specifying `p` or `a` in the `PDUAPREPLVL` client environment definition).

10.2 UAP error recovery

When an error occurs in a UAP, measures must be taken to prevent the entire HiRDB system from halting. This section explains the following three methods of recovering from UAP errors:

- UAP transaction rollback by HiRDB
- Transaction rollback by UAP instruction
- Memory capacity re-evaluation

Table 10-4 shows the UAP error types and the recovery methods.

Table 10-4: UAP error types and recovery methods

Error type	Detection method	System action	Recovery method
UAP abnormal termination	UAP processing time Monitoring	Disconnects the UAP process	UAP transaction rollback
UAP endless loop			
Transaction incomplete			
UAP processing error	Various error detection at the servers[1]	Sends error response to UAP	Transaction rollback by UAP instruction
Error detection and rollback request by UAP	Error detection by UAP	Follows an instruction from UAP	
Deadlock	HiRDB deadlock detection	Sends error response to UAP (implicit rollback)	Termination of UAP transaction
Memory shortage	Error during memory allocation	Disables UAP activation	Reevaluate shared memory and process-specific memory[2]

[1] Front-end or back-end server.

[2] Request that the HiRDB system administrator re-evaluate shared memory and process-specific memory.

(1) Monitoring UAP processing time

When a UAP is executed, HiRDB's UAP monitors the processing time to prevent a UAP error from halting the HiRDB processing for an extended period of time.

For time monitoring, a monitoring time must be specified in the `PDSWAITTIME` environment variable during client environment definition; if omitted, the UAP monitors by HiRDB's default monitoring time.

For details about client environment definition, see *6.6 Client environment definitions (setting environment variables)*.

(2) Detecting errors at servers

A HiRDB/Parallel Server returns an error status to the UAP when an error, such as a database processing error, is detected at the front-end server or back-end server while executing SQL statements; steps such as process disconnection must be taken. When a UAP issues a rollback request in response to an error status, HiRDB performs a recovery process.

(3) Detecting UAP errors

When an error is detected in a UAP, a recovery process is started when a `rollback` request is issued. If the UAP was processed normally, the process is disconnected based on a disconnection instruction from the UAP.

(4) Re-evaluating memory capacity

When a shortage occurs in the shared memory or process-specific memory, a message is output indicating the memory or disk space shortage. When such a message is output, enough memory to activate the UAP must be allocated and the UAP must be re-executed.

For details about how to check and, if necessary, revise shared memory and process-specific memory, see the *HiRDB Version 8 Installation and Design Guide* or contact the HiRDB system administrator.

Chapter

11. Using a Distributed Database (Limited to HP-UX and AIX 5L)

This chapter explains how to create a UAP that accesses a distributed database.

A distributed database can be used with the HP-UX and AIX 5L versions of HiRDB.

11.1 Format of a distributed database

This section explains the format of a distributed database.

11.1.1 Accessing a distributed database and its relationship to RD-nodes

HiRDB can use the following DBMSs (Database Management Systems) to implement a distributed database:

- HiRDB at other nodes
- XDM/RD (including XDM/RD E2)
- ORACLE
- RDB1 E2
- SQL/K

A distributed database between HiRDB and a DBMS of another node is implemented using the remote database access facility of DF/UX (Distributing Facility/for UNIX).

Accessing a distributed database involves multiple DBMSs that run on multiple nodes in a network (e.g., server machines, host computers, and PC servers). Each DBMS identified on the basis of its location in the network is called an *RD-node*. However, because multiple DBMSs may be active at the same time at a single node or a single DBMS may be active by linking multiple nodes, there is not necessarily a one-to-one relationship between RD-nodes and nodes in the network. Therefore, a DBMS and the databases managed by that DBMS may sometimes be referred to collectively as an RD-node.

An RD-node is identified by an RD-node name, and the HiRDB that is connected using a CONNECT statement without RD-node name specification (HiRDB specified at the connection-destination by the client environment definition) is called *the default RD-node*. RD-nodes other than the default RD-node are called *distributed RD-nodes*, which also include DBMSs other than the HiRDB.

11.1.2 Relationship between a connection between RD-nodes and an SQL connection

To access a remote database, the HiRDB of the distributed client must connect to the DBMS that acts as the distributed server. The facility for accessing the distributed server (distributed RD-node) of another node using the HiRDB of the local node as the distributed client (default RD-node) is called the *distributed client facility*.

The distributed client facility is characterized by the following two features:

- The distributed client facility enables the HiRDB to connect automatically to a

DBMS at another node, so a UAP for connecting to the DBMS at the other node is not required.

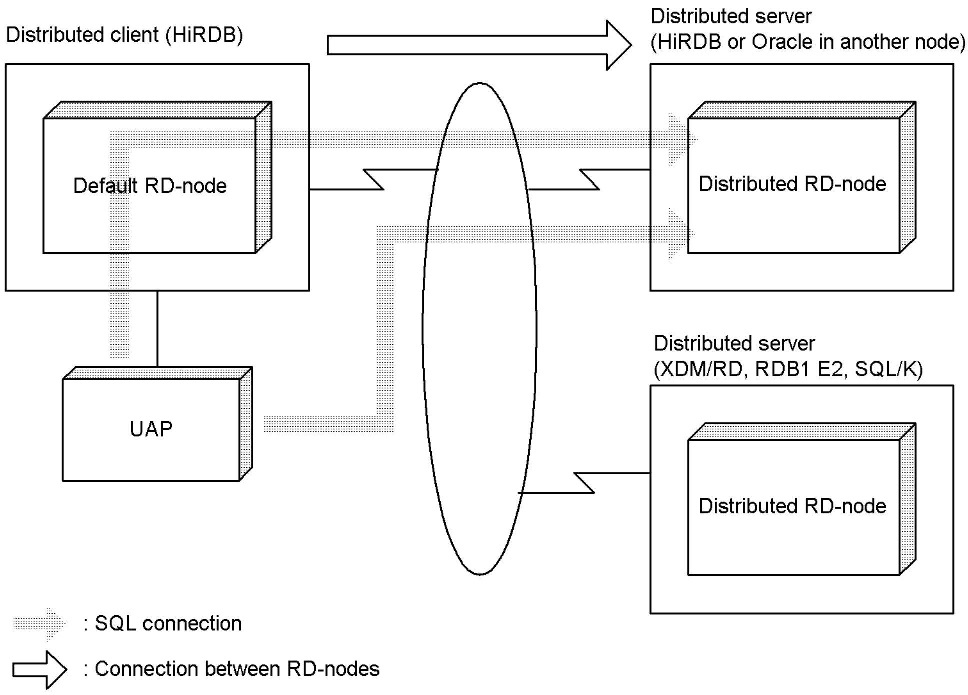
- It is possible to simultaneously access a remote database and the local database from a single UAP (note, however, that if the remote database and the local database are updated in a single transaction, transaction compatibility is not guaranteed).

The logical connection from a UAP to an RD-node for executing SQL statements is called an *SQL connection*. The SQL connection to the default RD-node is called the *default SQL connection*.

An SQL connection includes the authorization identifier for privilege checking at the connection-destination RD-node. This authorization identifier can be specified individually for each SQL connection.

Figure 11-1 shows the distributed database connection format.

Figure 11-1: Distributed database connection format



11.1.3 Generating and terminating an SQL connection

(1) Generating an SQL connection

Table 11-1 shows the times at which an SQL connection is generated, the connection-destination RD-node for the generated SQL connection, and the authorization identifier contained in the generated SQL connection.

Table 11-1: Generating an SQL connection

Generation time	Connection-destination RD-node	Authorization identifier
Execution of `CONNECT` statement without RD-node specification	Default RD-node	Authorization identifier specified in the `CONNECT` statement; if omitted, the authorization identifier specified in the `PDUSER` client environment variable.
Execution of `CONNECT` statement with RD-node specification after the default SQL connection has been generated	Specified distributed RDnode	Authorization identifier specified in the `CONNECT` statement with RD-node specification; if omitted, the authorization identifier contained in the default SQL connection.
Execution of `CONNECT` statement with RD-node specification before the default SQL connection has been generated*	Default RD-node	Authorization identifier specified in the `PDUSER` client environment variable.
	Specified distributed RDnode	Authorization identifier specified in the `CONNECT` statement with RD-node specification; if omitted, the authorization identifier specified in the `PDUSER` client environment variable.
Default SQL connection is used to execute SQL statements for accessing the database of a distributed RD-node before an SQL connection to the distributed RD-node has been generated*	Distributed RD-node where the database to be accessed is located	Authorization identifier contained in the default SQL connection.

* To access the database of a distributed RD-node, both an SQL connection to that distributed RD-node and the default SQL connection are required. Therefore, if a UAP issues a CONNECT statement with RD-node specification before the default SQL connection is generated, the HiRDB client generates the default SQL connection automatically. Also, if the UAP issues an `SQL` statement that uses the default SQL connection for accessing the database of a distributed RD-node before an SQL connection to the distributed RD-node is generated, the HiRDB of the default RD-node generates automatically an SQL connection to that distributed RD-node. For details about how to use the default SQL connection for accessing the database of a distributed RD-node, see *11.2.2 Using the default SQL connection*.

(2) Terminating an SQL connection

An SQL connection is terminated in the following cases:

- All SQL connections are terminated when a `DISCONNECT` statement without RD-node specification is executed
- When a `DISCONNECT` statement with RD-node specification is executed, only the SQL connection to the specified RD-node is terminated

There can be only one SQL connection to a distributed RD-node at a time. After a `DISCONNECT` statement with RD-node specification is used to delete the existing SQL connection to a distributed RD-node, an SQL connection to another distributed RDnode can be created.

11.1.4 Current SQL connection and database access

Each SQL statement for accessing a database is executed using an SQL connection called the *current SQL connection*. Even if multiple SQL connections have been generated, there is only one current SQL connection at a time. The connection-destination RD-node for the current SQL connection is called the *current RD-node*.

At any time, any of the existing SQL connections (other than the current SQL connection) can be made into the current SQL connection by issuing the `SET CONNECTION` statement.

Table 11-2 shows the times at which the current SQL connection is set and the current RD-node after the setting.

Table 11-2: Current SQL connection setting

Setting time	Current RD-node
`CONNECT` statement without RD-node specification is executed	Default RD-node
`CONNECT` statement with RD-node specification is executed	Specified distributed RD-node
`SET CONNECTION` statement with distributed RD-node specification is executed	Specified distributed RD-node
`SET CONNECTION` statement with DEFAULT specified is executed	Default RD-node
`DISCONNECT` statement with current RD-node and RD-node specifications is executed	Default RD-node

Table 11-3 shows the relationship between the current SQL connection and the range of databases that can be accessed.

Table 11-3: Current SQL connection and range of databases that can be accessed

Current SQL connection	Range of databases that can be accessed
Default SQL connection	• Databases at the current RD-node (= default RD-node) • Databases at all distributed RD-nodes that can be connected to the default RD-node*
SQL connection to distributed RDnode	• Databases at the current RD-node (= single distributed RD-node)

* For details about how to use the default SQL connection for accessing the database of a distributed RD-node, see *11.2.2 Using the default SQL connection*.

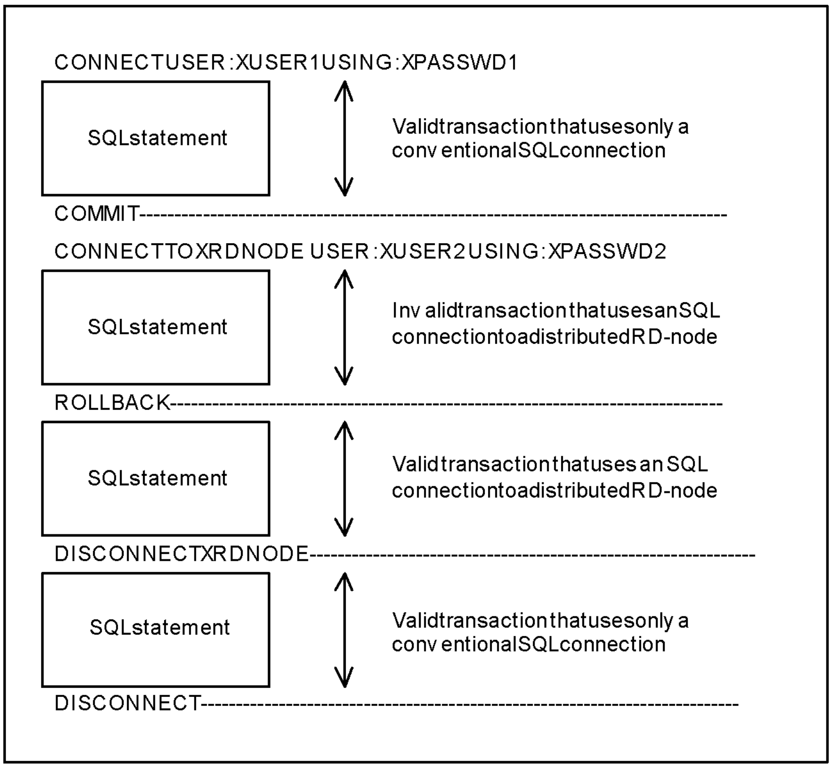
11.1.5 SQL connection and transaction control

Unlike transaction control that does not use a distributed database, when a distributed database is used in transaction control, execution of a `DISCONNECT` statement with RD-node specification must be added when the transaction is being terminated. When the `DISCONNECT` statement with RD-node specification is executed, the current transaction is terminated normally and a synchronization point is set.

By omitting the `COMMIT` statement and using the `DISCONNECT` statement with RDnode specification to terminate a transaction normally, the number of communication contacts with the default RD-node can be reduced from the number required in the case in which the `COMMIT` statement is used for transaction termination.

Figure 11-2 shows examples of transaction startup and termination using an SQL connection to a distributed RD-node.

Figure 11-2: Examples of transaction startup and termination using an SQL connection to a distributed RD-node



If the UAP is terminated without issuing a `DISCONNECT` statement without RD-node specification, the transaction that was being executed is rolled back. In this case, all changes made by the transaction are cancelled, even if the database of the distributed RD-node was changed by the transaction that was executing.

Because there can be only one SQL connection to a distributed RD-node at a time, only one distributed RD-node can be accessed within a single transaction. However, two RD-nodes can be accessed within a single transaction if the default RD-node is included.

By issuing the `SET CONNECTION` statement to switch the current SQL connection, the default SQL connection and the SQL connection to a distributed RD-node can be used alternately within a single transaction.

A `DISCONNECT` statement with RD-node specification cannot be used in X/Open application programs that execute in the OLTP environment.

11.2 Creating a UAP that accesses a remote database

This section explains how to create a UAP that accesses a remote database with the distributed client facility.

11.2.1 Rules governing distributed clients and servers

UAP creation is subject to rules governing distributed clients and servers; these rules are explained below.

(1) Distributed client rules

(a) Syntax rules

The syntax of the SQL coded in the UAP is checked according to the grammar of HiRDB. If the distributed server is not an HiRDB, SQL statements that comply with the grammar of the distributed server cannot be used because they will not comply with the HiRDB grammar.

(b) Search data length restrictions

The data length that can be searched by an SQL statement is restricted depending on the size of the DF/UX reception buffer. You must be especially careful when you use the batch search function and when you search BLOB type data.

For details about the restrictions on the search data length depending on the DF/UX reception buffer size, see the manual *Distributed Database System DF/UX*.

(2) Distributed server rules

(a) Syntax rules

The distributed server checks syntax according to the SQL grammar of the distributed server. If the SQL of HiRDB does not conform to the grammar of the distributed server, the nonconforming SQL statements cannot be used. The following SQL statements are transferred from the client to the server in their original syntax; even though an SQL statement is coded in the same format for HiRDB and for the distributed server, functional differences between HiRDB and the distributed server may produce different execution results:

- `DECLARE CURSOR` statement (only for direct cursor specification)
- `DELETE` statement
- `INSERT` statement
- `SELECT` statement
- `UPDATE` statement

(b) Character code conversion rules

If the distributed server is XDM/RD, RDB1D2, or SQL/K, different character code sets are used in the distributed client and in the distributed server. This means that when character data is sent or received, the other side must convert the data. Also, when character string data is used for sorting, the data is sorted according to the character code order used by the server. For this reason, you must use care and note that the sort results may not match those produced when data is sorted according to the character code order at the client.

11.2.2 Using the default SQL connection

This section explains use of the default SQL connection as the current SQL connection for accessing a remote database.

When the default SQL connection is used as the current SQL connection, it is possible to access the databases of all distributed RD-nodes that can be connected to the default RD-node. To do this, it is necessary to use one of the following methods to notify the default RD-node's HiRDB of the distributed RD-node where the table and procedure to be accessed are located:

1. By qualifying the table name with the RD-node name
2. By using a table alias
3. By qualifying the procedure name with the RD-node name

Methods (a) and (c) can also be used when a table or procedure located at the default RD-node is to be accessed. In such a case, the name is qualified with the RD-node name of the default RD-node. Tables or procedures whose names are not modified by an RD-node name are processed under the assumption that they are located at the current RD-node.

Use of the default SQL connection also enables access to the database of a distributed RD-node. However, the following restrictions apply:

- The tables of only one RD-node can be accessed from a single `SQL` statement
- Only one distributed RD-node can be accessed within a single transaction (two RD-nodes can be accessed if the default RD-node is included)

(1) Qualifying the table name with the RD-node name

The following format is used to qualify the table name with the RD-node name:
`RD-node-name.authorization-identifier.table-identifier`

RD-node-name

Specifies the name of the RD-node where the table is located.

authorization-identifier.table-identifier

Specifies the authorization identifier and table identifier defined at the RD-node where the table is located.

Specification example

- Retrieve the table named `MANAGER.ORDERS` at the RD-node named RDNODE10:

```
SELECT * FROM RDNODE10.MANAGER.ORDERS
```

(2) Using a table alias

The following format is used to specify the table at a distributed RD-node using a table alias. Note that a table alias is not supported if either the distributed server or the distributed client is Solaris.
[*authorization-identifier*.]*table-alias*

[*authorization-identifier.*]*table-alias*

Specifies the name defined in `CREATE ALIAS`. Note that the name specified here may not match the authorization identifier or the table identifier defined at the RD-node where the table is located.

Specification example

- Assign the alias `MANAGER.ORDERS`, which is the same as the name defined at the RD-node where the table is located, to the table named `MANAGER.ORDERS` at the distributed RD-node named `RDNODE10`, and retrieve the table using this defined alias:

1. Using `CREATE ALIAS` to define an alias

```
CREATE ALIAS MANAGER.ORDERS FOR RDNODE10.MANAGER.ORDERS
```

2. Retrieval using the alias defined above

```
SELECT * FROM MANAGER.ORDERS
```

(3) Qualifying the procedure name with the RD-node name

The following format is used to qualify the procedure name with the RD-node name:
RD-node-name.authorization-identifier.*routine-identifier*

RD-node-name

Specifies the name of the RD-node where the procedure is located.

authorization-identifier.routine-identifier

Specifies the authorization identifier and routine identifier defined at the RD-node where the procedure is located.

Specification example

- Retrieve the procedure named `MANAGER.PROC10` at the RD-node named

```
  RDNODE10:
CALL RDNODE10.MANAGER.PROC10 (arguments)
```

(4) Access using an authorization identifier that is different from the default SQL connection

Even when the default SQL connection is used, an SQL connection to a distributed RD-node is required for accessing a remote database. If the UAP issues an SQL for accessing a remote database without creating an SQL connection, the HiRDB at the default RD-node generates automatically an SQL connection to the distributed RD-node.

However, because the SQL connection generated automatically by the HiRDB contains the same authorization identifier as the default SQL connection, the remote database cannot be accessed if that authorization identifier does not have the required privilege at the distributed RD-node. In such a case, it is possible to create an SQL connection that contains an authorization identifier that has the required privilege at the distributed RD-node in advance and then to use that SQL connection.

Usage example

In the example shown below, a `CONNECT` statement with RD-node specification is used to create an SQL connection that contains an authorization identifier that has the required privilege at the distributed RD-node. Because the SQL connection to the distributed RD-node becomes the current SQL connection when the `CONNECT` statement with RD-node specification is executed, the `SET CONNECTION` statement is used first to revert the current SQL connection to the default SQL connection, and then an SQL for accessing the remote database is issued.

```
CONNECT TO RDNODE10 USER:USER2 USING :PSWD2SET CONNECTION
DEFAULTSELECT SQUANTITY INTO :QUANTITY FROM
RDNODE10.MANAGER.STOCK WHERE PCODE='302S'
```

11.2.3 Using an SQL connection to a distributed RD-node

This section explains use of an SQL connection to a distributed RD-node as the current SQL connection for accessing a remote database.

When an SQL connection to a distributed RD-node is used as the current SQL connection, only the databases at the current RD-node can be accessed.

Using an SQL-connection to a distributed RD-node offers the following advantage: It is not necessary to use an RD-node name as a modifier or a table alias, as is required when the default SQL connection is used for accessing a remote database. The table name and procedure name defined in the current RD-node can be specified directly in the `SQL` statement.

(1) Setting the current SQL connection using a CONNECT statement with RD-node specification

Because the current SQL connection becomes the SQL connection to the distributed RD-node when the `CONNECT` statement with RD-node specification is executed, the SQL connection to the distributed RD-node can be used immediately.

Usage example

In the example shown below, the current SQL connection specified by the `CONNECT` statement with RD-node specification is used to retrieve data from the inventory table titled `MANAGER.STOCK` at the distributed RD-node named `RDNODE10`.

```
CONNECT TO RDNODE10 USER:USER2 USING :PSWD2SELECT SQUANTITY
INTO :QUANTITY FROM MANAGER.STOCK WHERE PCODE='302S'
```

(2) Setting the current SQL connection using a SET CONNECTION statement

If the current SQL connection is not the SQL connection to the distributed RD-node that is to be accessed, the current SQL connection can be changed with the `SET CONNECTION` statement. The SQL connection to the distributed RD-node must be created before the `SET CONNECTION` statement is issued.

Usage example

In the example shown below, the `SET CONNECTION` statement is used to set the distributed RD-node named `RDNODE10` as the current RD-node, and then the procedure named `MANAGER.PROC10` at that RD-node is called.

```
SET CONNECTION RDNODE10CALL MANAGER.PROC10 (arguments)
```

11.3 Available SQL statements

This section explains the SQL statements that can be used for remote database access by means of the distributed client facility.

11.3.1 SQL statements usable for remote database access

Any SQL statements that are supported by the distributed client facility can be used in accessing a remote database. Table 11-4 lists the SQL statements that are supported by the distributed client facility.

Table 11-4: SQL statements supported by distributed client facility

Type	SQL statement
Data manipulation SQL	`CALL` statement (call procedure)
	`CLOSE` statement (close cursor)
	`DECLARE CURSOR` statement (declare cursor)
	`DELETE` statement (delete rows)
	`DESCRIBE` statement (receive retrieval information)
	`EXECUTE` statement (execute SQL statement)
	`EXECUTE IMMEDIATE` statement (preprocess and execute SQL statement)
	`FETCH` statement (fetch data)
	`INSERT` statement (insert rows)
	`OPEN` statement (open cursor)
	`PREPARE` statement (preprocess SQL statement)
	`PURGE TABLE` statement (delete all rows)*
	`SELECT` statement (retrieve data)
	`UPDATE` statement (update data)
Control SQL	`COMMIT` statement (terminate transaction normally)
	`LOCK TABLE` statement (lock table)*
	`ROLLBACK` statement (cancel transaction)
Embedded language	`GET DIAGNOSTICS` statement (get diagnostic information)

* The SQL statement is not supported for Solaris.

11.3.2 Details about available SQL statements

Table 11-5 shows details about the SQL statements that can be used for remote database access (for details about these SQL statements, see the *HiRDB Version 8 SQL Reference* manual).

Table 11-5: Details about SQL statements usable for remote database access

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
Variable	{:*embeddedvariable* [:*indicatorvariable*] | ? *parameter*}	Y[9]	Y[9]	Y[9]	Y[9]	Y[9]
Table name	*RD-node-name*.*authorization-identifier*.*tableidentifier*	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]
	[*authorization-identifier.] table-alias*	Y[1]	Y[1]	Y[1]	Y[1]	Y[1]
	[*authorization-identifier*.] *table-identifier*	Y[2]	Y[2]	Y[2]	Y[2]	Y[2]
Table specification	*{[authorization-identifier.] table-identifier |correlation-name}*	Y[3]	Y[3]	Y	Y[3]	Y[3]
Column specification	*[table-specification.] column-name*	Y	Y	Y	Y	Y
	[table-specification.] repetition-column-name[subscript]	Y	—	—	—	—
Value specification	*{literal | variable | USER*	Y	Y	Y	Y	Y[4]
	| CURRENT DATE | CURRENT TIME	Y	Y	—	—	—
	| [statement-label.] SQL-variable-name |[[authorization-identifier.] routine-identifier.] SQL-parameter-name}	—	—	—	—	—
Term specification	*{column-specification*	Y	Y	Y	Y	Y
	| [statement-label.] SQL-variable-name | [[authorization-identifier.] routine-identifier.] SQL-parameter-name}	—	—	—	—	—
Set function	AVG, SUM, MAX, MIN, COUNT	Y	Y	Y	Y	Y

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
Scalar function	VALUE	Y	Y	—	—	—
	DATE, TIME, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, DAYS,	Y	Y	—	—	—
	DECIMAL, DIGITS, FLOAT, INTEGER, CHARACTER, HEX,	Y	Y	—	—	—
	LENGTH, SUBSTR	Y	Y[5]	Y[5]	—	—
Labeled duration	*(value-expression)* {YEAR [S] | MONTH [S] | DAY [S] }	Y	Y	—	—	—
Primary	*{(value-expression) | column-specification | value-specification | set-function | scalar-function*	Y	Y	Y	Y	Y[4]
	| *labeled-duration}*	Y	Y	—	—	—
Value expression	{[{+ | -}] *primary* | *value-expression* {+ | - | * | / | ||} *primary*}	Y	Y	Y	Y[4]	Y[4]
Comparison operators	{= | < > | < | < = | > | > =}	Y	Y	Y	Y	Y[4]
Predicate	{*value- expression* IS [NOT] NULL	Y	Y	Y	Y	Y[4]
	| *value- expression* [NOT] LIKE *value-specification*	Y	Y	Y	Y	Y[4]
	| *value-expression* [NOT] BETWEEN *value-expression* AND *value-expression*	Y	Y	Y	Y	Y[4]
	| *value-expression* [NOT] IN {(*value-specification* [, *value-specification*] ...) | *subquery*}	Y	Y	Y	Y	Y[4]
	| *value-expression comparison-operator value-expression*	Y	Y	Y	Y	Y
	| *value-expression comparison-operator subquery*	Y	Y	Y	—	—
	| *value-expression comparison-operator*{ ANY | ALL | SOME] *subquery*	Y	Y	Y	—	—
	| EXISTS *subquery*	Y	Y	Y	—	—

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
	| *term-specification* [NOT] XLIKE-*pattern-character*	Y	—	—	—	—
	| ARRAY (*repetition-column-name* [,*repetition-column-name*] ...) [ANY] (*retrieval-condition*)	Y	—	—	—	—
Search condition	{ [NOT] {(*search-condition*) | *predicate*}	Y	Y	Y	Y	Y[4]
	| *search-condition* OR {(*search-condition*) | *predicate*}	Y	Y	Y	Y	Y
	| *search-condition* AND {(*search-condition*) | *predicate*}}	Y	Y	Y	Y	Y
Selection expression	*value-expression* (other than a variable) | *table-specification*.*	Y	Y	Y	Y	Y
	| [*table-specification*.] ROW}	—	—	—	—	—
Query specification	{SELECT [{ ALL | DISTINCT}]	Y	Y	Y	Y	—
	{* | *selection-expression* [. *selection-expression*] }	Y	Y	Y	Y	Y
	FROM *table-name* [. *table-name*] ...	Y	Y	Y	Y	Y
	[WHERE *search-condition*]	Y	Y	Y	Y	Y
	[GROUP BY *column -specification*]	Y	Y	Y	Y	Y
	[HAVING *search-condition*]	Y	Y	Y	Y	Y
Query expression	{*query-specification* | (*query-expression*) | *query-expression* UNION [ALL] {*query-specification* | (*query-expression*)}}	Y	Y	Y	—	—

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
Static cursor declaration	DECLARE *cursor-name* CURSOR FOR *query-expression*	Y	Y	Y	Y6	Y6
	[ORDER BY {*columnspecification* | *sort-item-specification-number*} [{ASC | DESC}] [, {*column-specification* | *sort-item-specification-number*} [{ASC | DESC}]] ...]	Y	Y9	Y	Y7	Y7
	[{WITH {SHARE | EXCLUSIVE} LOCK	Y	Y	—	—	—
	| WITHOUT LOCK [{WAIT | NOWAIT}]}]	Y	Y	—	Y	—
	[WITH ROLLBACK]	Y	Y	—	Y	—
	| NO WAIT}]	Y	—	—	—	—
	[FOR {UPDATE [OF *column-name* [, *column-name*] ...]	Y	Y	Y	Y	Y
	| READ ONLY}]	Y	Y	—	—	—
Dynamic cursor declaration	DECLARE *cursor-name* CURSOR FOR *SQL-statement-identifier-indicating-dynamic*-SELECT-*statement*	Y	Y	Y	Y	Y
Dynamic SELECT statement Format 1	*query-expression*	Y	Y	Y	Y6	Y6
	[ORDER BY {*column-specification* | *sort-item-specification-number*} [{ASC | DESC}] [, {*column-specification* | *sort-item-specification-number*} [{ASC | DESC}]] ...]	Y	Y7	Y	Y7	Y7
	[{WITH {SHARE | EXCLUSIVE} LOCK	Y	Y	—	—	—
	| WITHOUT LOCK [{WAIT | NOWAIT}]}]	Y	Y	—	Y	—
	[WITH ROLLBACK]	Y	Y	—	Y	—
	| NO WAIT}]	Y	—	—	—	—
	[FOR UPDATE]	Y	Y	Y	Y	Y

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
Dynamic SELECT statement Format 2	SELECT {{*column-name* |*repetition-column*[[*subscript*]] |*column-name*..*attribute-name*[..*attribute-name*]..} [,{{*column-name* |*repetition-column*[[*subscript*]] |*column-name*..*attribute-name*[..*attribute-name*]..}].. |*} FROM LIST *list-name* [*lock-option*]	—	—	—	—	—
ASSIGN LIST statement Format 1	ASSIGN LIST *list-name* FROM ([*authorization-identifier*.]*table-identifier*) [WHERE *search-condition*] [WITHOUT LOCK [{WAIT|NOWAIT}]] [WITH ROLLBACK|NO WAIT]	—	—	—	—	—
ASSIGN LIST statement Format 2	ASSIGN LIST *list-name* FROM *list-name-1* [{{AND|OR|AND NOT|ANDNOT}*list-name-2* |FOR ALTERLIST]	—	—	—	—	—
DROP LIST statement	DROP {LIST *list-name*|ALL LIST}	—	—	—	—	—
OPEN statement Format 1	OPEN *cursor-name* [USING:*embeddedvariable* [, *embeddedvariable*] ...]	Y	Y	Y	Y	Y
OPEN statement Format 2	OPEN *cursor-name* USING DESCRIPTOR [:] *SQL-data-area-name*	Y	Y	Y	Y	Y
CLOSE statement	CLOSE *cursor-name*	Y	Y	Y	Y	Y
FETCH statement Format 1	FETCH *cursor-name* INTO *variable* [, *variable*] ...	Y	Y	Y	Y	Y

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
FETCH statement Format 2	FETCH *cursor-name* USING DESCRIPTOR [:]*SQL-data-area-name*	Y	Y	Y	Y	Y
	BY *variable* [ROWS]	—	—	—	—	—
FETCH statement Format 3	FETCH *cursor-name* INTO *array-variable* [, *array-variable*] ...	—	—	—	—	—
Single-row SELECT statement	SELECT [{ALL | DISTINCT}] (* | *selectionexpression* [, *selection-expression*] ...} INTO *variable* [, *variable*] ... FROM *table-name* [,*table-name*] ... [WHERE *search-condition*] [GROUP BY *column-specification*] [HAVING *search-condition*]	Y	Y	Y	Y	—
	[{WITH {SHARE | EXCLUSIVE} LOCK	Y	Y	—	—	—
	| WITHOUT LOCK [{WAIT | NOWAIT}]}]	Y	Y	—	Y	—
	[WITH ROLLBACK]	Y	Y	—	Y	—
	| NO WAIT}]	Y	—	—	—	—
Insertion value	{*value-specification* | NULL}	Y	Y	Y	Y	Y[4]
INSERT statement Format 1	INSERT INTO *table-name* [(*columnname* [, *columnname*] ...)] {VALUES (*insertion-value* [, *insertionvalue*] ...) | *query-specification*}	Y	Y	Y	Y	Y
	[WITH ROLLBACK]	Y	Y	—	Y	—
INSERT statement Format 2	INSERT INTO *tablename* (ROW) {VALUES (*row-insertion-value*) | *query-specification*} [WITH ROLLBACK]	—	—	—	—	—

<table>
<tr><th rowspan="2">Category</th><th rowspan="2">SQL statement format</th><th colspan="5">Usable at distributed server</th></tr>
<tr><th>H</th><th>X</th><th>O</th><th>R</th><th>S</th></tr>
<tr><td>INSERT statement Format 3</td><td>FOR variable
INSERT INTO
table-name[(column-name[,column-name]...)]
{VALUES
(insertion-value[,insertion-value]...)
|query-specification}

[WITH ROLLBACK]</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>INSERT statement Format 4</td><td>FOR variable
INSERT INTO table-name(ROW)
{VALUES
(row-insertion-value)|query-specification}

[WITH ROLLBACK]</td><td>—</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
<tr><td>Update value</td><td>{value-expression | NULL}</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y[4]</td></tr>
<tr><td rowspan="2">UPDATE statement Format 1</td><td>UPDATE table-name SET
column-name=updatevalue
[, columnname=update-value] ...
[WHERE {search-condition|
CURRENT OF cursor-name}]</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td></tr>
<tr><td>UPDATE table-name SET
repetition-column-name
[{subscript|*}]
=element-value[,repetition-column-name
[{subscript|*}]
=element-value] ...
[WHERE{retrieval-condition|
CURRENT OF cursor-name}]</td><td>Y</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
	UPDATE *tablename* ADD *repetition-column-name* [{*subscript*|*}] =ARRAY[*element-value*[,*element-value*] ...] [,*repetition-column-name* [{*subscript*|*}] =ARRAY[*element-value*[,*element-value*] ...) ... [WHERE{*retrieval-condition* |CURRENT OF *cursor-name*}]	Y	—	—	—	—
	UPDATE *table-name* DELETE *repetition-columnname* [{*subscript*|*}] [,*repetition-column-name*[{*subscript*|*}]... [WHERE{*retrieval-condition*| CURRENT OF *cursor-name*}]	Y	—	—	—	—
	[WITH ROLLBACK]	Y	Y	—	Y	—
UPDATE statement Format 2	UPDATE *table-name* SET ROW=*variable* [WHERE {*search-condition* | CURRENT OF *cursor-name*}] [WITH ROLLBACK]	—	—	—	—	—
UPDATE statement Format 3	FOR *variable* UPDATE *table-name* SET *column-name*=*update-value*[,*column-name*=*update-value*]... [WHERE *search-condition*] [WITH ROLLBACK]	—	—	—	—	—
UPDATE statement Format 4	FOR *variable* UPDATE *table-name* SET ROW=*variable* [WHERE *search-condition*] [WITH ROLLBACK]	—	—	—	—	—
DELETE statement Format 1	DELETE FROM *table-name* [WHERE {*search-condition* |CURRENT OF *cursor-name*}]	Y	Y	Y	Y	Y
	[WITH ROLLBACK]	Y	Y	—	Y	—

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
DELETE statement Format 2	FOR *variable* DELETE FROM *table-name* WHERE *search-condition*	—	—	—	—	—
	[WITH ROLLBACK]	—	—	—	—	—
PREPARE statement	PREPARE *SQL-statement-identifier* FROM{'*character-string*' | *variable*}	Y	Y	Y	Y	Y
	[WITH SQLNAME OPTION]	Y	Y	Y	—	—
SQL statements preprocessable by PREPARE statement	INSERT, UPDATE (without cursor), DELETE (without cursor), *dynamic* SELECT *statement*	Y	Y	Y	Y	Y
	PURGE TABLE, LOCK TABLE, CALL	Y	—	—	—	—
	Definition SQL ASSIGN LIST, DROP LIST, *dynamic* SELECT *statement Format 2*	—	—	—	—	—
DESCRIBE OUTPUT statement	DESCRIBE [OUTPUT] *SQL-statement-identifier* INTO [:] *SQL-Data-Area-name* [[:] *column-name-data-areaname*]	Y	Y	Y	Y	Y
DESCRIBE INPUT statement	DESCRIBE INPUT *SQL-statement-identifier* INTO [:] *SQL-data-area-name* [[:] *column-name-data-area-name*]	Y	—	—	—	—
EXECUTE statement Format 1	EXECUTE *SQL-statement-identifier*	Y	Y	Y	Y	Y
	{INTO *variable* [,*variable*]... | INTO DESCRIPTOR [:]*SQL -description-area-name*}	Y	—	—	—	—
	[{USING *variable* [,*variable*]... | USING DESCRIPTOR [:]*SQL -description-area-name*}]	Y	Y	Y	Y	Y

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
EXECUTE statement Format 2	EXECUTE *SQL-statement-identifier* {USING *array-variable*[,*array-variable*]... |USING DESCRIPTOR [:]*SQL-descriptor-area-name*} BY *variable* [ROWS]	—	—	—	—	—
SQL statements executable by EXECUTE statement	*SQL statements other than dynamic* SELECT *statement preprocessed by* PREPARE *statement*	Y	Y	Y	Y	Y
EXECUTE IMMEDIATE statement	EXECUTE IMMEDIATE {'*characterstring*' | *variable*}	Y	Y	Y	—	—
SQL statements executable by EXECUTE IMMEDIATE statement	INSERT, UPDATE (without cursor), DELETE (without cursor)	Y	Y	Y	—	—
	PURGE TABLE, LOCK TABLE, CALL	Y	—	—	—	—
	Definition SQL ASSIGN LIST, DROP LIST	—	—	—	—	—
PURGE TABLE statement	PURGE TABLE *table-name* [WITH ROLLBACK | NO WAIT]	Y	—	—	—	—
LOCK TABLE statement	LOCK TABLE *tablename* [, *tablename*] ... [IN {SHARE | EXCLUSIVE} MODE] [WITH ROLLBACK | NO WAIT]	Y	—	—	—	—
	UNTIL DISCONNECT	—	—	—	—	—
CALL statement[10]	CALL *RD-node-name*.*authorization-identifier*.*routine-identifier*(*argument-specification*)	Y	—	—	—	—
GET DIAGNOSTICS statement	GET DIAGNOSTICS...	Y[8]	Y[8]	Y[8]	Y[8]	Y[8]

Category	SQL statement format	Usable at distributed server				
		H	X	O	R	S
SET SESSION AUTHORIZATION statement	SET SESSION AUTHORIZATION :*embedded-variable-1* [{USING|IDENTIFIED BY} :*embedded-variable-2*]	—	—	—	—	—
FREE LOCATOR statement	FREE LOCATOR :*locator-reference* [,:*locator-reference*]...	—	—	—	—	—
Assignment statement	SET *assignment-destination=assignment-value*	—	—	—	—	—

The letter in each column under the *Usable at distributed server* column denotes the following:

H: HiRDB

X: XDM/RD

O: ORACLE

R: RDB1 E2

S: SQL/K

Legend:

Y: Can be used.

—: Cannot be used.

[1] The table name formats *RD-node-name.authorization-identifier.table-identifier* and [*authorization-identifier*.]*table-alias* can be used only when the default SQL connection is used as the current SQL connection.

[2] The table name format [*authorization-identifier*.]*table-identifier* can be used only when an SQL connection to a distributed RD-node is used as the current SQL connection.

[3] If a correlation name contains both 1- and 2-byte characters, it may not be processed by a distributed server.

[4] The following functions cannot be used:

- USER literal (SQL/K only)
- NULL literal (SQL/K only)

- Comparison operator `||` (RDB1 E2 only)
- Comparison operator <> (SQL/K only)
- Predicate and search condition `NOT` (SQL/K only)
- Scalar function (RDB1 E2 and SQL/K only)
- Query specifications `ALL` and `DISTINCT` (SQL/K only)

[5] When the `LENGTH` or `SUBSTR` function is used on an `MCHAR` type column, HiRDB processes the data length and location based on the number of characters, while XDM/RD processes based on the number of bytes; this means that the execution results are different. Also, in ORACLE, a character type column may contain both one-byte and two-byte characters, so processing is performed based on the number of bytes when the `LENGTH` or `SUBSTR` function is used.

[6] For RDB1 E2 or SQL/K, define using an inquiry specification rather than an inquiry equation.

[7] When `UNION[ALL]` is not specified, XDM/RD, RDB1 E2, and SQL/K use a different number for the sort specification item number in the `ORDER BY` clause from the number used by other systems. In XDM/RD, RDB1 E2, and SQL/K, the number that represents the position of the selection expression of the column to be used as the sort key (the position specified in the `SELECT` clause) is specified. In other systems, the number that represents the position of the column to be used as the sort key (the position specified in a derived table) is specified.

[8] Only errors that occur in the distributed server can be collected. For details about the errors that can occur in a distributed server, see *11.5 Handling distributed server errors*.

[9] Embedded variables and `?` parameters that have repetition structures cannot be used.

[10] Procedures that use the `PURGE TABLE`, `COMMIT`, or `ROLLBACK` statement cannot be executed.

11.4 Available data types

This section explains the data types of distributed servers and the variables that can be used in accessing a remote database under the distributed client facility.

11.4.1 Data types of variables usable in remote database access

In order to use a variable during access to a remote database, its data type must be supported by the distributed client facility. Table 11-6 lists the data types of variables that are supported by the distributed client facility.

Table 11-6: Data types of variables supported by distributed client facility

Classification	Data type
Numeric data	INT[EGER]
	SMALLINT
	DEC[IMAL]
	FLOAT
	SMALLFLT
Character data	CHAR[ACTER]
	VARCHAR
National character data	NCHAR
	NVARCHAR
Mixed character data	MCHAR
	MVARCHAR
Large-object data	BLOB

Variables of the following data types are not supported by the distributed client facility and cannot be used: date data type (DATE), time data type (TIME), date interval data type (INTERVAL YEAR TO DAY), time interval data type (INTERVAL HOUR TO SECOND), and ROW.

11.4.2 Correspondence between distributed server data types and HiRDB data types

Execution of the DESCRIBE statement converts distributed server data types into the corresponding HiRDB data types. The results are set in the SQL Descriptor Area. If no corresponding HiRDB data type exists for a particular distributed server data type, data code 0 is set in SQLDA.

A UAP must be created so that the DESCRIBE statement is executed first, then the desired column in the table at the distributed server can be accessed using variables of the appropriate data types that are set in the SQL Descriptor Area. However, there are some exceptions to this rule (e.g., accessing a DATE-type column using a CHAR-type variable).

(1) HiRDB distributed server

Table 11-7 shows the relationships between the data types that are set in the SQL Descriptor Area of a HiRDB distributed client after the DESCRIBE statement has executed and the data types of HiRDB.

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

HiRDB data type	Data type set in SQL Descriptor Area of HiRDB	Description
INTEGER	INTEGER	Integer (4-byte binary format)
SMALLINT	SMALLINT	Integer (2-byte binary format)
DECIMAL (*p*, *s*)	DECIMAL (*p*, *s*)	Fixed-point number ● Precision (total number of digits) = *p* ● Scale factor (number of digits following the decimal point) = *s* $1 \leq p \leq 29, 0 \leq s \leq p$
FLOAT	FLOAT	Double-precision floating-point number
SMALLFLT	SMALLFLT	Single-precision floating-point number
CHAR (*n*)	CHAR (*n*)	Fixed-length character string
VARCHAR (*n*)	VARCHAR (*n*)	Variable-length character string
NCHAR (*n*)	NCHAR (*n*)	Fixed-length national character string
NVARCHAR (*n*)	NVARCHAR (*n*)	Variable-length national character string
MCHAR (*n*)	MCHAR (*n*)	Fixed-length mixed character string
MVARCHAR (*n*)	MVARCHAR (*n*)	Variable-length mixed character string
DATE[1]	DATE	Date
TIME[2]	TIME	Time
INTERVAL YEAR TO DAY	INTERVAL YEAR TO DAY	Date interval

HiRDB data type	Data type set in SQL Descriptor Area of HiRDB	Description
INTERVAL HOUR TO SECOND	INTERVAL HOUR TO SECOND	Time interval
BLOB	BLOB	Binary
ROW	ROW	ROW type

[1] DATE-type variables are not supported by the distributed client facility. However, access to DATE-type columns can be performed by using CHAR(10)-type input variables, as in local access to HiRDB.

[2] TIME-type variables are not supported by the distributed client facility. However, access to TIME-type columns can be performed by using CHAR(8)-type input variables, as in local access to HiRDB.

(2) XDM/RD distributed server

Table 11-8 shows the relationships between the data types that are set in the SQL Descriptor Area of a HiRDB distributed client after the DESCRIBE statement has executed and the data types of XDM/RD.

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

XDM/RD data type	Data type set in SQL Descriptor Area of HiRDB	Description
INTEGER	INTEGER	Integer (4-byte binary format)
SMALLINT	SMALLINT	Integer (2-byte binary format)
DECIMAL (*p*, *s*) LARGE DECIMAL (*p*, *s*)	DECIMAL (*p*, *s*)	Fixed-point number • Precision (total number of digits) = *p* • Scale factor (number of digits following the decimal point) = *s* $1 \leq p \leq 29, 0 \leq s \leq p$
FLOAT	FLOAT	Double-precision floating-point number
SMALLFLT	SMALLFLT	Single-precision floating-point number
CHAR (*n*)	CHAR (*n*)	Fixed-length character string
VARCHAR (*n*) LONG VARCHAR (*n*)	VARCHAR (*n*)	Variable-length character string

XDM/RD data type	Data type set in SQL Descriptor Area of HiRDB	Description
NCHAR (*n*)	NCHAR (*n*)	Fixed-length national character string
NVARCHAR (*n*)	NVARCHAR (*n*)	Variable-length national character string
LONG NVARCHAR (*n*)		
MCHAR (*n*)	MCHAR (*n*)	Fixed-length mixed character string
MVARCHAR (*n*)	MVARCHAR (*n*)	Variable-length mixed character string
LONG MVARCHAR (*n*)		
DATE[1]	DATE	Date
TIME[2]	TIME	Time
INTERVAL YEAR TO DAY	INTERVAL YEAR TO DAY	Date interval
INTERVAL HOUR TO SECOND	INTERVAL HOUR TO SECOND	Time interval
ROW	ROW	ROW type

[1] DATE-type variables are not supported by the distributed client facility. However, access to DATE-type columns can be performed by using CHAR(10)-type input variables, as in local access to HiRDB.

[2] TIME-type variables are not supported by the distributed client facility. However, access to TIME-type columns can be performed by using CHAR(8)-type input variables, as in local access to HiRDB.

(3) ORACLE distributed server

Table 11-9 shows the relationships between the data types that are set in the SQL Descriptor Area of a HiRDB distributed client after the DESCRIBE statement has executed and the data types of ORACLE.

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

ORACLE data type	Data type set in SQL Descriptor Area of HiRDB	Description
`NUMBER` (*p, s*)	`DECIMAL` (*p, s*)	Fixed-point number ● Precision (total number of digits) = *p* ● Scale factor (number of digits following the decimal point) = *s* $1 \leq p \leq 29, 0 \leq s \leq p$
`NUMBER` (*p, s*)[1]	`DECIMAL` (*p*, `0`)	Fixed-point number ● Precision (total number of digits) = *p* ● Scale factor (number of digits following the decimal point) = *s* $1 \leq p \leq 29, s < 0$
`NUMBER` (*p, s*)[2]	`DECIMAL` (*p, p*)	Fixed-point number ● Precision (total number of digits) = *p* ● Scale factor (number of digits following the decimal point) = *s* $1 \leq p \leq 29, s > p$
`NUMBER`[3]	`FLOAT`	Double-precision floating-point number
`NUMBER` (*p, s*)[4]		Fixed-point number ● Precision (total number of digits) = *p* ● Scale factor (number of digits following the decimal point) = *s* $30 \leq p \leq 38, 0 \leq s \leq p$
`CHAR` (*n*)	`CHAR` (*n*)	Fixed-length character string $n \leq 255$
`VARCHAR2` (*n*)	`VARCHAR` (*n*)	Variable-length character string $n \leq 2000$
`LONG`	`VARCHAR` (32000)[5]	Variable-length character string
`DATE`[6]	`DATE`	Date
`ROW`	Data code `0`	Data code `0` is set in SQLDA because no corresponding data type is found in the HiRDB database.
`LONG ROW`		
`ROWID`		
`MLSLABEL`		

[1] If $s < 0$ and data is entered by the `UPDATE` or `INSERT` statement into an appropriate column based on the results of executing the `DESCRIBE` statement, the data is subject

to rounding depending on the actual value of the input data.

2 If $s > p$ and data is entered by the `UPDATE` or `INSERT` statement into an appropriate column based on the results of executing the `DESCRIBE` statement, a precision error can occur depending on the actual value of the input data.

3 The permissible range of the absolute values of `NUMBER`-type data is `1.0E-129` through `9.99...E125` with a precision of 38 decimal digits. `NUMBER`-type columns can be accessed using a `FLOAT`-type output variable at the expense of reduced numeric value precision.

4 In the case of $p > 29$, `NUMBER`-type columns can be accessed using a `FLOAT`-type output variable at the expense of reduced numeric value precision.

5 A maximum of 2 GB of character data can be stored as `LONG`-type data. Therefore, a `VARCHAR(32000)`-type output variable can retrieve only part of the data in some cases.

6 `DATE`-type variables are not supported by the distributed client facility. However, access to `DATE`-type columns can be performed by using `CHAR(10)`-type input variables, as in local access to HiRDB. Although the `DATE` type of ORACLE contains time information as part of the data, only the date information can be accessed from a distributed client.

(4) RDB1 E2 distributed server

Table 11-10 shows the relationships between the data types that are set in the SQL Descriptor Area of a HiRDB distributed client after the `DESCRIBE` statement has been executed and the data types of RDB1 E2.

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

RDB1 E2 data type	Data type set in SQL Descriptor Area of HiRDB	Description
`INTEGER`	`INTEGER`	Integer (4-byte binary format)
`SMALLINT`	`SMALLINT`	Integer (2-byte binary format)
`DECIMAL` *(p, s)*	`DECIMAL` *(p, s)*	Fixed-point number Precision (total number of digits) = p Scale factor (number of digits following the decimal number) = s $1 \leq p \leq 29, 0 \leq s \leq p$
`FLOAT`	`FLOAT`	Double-precision floating-point number
`SMALLFLT`	`SMALLFLT`	Single-precision floating-point number

RDB1 E2 data type	Data type set in SQL Descriptor Area of HiRDB	Description
CHAR *(n)*	CHAR *(n)*	Fixed-length character string $n \leq 254$
VARCHAR *(n)*	VARCHAR *(n)*	Variable-length character string $n \leq 254$
LONG VARCHAR *(n)*	VARCHAR *(n)*	Variable-length character string $255 \leq n \leq 4000$
NCHAR *(n)*	NCHAR *(n)*	Fixed-length national character string $n \leq 127$
NVARCHAR *(n)*	NVARCHAR *(n)*	Variable-length national character string $n \leq 127$
ROW	ROW	ROW type

(5) SQL/K distributed server

Table 11-11 shows the relationships between the data types that are set in the SQL Descriptor Area of a HiRDB distributed client after the DESCRIBE statement has been executed and the data types of SQL/K.

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

SQL/K data type	Data type set in SQL Descriptor Area of HiRDB	Description
INTEGER	INTEGER	Integer (4-byte binary format)
SMALLINT	SMALLINT	Integer (2-byte binary format)
DECIMAL *(p,s)*	DECIMAL *(p,s)*	Fixed-point number Precision (total number of digits) = *p* Scale factor (number of digits following the decimal number) = *s* $1 \leq p \leq 29, 0 \leq s \leq$ p
CHAR *(n)*	CHAR *(n)*	Fixed-length character string $n \leq 32000$
CHAR *(n)*	CHAR(32000) [1]	Fixed-length character string $n > 32000$

SQL/K data type	Data type set in SQL Descriptor Area of HiRDB	Description
NCHAR *(n)*	NCHAR *(n)*	Fixed-length national character string $n \leq 16000$
NCHAR *(n)*	NCHAR (16000)[2]	Fixed-length national character string $n > 16000$
MCHAR *(n)*	MCHAR *(n)*	Fixed-length mixed character string $n \leq 32000$
MCHAR *(n)*	MCHAR(32000)[1]	Fixed-length mixed character string $n > 32000$
LARGE INT	Data code 0	Data code 0 is set in SQLDA because no corresponding data type is found in the HiRDB database.
NUMERIC TRAILING *(P,S)*		
NUMERIC UNSIGNED *(P,S)*		
XCHAR *(n)* BIT *(n)*		

[1] If $n > 32000$, using a VARCHAR(32000)-type output variable may result in partial data search only.

[2] If $n > 16000$, using an NVARCHAR(16000)-type variable may result in partial data search only.

11.5 Handling distributed server errors

This section explains the handling of errors that may occur in a distributed server.

11.5.1 Return codes set by the distributed client

If an error occurs during execution of an SQL statement at a distributed server, the HiRDB of the distributed client sets a return code (`SQLCODE`) in the SQLCODE variable.

Table 11-12 shows the SQLCODEs that are set by the distributed client when errors occur at the distributed server.

Table 11-12: SQLCODEs set by distributed client when errors occur at distributed server

SQLCODE	Description
`-861`	The distributed server returned a negative SQLCODE during remote database access.
`-862`	The distributed server returned an RDA error during remote database access.

Note

Error information (return code or message text from the distributed server) that is returned from the distributed server is inserted into messages corresponding to the above SQLCODEs. However, because there are restrictions on the length of information that can be inserted into a message, sometimes only part of the message text returned from the distributed server is displayed.

11.5.2 Obtaining and using detailed error information

When an error occurs in the distributed server and an SQLCODE shown in Table1113 is set in the distributed client, detailed information can be obtained by issuing the `GET DIAGNOSTICS` statement. The type of the DBMS of the distributed server can also be determined by displaying as part of the detailed information the contents of the SQLCAIDE area of the SQL Communications Areas.

The `GET DIAGNOSTICS` statement can obtain the following four types of detailed information:

- Return code returned by the distributed server indicating the execution results of an SQL statement (only when SQLCODE of the distributed client is `-861`).
- The entire message text returned by the distributed server as a result of executing the SQL statement
- The RD node name of the distributed server at which the error occurred

- The diagnostic information stored in the diagnostics area of the distributed server (in the case of the `CALL` statement)

The `GET DIAGNOSTICS` statement cannot obtain detailed information on an error that occurs while detailed information is being obtained as a result of execution of the `GET DIAGNOSTICS` statement.

Table 11-13 shows the statement information items that can be obtained by the `GET DIAGNOSTICS` statement when an error occurs at the distributed server.

Table 11-13: Statement information items obtained by GET DIAGNOSTICS statement when error occurs at distributed server

Statement information item name	SQL statement	Contents
NUMBER	Other than `CALL` statement	1
	`CALL` statement	1 + number of errors in distributed server diagnostic area
MORE	Other than `CALL` statement	`N`

	`CALL` statement	`Y`
		Number of errors at distributed server is greater than number of errors in distributed server diagnostic area
		`N`
		Number of errors at distributed server equals number of errors in distributed server diagnostic area

When an error occurs at the distributed server, condition information items can be obtained by the `GET DIAGNOSTICS` statement by specifying condition number 1. If a `CALL` statement error occurs at the distributed server, diagnostic information in the distributed server diagnostic information area can be obtained by the `GET DIAGNOSTICS` statement by specifying condition number 2 or greater.

Table 11-14 shows the condition information items that can be obtained when an error occurs at the distributed server by specifying condition number 1.

Table 11-14: Condition information items obtained by specifying condition number 1 (error at the distributed server)

Condition information item name	Information	Contents
`RETURNED_SQLCODE`	Available	SQLCODE returned by distributed server[1]
	None	`0`[2]

Condition information item name	Information	Contents
ERROR_POSITION	None	0
ERROR_SQL_NO	None	0
ERROR_SQL	None	One blank
ROUTINE_TYPE	None	One blank
ROUTINE_SCHEMA	None	One blank
ROUTINE_NAME	None	One blank
MESSAGE_TEXT	Available	Message text returned by distributed server
	None	One blank
RDNODE_NAME	Available	RD node name of distributed server

[1] Applicable when the SQLCODE of the distributed server is -861.

[2] A 0 is set in SQLCODE because no SQLCODE is returned when the SQLCODE of the distributed server is -862.

11.6 Notes about using a distributed database

This section provides notes about using a distributed database.

11.6.1 Notes about using a distributed client

(1) Synchronous update operation on remote and local databases is not supported

If a communications error occurs during COMMIT processing for updating a remote database, it cannot be determined solely on the basis of information provided by the distributed client whether or not the distributed server has completed its COMMIT processing or performed a rollback. In such a case, the user must check the server's status.

When a remote database is updated, it is important that a local database not be updated within the same transaction. If a remote database and a local database are both updated and the COMMIT processing fails, the local database reverts to its status at the preceding synchronization point but the remote database is updated. When only the remote database becomes updated, it may be difficult to restore it to the status at the synchronization point.

(2) Space character conversion

(a) Data substitution and comparison

During data substitution or comparison, the system ignores the pd_space_level operand in the system common definitions and the value of PDSPACELVL in the client environment definition specified with the distributed client's HiRDB.

The space conversion level specified at the distributed server takes effect (that is, the value of the pd_space_level operand in the system common definitions if the distributed server is HiRDB and the value of KEIS CODE SPACE LEVEL in the RD environment definitions if the distributed server is XDM/RD). If necessary, specify the space conversion level at the distributed server.

(b) Data retrieval

- **If the distributed client's pd_space_level operand in the system common definitions or in PDSPACELVL in the client environment definitions is set to 0 or 1**

Space conversion is not executed at the distributed client.

If a space conversion level is specified at the distributed server, the retrieval result depends on the distributed server's specifications. Table 11-15 shows the space conversion to be executed when the distributed server is HiRDB.

Table 11-15: Space conversion when the distributed server is HiRDB

Distributed server		Data type of column to be accessed		
DBMS	**Space conversion level**	**NCHAR, NVARCHAR**	**MCHAR, MVARCHAR**	**CHAR, VARCHAR**
HiRDB	1	—[1]	—[2]	—
	3	Y[3]	—[2]	—

Legend:

Y: Space conversion is executed when data is fetched.

—: Space conversion is not executed when data is fetched.

[1] If space conversion is executed when data is stored, two consecutive single-byte spaces are converted to a single double-byte space. If the same space conversion level is specified when data is stored as for data retrieval, all spaces in the resulting retrieval data become double-byte spaces.

[2] If space conversion is executed when data is stored, each double-byte space is converted to two consecutive single-byte spaces. If the same space conversion level is specified when data is stored as for data retrieval, all spaces in the resulting retrieval data become single-byte spaces.

[3] A single double-byte space is converted to two consecutive single-byte spaces during data retrieval; therefore, all spaces in the resulting retrieval data become single-byte spaces.

- **If the distributed client's pd_space_level operand in the system common definitions or in PDSPACELVL in the client environment definitions is set to 3**

The distributed client's HiRDB converts each double-byte space to two consecutive single-byte spaces in the resulting retrieval data. This applies not only to the data types `NCHAR` and `NVARCHAR` but also to `MCHAR`, `MVARCHAR`, `CHAR`, and `VARCHAR`. Therefore, all the spaces in the resulting retrieval data to be returned to the UAP become single-byte spaces.

11.6.2 Notes about using a distributed server

- When a UAP operating under a DBMS other than HiRDB accesses a HiRDB database by using the distributed server facility of HiRDB, character strings such as names that are specified in SQL statements must be spelled carefully. Whereas character strings are not case sensitive in XDM/RD, HiRDB requires that lowercase character strings be enclosed in quotation marks in order to distinguish them from uppercase character strings. Therefore, if an SQL statement that is used to access a HiRDB database contains a character string that must be recognized as lowercase characters, it must be enclosed in quotation marks.

- If a table identifier or column name contains both 1- and 2-byte characters, that table cannot be accessed from an XDM/RD UAP.

Chapter

12. Command Execution from UAPs

This chapter explains how to execute commands from UAPs.

This chapter contains the following sections:

12.1 Overview

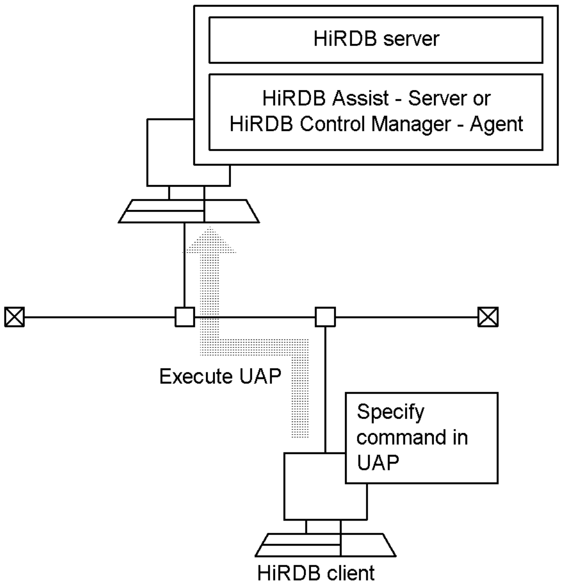
You can execute commands by specifying them in a UAP. The specified commands are executed at the HiRDB server. Such commands include HiRDB's operation commands, utilities, and OS commands.

You use `COMMAND EXECUTE` of SQL to execute commands from a UAP. Because execution of commands from a UAP is implemented by collaboration between the HiRDB client and HiRDB Control Manager - Agent, HiRDB Control Manager - Agent must be installed on the HiRDB server. For details about HiRDB Control Manager - Agent, see the respective `README.TXT`.

Command execution from a UAP can be used only if the UAP is written in C.

Figure 12-1 shows an overview of command execution from UAPs.

Figure 12-1: Overview of command execution from UAPs

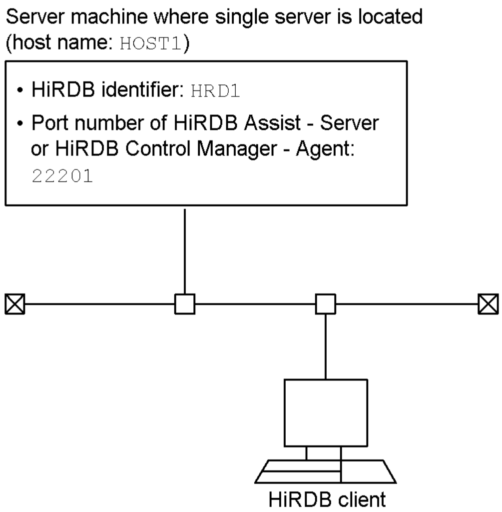


12.2 Preparations for executing commands from a UAP

(1) HiRDB/Single Server

This section uses a sample UAP that executes data loading (database load utility). Figure 12-2 shows a sample server-client configuration for a HiRDB/Single Server.

Figure 12-2: Sample server-client configuration for a HiRDB/Single Server



To execute a data-loading UAP with the server-client configuration shown in Figure 12-2, you need to define the following information beforehand:

1. Specify the following client environment definitions:

   `PDSYSTEMID`

   Specifies the HiRDB server's HiRDB identifier (`HRD1`).

   `PDASTHOST`

   Specifies the HiRDB Control Manager - Agent's host name (`HOST1`).

   `PDASTPORT`

   Specifies the HiRDB Control Manager - Agent's port number (`22201`).

2. Prepare the control information file and input data file needed for loading data at the HiRDB server.

3. Suppose that the HiRDB administrator's user name is `USERA` (password: `USERA`) and the owner of the table subject to data loading is `USERB` (password: `USERB`).

In this case, specify the following client environment definitions:

```
PDASTUSER=USERA/USERA
PDUSER=USERB/USERB
```

You can now execute the data-loading UAP. For details about each client environment definition, see *6.6.4 Environment definition information*.

The following shows a sample UAP for loading data:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                   /* CmdLine variable */
long ReturnCode;                   /* variable receiving return
                                        code */
long OutBufLen;                   /* size of area for receiving
                                        execution result */
long CmdRetCode;                      /* variable for receiving
                                      executed command's return
                                        code */
long OutDataLen;                  /* variable for receiving the
                                  length of execution result */
PDOUTBUF OutBuf;                       /* area for receiving
execution
                                        result */
char EnvGroup[256];               /* environment variable group
                                        name variable */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10");  /* specifying execution
command
                                      line command line (RDAREA
                                        shutdown) */
OutBuf = malloc(30000);             /* allocating the execution
                                       result receiving area */
if (OutBuf == NULL){              /* memory allocation error */
printf("Memory allocation error\n");
return ;
}
OutBufLen = 30000 ;                   /* specifying the size of
                                        execution result
                                        receiving area */
EnvGroup[0] = '\0' ;               /* specifying no environment
                                        variable group */
```

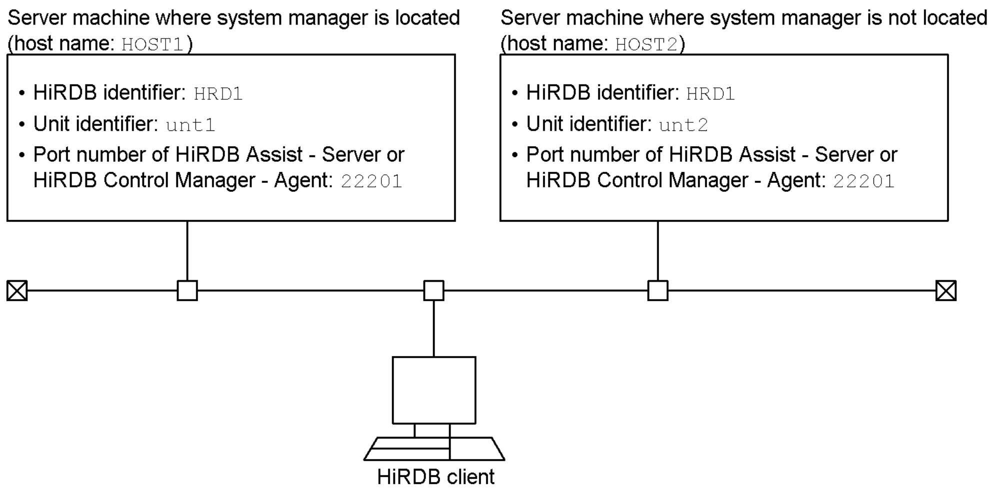
```
/* Command execution */
EXEC SQL COMMAND EXECUTE :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                        /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                   /* if command execution is
                                          normal */

/* Specifying execution command line (to execute dataloading) */
strcpy(CmdLine,"pdload  -i c  -be STOCK c:\HiRDB_S\conf\LOAD");
EXEC SQL COMMAND EXECUTE :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                /* if COMMAND EXECUTE terminates
                                          normally */
if (CmdRetCode==0) {                   /* if command execution is
                                          normal */
printf("pdload command successfully\n");
printf("%s\n", OutBuf);
} else {                            /* execution command error */
printf("pdload command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                              /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                            /* execution command error */
printf("pdhold command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
strcpy(CmdLine,"pdrels -r RDDATA10");
                                 /* specifying execution command
                                          line (RDAREA shutdown
                                          release) */
EXEC SQL COMMAND EXECUTE :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {
                                        /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode!=0) {                /* execution command error */
printf("pdrels command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                              /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
```

```
}
} else {                                    /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
return ;
}
```

(2) HiRDB/Parallel Server

This section uses a sample UAP that executes data loading (database load utility). Figure 12-3 shows a sample server-client configuration for a HiRDB/Parallel Server.

Figure 12-3: Sample server-client configuration for a HiRDB/Parallel Server



To execute a data-loading UAP with the server-client configuration shown in Figure 12-3, you need to define the following information beforehand:

1. Specify the following client environment definitions:

   `PDSYSTEMID`

   Specifies the HiRDB server's HiRDB identifier (`HRD1`).

   `PDASTHOST`

   Specifies the HiRDB Control Manager - Agent's host name (`HOST1`). For a HiRDB/Parallel Server, specify the host name of the server machine where the system manager is located.

   `PDASTPORT`

   Specifies the HiRDB Control Manager - Agent's port number (`22201`).

2. Prepare the control information file and input data file needed for loading data at the HiRDB server.
3. Suppose that the HiRDB administrator's user name is `USERA` (password: `USERA`) and the owner of the table subject to data loading is `USERB` (password: `USERB`). In this case, specify the following client environment definitions:
   ```
   PDASTUSER=USERA/USERA
   PDUSER=USERB/USERB
   ```

You can now execute the data-loading UAP. For details about each client environment definition, see *6.6.4 Environment definition information*.

The following shows a sample UAP for loading data:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC SQL BEGIN DECLARE SECTION;
char CmdLine[30000];                        /* CmdLine variable */
long ReturnCode;                        /* variable receiving return
                                            code */
long OutBufLen;                        /* size of area for receiving
                                             execution result */
long CmdRetCode;                          /* variable for receiving
                                          executed command's return
                                              code */
long OutDataLen;                       /* variable for receiving the
                                       length of execution result */
PDOUTBUF OutBuf;                            /* area for receiving
execution
                                              result */
char EnvGroup[256];                    /* environment variable group
                                            name variable */
EXEC SQL END DECLARE SECTION;

void main()
{
strcpy(CmdLine,"pdhold -r RDDATA10");  /* specifying execution
command
                                      line (RDAREA shutdown) */
OutBuf = malloc(30000);              /* allocating the execution
                                        result receiving area */
if (OutBuf == NULL){                 /* memory allocation error */
printf("Memory allocation error\n");
return ;
}
OutBufLen = 30000 ;                      /* specifying the size of
                                       execution result receiving
```

```
                                          area */
EnvGroup[0] = '\0' ;                /* specifying no environment
                                          variable group */

/* Command execution */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {      /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                  /* if command execution is
                                          normal */

/* Specifying execution command line (to execute dataloading) */
strcpy(CmdLine,"pdload  -i c  -be STOCK c:\HiRDB_P\conf\LOAD");
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {      /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode==0) {                  /* if command execution is
                                          normal */
printf("pdload command successfully\n");
printf("%s\n", OutBuf);
} else {                           /* execution command error */
printf("pdload command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else {                             /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
} else {                           /* execution command error */
printf("pdhold command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
strcpy(CmdLine,"pdrels -r RDDATA10");  /* specifying execution
command
                                         line (RDAREA shutdown
                                          release) */
EXEC  SQL  COMMAND  EXECUTE  :CmdLine, :ReturnCode, :OutBufLen,
:OutDataLen, :OutBuf, :CmdRetCode, :EnvGroup ;
if (ReturnCode == p_rdb_RC_NORM) {      /* if COMMAND EXECUTE
                                         terminates normally */
if (CmdRetCode!=0) {               /* execution command error */
printf("pdrels command Error,Code = %d\n", CmdRetCode);
printf("%s\n", OutBuf);
}
} else{                              /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
```

```
printf("%s\n", OutBuf);
}
} else {                                 /* COMMAND EXECUTE error */
printf("COMMAND EXECUTE Error,Code = %d\n", ReturnCode);
printf("%s\n", OutBuf);
}
return ;
}
```

12.3 Command executability

Some HiRDB commands can be executed from UAPs and some cannot. Table 12-1 shows whether each command is executable from UAPs.

Table 12-1: Command executability from UAPs

Type	Command	Description	Executability from UAP
System operation	pdadmvr	Gets the HiRDB version information.	E
	pdcat	Displays file contents.	E
	pdchgconf	System reconfiguration command	--
	pdconfchk	Checks system definitions.	--
	pdcspool	Deletes troubleshooting information.	E
	pdgen	Generates the system (system generator).	--
	pdgeter	Acquires error information.	--
	pditvtrc	Periodically gets the HiRDB status.	E
	pditvstop	Stops periodic acquisition of the HiRDB status.	E
	pdjarsync	Manipulates JAR files.	E
	pdlistls	Displays list definition information.	E
	pdlodsv	Reduces the size of the installation directory.	--
	pdls	Displays HiRDB system status.	E
	pdmemsv	Saves memory space.	--
	pdntenv	Sets the HiRDB operation environment.	--
	pdobjconv	Migrates SQL objects into 64-bit-mode HiRDB.	E
	pdopsetup	Installs an additional HiRDB program product.	--
	pdsetup	Registers or deletes a HiRDB system in the OS.	--
	pdsvhostname	Displays the server host name.	--

Type	Command	Description	Executability from UAP
	pdvrup	Upgrades HiRDB.	—
HiRDB file system	pdfbkup	Backs up the HiRDB file system.	E
	pdfls	Displays HiRDB file system information.	E
	pdfmkfs	Initializes a HiRDB file system area.	E
	pdfrm	Deletes a HiRDB file.	E
	pdfrstr	Restores the HiRDB file system.	E
	pdfstatfs	Displays the status of a HiRDB file system area.	E
Log files	pdlogadpf	Allocates a log file.	E
	pdlogatul	Controls the automatic log unloading facility.	—
	pdlogchg	Changes the status of a log file.	E
	pdlogcls	Closes a log file.	E
	pdloginit	Initializes a log file.	E
	pdlogls	Displays log file information.	E
	pdlogopen	Opens a log file.	E
	pdlogrm	Deletes a log file.	E
	pdlogswap	Swaps log files.	E
	pdlogsync	Collects a synchronization point dump.	E
	pdlogucat	Displays unload log file information.	E
	pdlogunld	Unloads a log file.	E
Status files	pdstscls	Closes an open status file.	E
	pdstsinit	Initializes a status file.	E
	pdstsopen	Opens a status file.	E
	pdstsrm	Deletes a status file.	E
	pdstsswap	Swaps status files.	E
HiRDB startup and termination	pdstart	Starts a HiRDB system, unit, or server.	E
	pdstop	Terminates a HiRDB system, unit, or server.	E

Type	Command	Description	Executability from UAP
Statistics log	pdstbegin	Starts output of statistical information.	E
	pdstend	Stops output of statistical information.	E
	pdstjswap	Swaps statistics log files.	E
	pdstjsync	Copies the contents of the statistics log buffer to the statistics log file.	—
RDAREAs	pdclose	Closes RDAREAs.	E
	pddbls	Displays the status of RDAREAs.	E
	pdhold	Shuts down RDAREAs.	E
	pdopen	Opens RDAREAs.	E
	pdrels	Releases RDAREAs from shutdown status.	E
	pddbfrz	Executes frozen update of full HiRDB files in the user LOB RDAREA.	E
	pdrdrefls	Displays related RD area information.	E
Global Buffer	pdbufls	Displays global buffer information.	E
	pdbufmod	Dynamically changes the global buffer.	E
Transaction control	pdcmt	Commits a transaction.	E
	pdfgt	Forcibly terminates a transaction.	E
	pdrbk	Rolls back a transaction.	E
	pdtrndec	Forcibly completes uncompleted transactions automatically.	—
Process control	pdcancel	Forcibly terminates UAP and utility processing.	E
	pdchprc	Changes the number of server process activations.	E
	pdkill	Stops a process forcibly.	—
	pdpfresh	Refreshes a server process.	E
	pdrpause	Restarts the process service.	—

Type	Command	Description	Executability from UAP
Update to HiRDB update version	`pdprgcopy`	Copies the HiRDB update version.	--
	`pdprgrenew`	Updates to the HiRDB update version.	--
Update to HiRDB update version	`pdprgcopy`	Copies the HiRDB update version.	--
	`pdprgrenew`	Updates HiRDB to the update version.	--
HiRDB Datareplicator linkage	`pdrplstart`	Starts HiRDB Datareplicator linkage.	--
	`pdrplstop`	Stops HiRDB Datareplicator linkage.	--
Directory Server linkage facility	`pdgrprfl`	Refreshes user information and role information.	--
	`pdusrchk`	Checks the user's compatibility with the Directory Server.	E
Inner replica facility	`pddbchg`	Switches the replica status of the replica RDAREA.	E
Updatable online reorganization	`pdorbegin`	Commits the database for online reorganization.	E
	`pdorcheck`	Checks the application conditions for online reorganization.	E
	`pdorchg`	Switches the current RDAREA for online reorganization.	E
	`pdorcreate`	Creates a reflection environment for online reorganization.	E
	`pdorend`	Executes reflection of online reorganization.	E
Security audit	`pdaudbegin`	Starts audit trail acquisition.	E
	`pdaudend`	Stops audit trail acquisition.	E
	`pdaudrm`	Deletes audit trail files that are shut down.	E
	`pdaudswap`	Swaps the current audit trail file.	E
Connection security facility	`pdacunlck`	Unlocks the consecutive certification failure account lock state.	--
HiRDB External Data Access facility	`pddbadset`	Sets up the HiRDB External Data Access Adapter.	--

Type	Command	Description	Executability from UAP
Real Time SAN Replication	pdrisechk	Checks the configuration of Real Time SAN Replication.	—
	pdrisedbto	Inherits the Real Time SAN Replication database.	—
	pdriseset	Sets the site status of Real Time SAN Replication.	—
SQL trace acquisition	pdclttrc	Dynamically acquires an SQL trace.	E
SQL object information display	pdobils	Displays statistical information for an SQL object.	—
SQL compilation	pdcbl	COBOL preprocessor	—
	pdcpp	C preprocessor	—
	pdocb	OOCOBOL preprocessor	—
	pdocc	C++ preprocessor	—
Database creation	pdinit	Database initialization utility	—
	pddef	Database definition utility	—
	pdload	Database load utility	E
	pdsql*	Interactive SQL execution utility	—
	pddefrev	Generates a definition SQL statement.	—
Database operations	pdmod	Database structure modification utility	E
	pdrorg	Database reorganization utility	E
	pdexp	Dictionary import/export utility	—
	pdrbal	Rebalancing utility	—
	pdreclaim	Free page release utility	E
	pdpgbfon	Global buffer residence utility	E
Tuning	pdstedit	Statistics analysis utility	—
	pddbst	Database condition analysis utility	E
	pdgetcst	Optimizing information collection utility	—
	pdvwopt	Access path display utility	—

Type	Command	Description	Executability from UAP
Database error handling	`pdcopy`	Database copy utility	E
	`pdbkupls`	Displays backup file information.	E
	`pdrstr`	Database recovery utility	E
Plug-in-related	`pdplgrgst`	Registers a plug-in.	E
	`pdplgset`	Sets up a plug-in.	—
	`pdreginit`	Registry facility initialization utility	E

E: Can be executed from UAPs.

—: Cannot be executed from UAPs.

Notes

1. The following commands cannot be used in the UNIX version:

   `pdkill`, `pdntenv`

2. The following commands cannot be used in the Windows version:

   `pddbadset`, `pdgen`, `pdgeter`, `pditvtrc`, `pditvstop`, `pdlodsv`, `pdmemsv`, `pdobjconv`, `pdopsetup`, `pdplgset`, `pdrisechk`, `pdrisedbto`, `pdriseset`, `pdrpause`, `pdsetup`

* This command does not exist in the Windows version; instead, the HiRDB SQL Executer is used.

Chapter

13. HiRDB Access from ODBC Application Programs

This chapter explains the OBDC driver installation procedure, ODBC functions, and tuning and troubleshooting procedures that are necessary when ODBC application programs access HiRDB.

This chapter contains the following sections:

13.1 ODBC application programs

Examples of ODBC application programs are Microsoft Access and Microsoft Excel. The ODBC driver must be installed before these application programs can access HiRDB. For information on ODBC driver installation, see *13.2 Installing the ODBC2.0 driver*. You can also access HiRDB via the ODBC driver from a UAP that uses the ODBC functions provided by HiRDB. For information on the ODBC functions provided by HiRDB, see *13.4 ODBC functions provided by HiRDB*.

When the ODBC driver is used, you can access HiRDB from a UAP that uses the ODBC3.x interface.

13.2 Installing the ODBC2.0 driver

To run an ODBC application program or a UAP that uses ODBC functions, you need to install the ODBC driver in the HiRDB client beforehand. To execute a UAP via ODBC on the HiRDB server, you also need to install the ODBC driver in the HiRDB server.

This section presents the ODBC driver installation procedure. Be sure to exit all Windows applications before starting the installation.

To install the ODBC driver:.

1. Execute `hcd_inst.exe` found on the integrated CD-ROM to start Hitachi Integrated Installer.
2. At the Hitachi Integrated Installer screen, select one of the following, and then click the **Execute Installation** button to start the HiRDB setup program:

   For the UNIX version:

   - For HiRDB/Run Time: **HiRDB/Run Time**
   - For HiRDB/Developer's Kit: **HiRDB/Developer's Kit**

   For the Windows version:

   - For a HiRDB/Single Server: **HiRDB/Single Server**
   - For a HiRDB/Parallel Server: **HiRDB/Parallel Server**
3. Perform the following operation; the setup program of the selected program process starts:

   For the UNIX version:

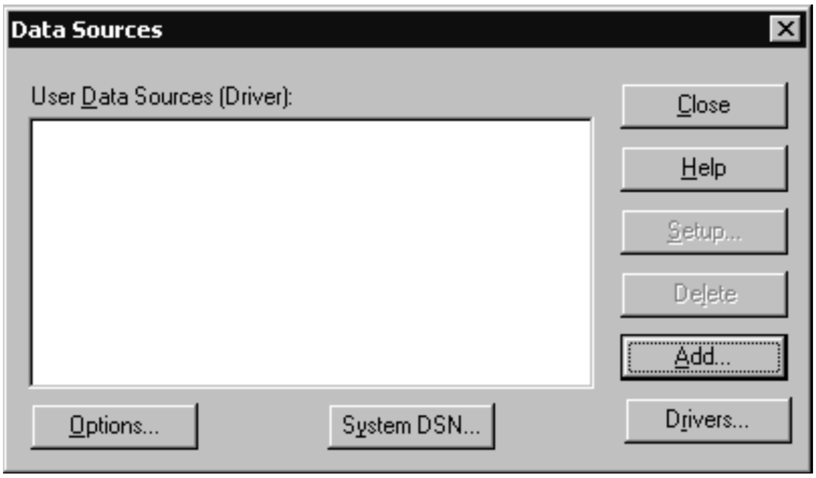
   From the Select Program Process window of the HiRDB setup program, select one of the following, and then click the **Next** button:

   - For HiRDB/Run Time: **Previous Product**, and then **HiRDB/Run Time (ODBC 2.0)**
   - For HiRDB/Developer's Kit: **Previous Product**, and then **HiRDB/Developer's Kit (ODBC 2.0)**

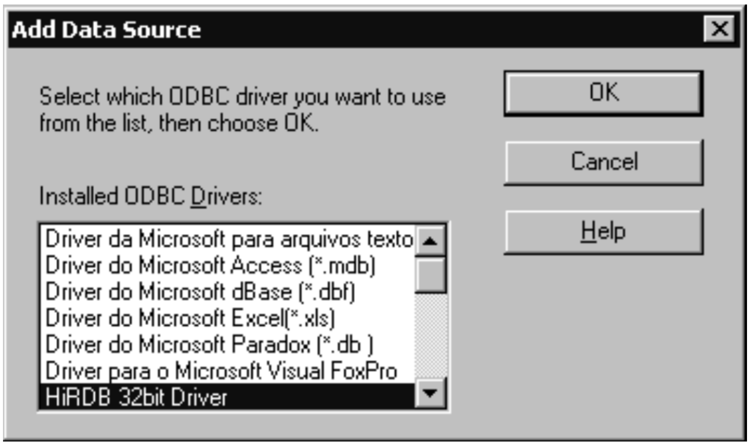
   For the Windows version:

   From the Select Program Process window of the HiRDB setup program, select **Previous Product** and **HiRDB/Run Time (ODBC 2.0)**, and then click the **Next** button.
4. Select the displayed HiRDB driver and choose the **OK** button. Installation does not take place if you choose **OK** without selecting anything.

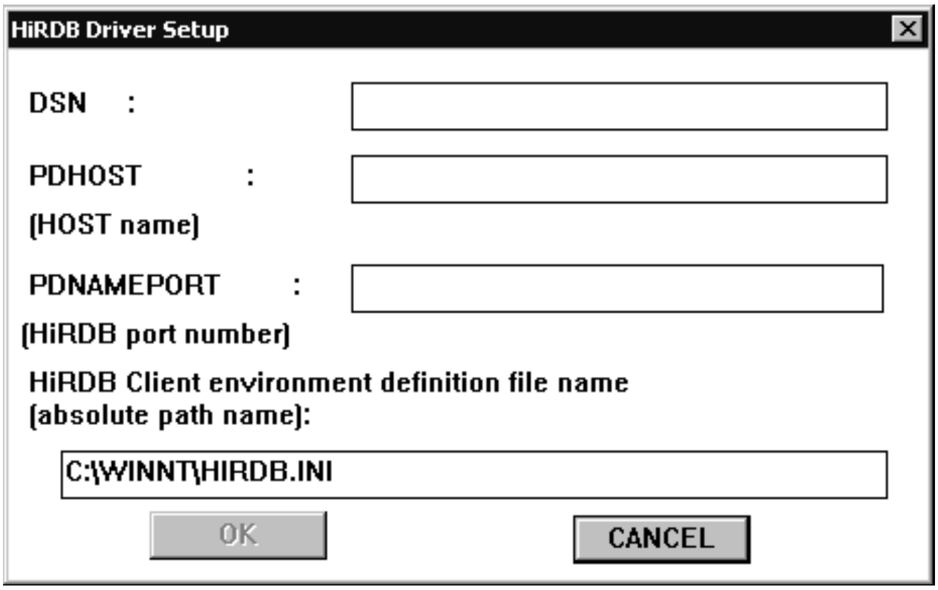
5. Existing data sources are displayed. If no data source has been defined, nothing is displayed. Choose the **Add** button.



6. Select the HiRDB driver as being subject to data source addition.



7. A dialog box for setting up the data source is displayed.



Explanation

Data source name

Specify a name identifying the data source. The name can have up to 32 single-byte characters or 16 double-byte characters. Single-byte and double-byte characters can also be mixed.

PDHOST (host name)

Specify the host name of the server machine. This is the name specified in the client environment definition. For details about `PDHOST`, see *6.6.4 Environment definition information*. If this information is omitted, the system assumes the value in the client environment definition.

PDNAMEPORT (HiRDB port number)

Specify the port number of the server machine. This is the port number specified in the client environment definition. For details about `PDNAMEPORT`, see *6.6.4 Environment definition information*. If this information is omitted, the system assumes the value in the client environment definition.

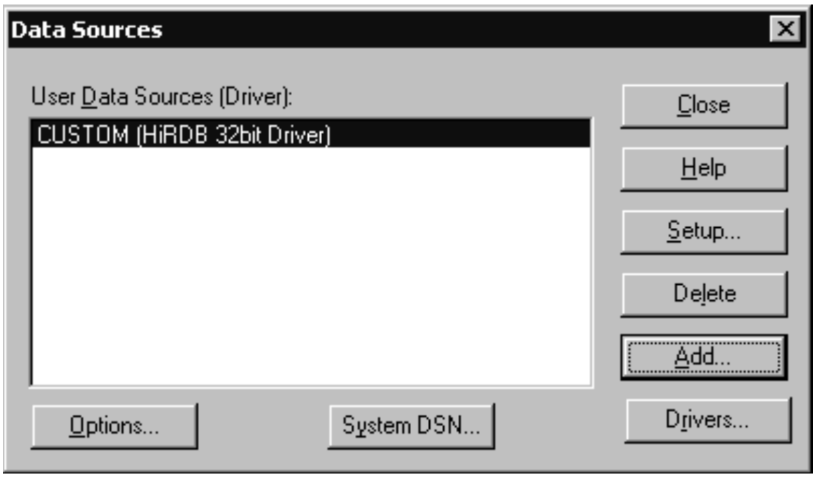
HiRDB client environment definition file name (absolute path name) *

Specify the absolute path name of the HiRDB client environment definition file. Use this item to change the specification values for the HiRDB client environment variables for a particular data source. For example, if you are using the high-speed connection facility (PDSERVICEPORT) to connect to multiple HiRDB systems, you can use this item to specify the file name of the HiRDB client environment definition file and change the connection destination for each data source.

If this information is omitted, the system assumes HIRDB.INI. For all client environment variables, except PDHOST and PDNAMEPORT, the system uses the settings in the HiRDB client environment definition file specified here.

If the specified file is not HIRDB.INI, the system ignores the specifications in HIRDB.INI.

8. After specifying all items, choose the **OK** button. The specified data source is displayed. To change the settings, choose the **Set** button to display the previous dialog box.



* When you install a HiRDB client, a HiRDB client environment definition file is automatically created under the filename HiRDB.INI in the system directory. To install the ODBC driver before installing the HiRDB client, you need to create the HIRDB.INI file, because this file has not been created. To create a client environment definition file, copy the HIRDB.INI file found in the odb32\Disk1\Sampleap directory of the installation CD-ROM to an appropriate directory, and then edit the file. For details about each client environment variable, see *6.6.4 Environment definition information*.

13.3 Installing the ODBC3.0 driver and setting the environment variables

13.3.1 Installation

(1) Installation directory

Table 13-1 shows the ODBC3.0 driver installation directory.

Table 13-1: ODBC3.0 driver installation directory

Platform	Installation directory
Windows 2000	*Windows-directory*\System32
Windows Server 2003	
Windows XP	

Note

The default Windows directories are as follows:

- Windows Server 2003 and Windows XP: `C:\WINDOWS`
- Windows 2000: `C:\WINNT`

(2) Installation flow

The ODBC3.0 driver installation flow is described as follows.

1. Installing the ODBC3.0 driver

   Insert the provided medium and follow the installation procedure.

2. Installing the ODBC driver manager

   If the ODBC driver manager version is old, install the new ODBC driver manager.

3. Setting data sources

   Set data sources.

(3) Installation procedure

(a) Installing the ODBC3.0 driver

To install the ODBC 3.0 driver:

1. Execute `hcd_inst.exe` found on the integrated CD-ROM to start Hitachi Integrated Installer.
2. At the Hitachi Integrated Installer screen, select one of the following, and then

click the **Execute Installation** button to start the HiRDB setup program:

For the UNIX version:

- For HiRDB/Run Time: **HiRDB/Run Time**
- For HiRDB/Developer's Kit: **HiRDB/Developer's Kit**

For the Windows version:

- For a HiRDB/Single Server: **HiRDB/Single Server**
- For a HiRDB/Parallel Server: **HiRDB/Parallel Server**

3. Perform the following operation; the setup program for the selected program process starts:

   For the UNIX version:

   From the Select Program Process window of the HiRDB setup program, select one of the following, and then click the **Next** button:

   - For HiRDB/Run Time: **HiRDB/Run Time**
   - For HiRDB/Developer's Kit: **HiRDB/Developer's Kit**

   For the Windows version:

   From the Select Program Process window of the HiRDB setup program, select **HiRDB/Run Time**, and then click the **Next** button.

4. When the Select Installation Destination dialog box appears, change the installation destination as needed and click the **Next** button.
5. From the Select Setup Type dialog box, select **Custom** and click the **Next** button.
6. From the Select Component dialog box, select ODBC3.0 driver and click the **Next** button.
7. Both the ODBC3.0 driver and MDAC2.6RTM are copied to `C:\Program Files\HITACHI\HiRDB\utl` (for the default).
8. The installation procedure is now complete.

(b) Installing the ODBC driver manager (which is included in MDAC2.6RTM)

If the version of the installed ODBC driver manager is old, you must install MDAC by using the following procedure. To determine the version of the ODBC driver manager, start the ODBC Administrator and double-click the **About the ODBC driver manager** tab. If the driver manager version is 3.520.6526.0 or earlier, it is old.

To install the ODBC driver manager:

1. Double-click `mdac_typ.exe`, which has been copied to the ODBC3.0 driver installation folder `C:\Program Files\HITACHI\HiRDB\utl` (for the

default).

2. Follow the installation procedure displayed on the screen.

(c) Setting data sources

To set data sources:

1. Start the ODBC Data Source Administrator.
2. Make sure that the tab item is **User DSN** and click the **Add** button.
3. When the Add Data Source dialog box appears, select **HiRDB ODBC3.0 Driver** and click the **Finish** button.
4. When the HiRDB ODBC3.0 Driver Setup dialog box appears, specify the necessary items.

Data source name

Specify a name identifying the data source. The name can have up to 32 single-byte characters or 16 double-byte characters. Single-byte and double-byte characters can also be mixed.

PDHOST (host name)

For a HiRDB/Single Server, specify the host name of the server machine on which the single server is located. For a HiRDB/Parallel Server, specify the host name of the server machine on which the system manager is located.

If this item is omitted, the value specified for `PDHOST` in the client environment definition is assumed. For details about `PDHOST`, see *6.6.4 Environment definition information*.

PDNAMEPORT (HiRDB port number)

Specify the port number (the value specified for the `pd_name_port` operand of the system definition) of the HiRDB server to be accessed.

If this item is omitted, the value specified for `PDNAMEPORT` in the client environment definition is assumed. For details about `PDNAMEPORT`, see *6.6.4 Environment definition information*.

HiRDB client environment definition file name

Specify the absolute path name of the HiRDB client environment definition file. Use this item to change the specification values for the HiRDB client environment variables for a particular data source. For example, if you are using the high-speed connection facility (`PDSERVICEPORT`) to connect to multiple HiRDB systems, you can use this item to specify the file name of the HiRDB client environment definition file and change the connection destination for each data source.

If this item is omitted, `HIRDB.INI` is assumed.

5. Choosing the **OK** button returns the window to the **User DSN** tab, and the registered data sources are displayed.

- Stopping data source setup

To stop data source setup, from the HiRDB ODBC3.0 Driver Setup dialog box, click the **Cancel** button. When the **Cancel** button is clicked, no data source is registered.

- Deleting a data source

To delete a data source:

1. From the Data Source dialog box, select the name of the data source to be deleted.
2. Click the **Delete** button to delete the data source.

13.3.2 Setting the environment variables

Set up the following environment variables:

`PATH=`*Windows-directory*`;`*Windows-directory*`\System32`

Note 1

The default Windows directories are as follows:

- Windows 2000: `C:\WINNT`
- Windows XP: `C:\WINDOWS`

Note 2

Set as system environment variables.

13.3.3 Determining the version number of the ODBC3.0 driver

To determine the version number of the ODBC driver, start the ODBC Data Source Administrator and select the **Driver** tab.

13.4 ODBC functions provided by HiRDB

HiRDB provides ODBC functions, and you can access HiRDB on a server from a UAP that utilizes these ODBC functions. Table 13-2 shows the ODBC functions provided by HiRDB.

Table 13-2: ODBC functions provided by HiRDB

Classification	ODBC functions	ODBC2.0 driver		ODBC3.0 driver	
		Provided?	Expansion level	Provided?	Expansion level
Connection to data source	SQLAllocEnv	Y	Core	—	—
	SQLAllocHandle	—	—	Y	Core
	SQLAllocConnect	Y	Core	—	—
	SQLConnect	Y	Core	Y	Core
	SQLDriverConnect	Y	1	Y	Core
	SQLBrousConnect	Y	2	Y	1
Driver and data source information acquisition	SQLDataSources	Y[1]	2	Y[1]	Core
	SQLDrivers	—	—	Y[1]	Core
	SQLGetInfo	Y	1	Y	Core
	SQLGetFunctions	—	—	Y[1]	Core
	SQLGetTypeInfo	Y	1	Y	Core
Driver option setting and acquisition	SQLSetConnectOption	Y	1	—	—
	SQLGetConnectOption	Y	1	—	—
	SQLSetStmtOption	Y	1	—	—
	SQLGetStmtOption	Y	1	—	—
	SQLSetConnectAttr	—	—	Y	Core
	SQLGetConnectAttr	—	—	Y	Core
	SQLSetEnvAttr	—	—	Y	Core
	SQLGetEnvAttr	—	—	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC3.0 driver	
		Provided?	Expansion level	Provided?	Expansion level
	`SQLSetStmtAttr`	—	1	Y	Core
	`SQLGetStmtAttr`	—	1	Y	Core
Descriptor value setup	`SQLGetDescField`	—	—	Y	Core
	`SQLGetDescRec`	—	—	Y	Core
	`SQLSetDescField`	—	—	Y	Core
	`SQLSetDescRec`	—	—	Y	Core
	`SQLCopyDesc`	—	—	Y	Core
SQL request creation	`SQLAllocStmt`	Y	Core	—	—
	`SQLPrepare`	Y	Core	Y	Core
	`SQLBindParameter`	Y	1[1]	Y	Core
	`SQLSetParam`[2]	Y	1	—	—
	`SQLGetCursorName`	Y	Core	Y	Core
	`SQLSetCursorName`	Y	Core	Y	Core
	`SQLDescribeParam`	Y	2	—	—
	`SQLNumParam`	Y	2	—	—
	`SQLParamOptions`	N	2	—	—
	`SQLSetScrollOptions`	N[3]	2	N	2
SQL execution	`SQLExecute`	Y	Core	Y	Core
	`SQLExecDirect`	Y	Core	Y	Core
	`SQLNativeSql`	Y	2	Y	Core
	`SQLDescribeParams`	—	—	Y	2
	`SQLNumParams`	—	—	Y	Core
	`SQLParamData`	Y	1	Y	Core
	`SQLPutData`	Y	1	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC3.0 driver	
		Provided?	Expansion level	Provided?	Expansion level
Execution result and execution result information acquisition	SQLRowCount	Y	Core	Y	Core
	SQLNumResultCols	Y	Core	Y	Core
	SQLDescribeCol	Y	Core	Y	Core
	SQLColAttributes	Y	Core	Y	Core
	SQLBindCol	Y	Core	Y	Core
	SQLFetch	Y	Core	Y	Core
	SQLFetchScroll	—	—	Y[4]	Core
	SQLExtendedFetch	N[3]	2	Y	Core
	SQLGetData	Y	1	Y	Core
	SQLSetPos	N[3]	2	Y[4]	1
	SQLBulkOperations	—	—	N	1
	SQLMoreResults	Y	2	Y	1
	SQLError	Y	Core	—	—
	SQLGetDiagField	—	—	Y	Core
	SQLGetDiagRec	—	—	Y	Core
Data source system information acquisition	SQLColumnPrivileges	Y	2	Y	2
	SQLColumns	Y	1	Y	Core
	SQLForeignKeys	Y	2	Y	2
	SQLPrimaryKeys	Y	2	Y	1
	SQLProcedureColumns	Y	2	Y	1
	SQLProcedure	Y	2	Y	1
	SQLSpecialColumns	Y	1	Y	Core
	SQLStatistics	Y	1	Y	Core
	SQLTablePrivileges	Y	2	Y	2
	SQLTables	Y	1	Y	Core

Classification	ODBC functions	ODBC2.0 driver		ODBC3.0 driver	
		Provided?	Expansion level	Provided?	Expansion level
SQL execution termination	`SQLFreestmt`	Y	Core	Y	Core
	`SQLCloseCursor`	—	—	Y	Core
	`SQLCancel`	Y	Core	Y	Core
	`SQLTransact`	Y	Core	Y	Core
	`SQLEndTran`	—	—	Y	Core
Disconnection	`SQLDisconnect`	Y	Core	Y	Core
	`SQLFreeConnect`	Y	Core	—	—
	`SQLFreeEnv`	Y	Core	—	—
	`SQLFreeHandle`	—	—	Y	Core

Legend:

Y: The applicable ODBC function is provided.

N: The applicable ODBC function is not provided.

— : Not applicable

1: Level 1

2: Level 2

Core: Core level

[1] This function is provided by the drive manager.

[2] Although the `SQLSetParam` function was included in `SQLBindParameter` beginning with ODBC 2.0, this function is provided to maintain compatibility with applications that do not support ODBC 2.0.

[3] Because this function is installed in the ODBC2.0 cursor library, the range of functions specified by the cursor library can be used. To use `SQLExtendedFetch`, set up a cursor library. For details on setting up a cursor library, see *13.7 Setting cursor libraries*.

[4] To use these ODBC functions, you must use the cursor library provided by Microsoft.

13.5 ODBC function data types and HiRDB data types

Table 13-3 shows the correspondence between ODBC function data types and server HiRDB data types.

ODBC function data type refers to an SQL data type that is specified in an argument of an ODBC function.

Table 13-3: ODBC function data types and HiRDB data types

Classification	ODBC data type	HiRDB data type	Description	Availability
Character data	SQL_CHAR	CHAR(*n*)	Fixed-length character string	U
	SQL_VARCHAR	VARCHAR(*n*)	Variable-length character string	U
	SQL_LONGVARCHAR	VARCHAR(*n*)	Variable-length character string	U
	SQL_CHAR	NCHAR(*n*)	Fixed-length national character string NATIONAL CHARACTER(*n*)	U
	SQL_VARCHAR	NVARCHAR(*n*)	Variable-length national character string	U
	SQL_CHAR	MCHAR(*n*)	Fixed-length mixed character string	U
	SQL_VARCHAR	MVARCHAR(*n*)	Variable-length mixed character string	U

Classification	ODBC data type	HiRDB data type	Description	Availability
Numeric data	`SQL_DECIMAL`	`DEC[IMAL]`(*p*,*s*)	Fixed-point number Precision (total number of digits) = *p*, Scale (number of digits below the decimal point) = *s* $1 \leq p \leq 15, 0 \leq s \leq p$	U
	`SQL_NUMERIC`	—		NU
	`SQL_SMALLINT`	`SMALLINT`	Integer from `-32,768` to `32,767`	U
	`SQL_INTEGER`	`INTEGER`	Integer from `-2,147,483,648` to `2,147,483,647`	U
	`SQL_TINYINT`	—	Integer from `-256` to `255`	NU
	`SQL_BIGINT`	—	1-digit sign and 19-digit integer	NU
	`SQL_REAL`	`SMALLFLT,REAL`	Single-precision floating-point number	U
	`SQL_FLOAT`	`FLOAT, DOUBLE PRECISION`	Double-precision floating-point number	U
	`SQL_DOUBLE`	`FLOAT, DOUBLE PRECISION`	Double-precision floating-point number	U
	`SQL_BIT`	—	Bit	NU
	`SQL_BINARY`	—	Fixed-length binary data	NU
	`SQL_LONGVARBINARY`	`BINARY(n)`	Variable-length binary data	U
	`SQL_LONGVARBINARY`	`BLOB`	Variable-length binary data	U
Date and time data	`SQL_TYPE_DATE`	`DATE`	Date	U
	`SQL_TYPE_TIMESTAMP`	`TIMESTAMP`	Date/time	U
	`SQL_TYPE_TIME`	`TIME`	Time	U
	—*	`INTERVAL YEAR TO DAY`	Date interval	NU
	`SQL_INTERVAL_HOUR_TO_SECOND`	`INTERVAL HOUR TO SECOND`	Time interval	U

Classification	ODBC data type	HiRDB data type	Description	Availability
User-defined type	—	Abstract data type	Abstract data type	NU

—: Data type not available in ODBC.

U: Can be used.

NU: Cannot be used.

* Database data types in the server are reported without change.

Note

For details about the maximum character string lengths and value ranges for the various data types, see the manual *HiRDB Version 8 SQL Reference*.

(1) Facilities available to ODBC functions

When a UAP uses ODBC functions to access the HiRDB system in the server, not all HiRDB facilities are available to the UAP. Table 13-4 lists the facilities that can be used by such a UAP.

Table 13-4: Available facilities

Facility	Availability
Obtaining special column information	—
Obtaining index information	U
Using date and time data types	U[1]
Using repetition columns	NU[3]
Using array columns	—
Obtaining table and column headers	—
Asynchronous processing	NU
Using the escape character for the `LIKE` predicate	U
Obtaining an updated row count	U
Setting the timeout value for logging in	NU
Using Japanese data types	U[2]
Executing definition SQL statements	U

U: Can be used.

NU: Cannot be used.

—: Not a DBMS function

[1] The INTERVAL YEAR TO DAY data type cannot be used.

[2] The database data types are reported without change.

[3] A repetition column can be accessed if it has a simple structure without repeated ? parameters.

Example

Column C1 of table T1 is a repetition column.

```
SELECT C1[1],C1[2] FROM T1              A
SELECT C1 FROM T1                       —
INSERT INTO T1 VALUES(ARRAY[?,?])       A
INSERT INTO T1 VALUES(?)                —
```

A: Can be accessed

—: Cannot be accessed

(2) Setting update and deletion operations that use cursors

The SQLGetCursorName function obtains the user cursor name that was set with the SQLSetCursorName function. If no cursor name has been set, the SQLGetCursorName function cannot obtain a system-defined cursor. Therefore, set an appropriate user cursor name to update or delete an item with a cursor.

(3) Setting driver options

The options that can be set with the SQLSetConnectOption or SQLGetConnectOption function are limited. Table 13-5 shows the options that can be set.

Table 13-5: Options that can be set with the SQLSetConnectOption and SQLGetConnectOption functions

Option	Setting
SQL_ACCESS_MODE	SQL_MODE_READ_WRITE
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF or SQL_AUTOCOMMIT_ON
SQL_LOGIN_TIMEOUT	—
SQL_TRANSLATE_DLL	—
SQL_TRANSLATE_OPTION	—

Option	Setting
SQL_TXN_ISOLATION	—

— : Cannot be set

13.6 Asynchronous execution of ODBC functions

(1) About asynchronous execution of ODBC functions

When an ODBC application program accesses HiRDB, the program can execute the ODBC functions asynchronously.

When ODBC functions are executed simultaneously, the ODBC driver does not return control to the application until function calling ends. However, when ODBC functions are executed asynchronously, the ODBC driver can return control to the application program at any time. The application program can therefore execute other processes when the ODBC functions are being executed asynchronously.

The following ODBC functions can be executed asynchronously:

- `SQLColumnPrivileges`
- `SQLColumns`
- `SQLExecute`
- `SQLExecDirect`
- `SQLParamData`
- `SQLProcedureColumns`
- `SQLFetch`
- `SQLStatistics`
- `SQLTablePrivileges`
- `SQLTables`
- `SQLProcedures`

(2) Procedure for asynchronous execution of ODBC functions

To execute asynchronous ODBC functions:

1. To enable asynchronous execution in a specific `hstmt` (statement handle) only, use the `SQL_ASYNC_ENABLE` option to call `SQLSetStmtOption`.[1] To enable asynchronous execution in all `hstmt` handles related to `hdbc` (connection handle), use the `SQL_ASYNC_ENABLE` option to call `SQLSetConnectOption`.[2]

2. When an ODBC function that can be executed asynchronously[1] is called with an `hstmt` for which asynchronous execution has been enabled, the ODBC driver starts asynchronous execution of that function and returns `SQL_STILL_EXECUTING`. (If asynchronous execution is not set or if an error occurs, the ODBC driver returns a synchronous execution code, such as

SQL_SUCCESS or SQL_ERROR.)

3. The application program can execute another process while an ODBC function is being executed asynchronously. An application program can call only the SQLAllocStmt, SQLCancel, and SQLGetFunctions with the hstmt that is executing the function asynchronously. If any other function is called (except the function being executed asynchronously), the driver manager returns a sequence error.

4. The application program calls the ODBC function that was being executed asynchronously to check whether execution of that function terminated. If the function is still executing, SQL_STILL_EXECUTING is returned. If the process has terminated, a return code such as SQL_SUCCESS or SQL_ERROR is returned.

   When an application program calls a function to check the execution status, all specified arguments, except hstmt, are ignored. (However, the specified argument values must be effective; otherwise, an error can occur if an incorrect address or value is specified.) For example, if SQLExecDirect is executed asynchronously with the INSERT statement function, and SQLExecDirect is called again, the execution status of the INSERT statement is returned, even if the UPDATE statement is specified.

Note

To disable asynchronous execution in a specific hstmt only, use the SQL_ASYNC_ENABLE option to call SQLSetStmtOption. To disable asynchronous execution in all hstmt handles related to hdbc, use the SQL_ASYNC_ENABLE option to call SQLSetConnectOption.

[1] The settings for SQLSetStmtOption are shown as follows.

Option	Setting
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF or SQL_ASYNC_ENABLE_ON
SQL_BIND_TYPE	Cannot be set.
SQL_MAX_LENGTH	Limit specified by server or value specified by user
SQL_NOSCAN (Default=FALSE)	SQL_NOSCAN_OFF or SQL_NOSCAN_ON
SQL_QUERRY_TIMEOUT	Cannot be set.
SQL_MAX_ROWS	Limit specified by server or value specified by user

[2] The settings for SQLSetConnectOption are shown as follows.

Option	Setting
SQL_ACCESS_MODE	Fixed to SQL_MODE_READ_WRITE

Option	Setting
SQL_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF or SQL_AUTOCOMMIT_ON
SQL_LOGON_TIMEOUT	Cannot be set.
SQL_OPT_TRACE	Fixed to 0 (Off). This option is returned from the ODBC driver manager.
SQL_OPT_TRACEFILE	Fixed to NULL. This option is returned from the ODBC driver manager.
SQL_TRANSLATE_DLL	Cannot be set.
SQL_TRANSLATE_OPTION	Cannot be set.
SQL_TXN_ISOLATION	SQL_TXN_READ_UNCOMMITED
SQL_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF or SQL_ASYNC_ENABLE_ON

(3) Cancelling asynchronous execution for an ODBC function

(a) Cancelling asynchronous execution of an ODBC function

To cancel an ODBC function during asynchronous execution, call SQLCancel.

SQLCancel issues a process cancellation request to the server as soon as it confirms that the specified hstmt is currently undergoing asynchronous execution.

The return value for SQLCancel only reports whether the cancel request was completed. To find out whether asynchronous execution of the function was actually cancelled, call the function that was being processed asynchronously and check the return value. If the function is still executing, SQL_STILL_EXECUTING is returned. If cancel processing was completed, SQL_ERROR and SQLSTATE S1008 (process cancellation) are returned. If the function has already terminated normally, or if an error occurred, a code such as SQL_SUCCESS or SQL_ERROR is returned.

(b) Cancelling asynchronous execution in a multi-thread application program

A multi-thread application program can cancel an ODBC function that is being executed asynchronously with hstmt. To cancel the function, the application program calls SQLCancel from a different thread and uses the same hstmt as that used by the function being cancelled.

The return value of SQLCancel indicates whether the driver received the request correctly. The return values of the original function are SQL_SUCCESS, or SQL_ERROR and SQLSTATE S1008 (process cancellation).

Note

The HiRDB cancel process is executed for an individual connection, and the connection with the server is forcibly disconnected. (The server outputs `KFPS00993-I: Server process termination REQUEST=clt_attention`). Consequently, all statements of the `hstmt` handlers related to the specified `hstmt` are cancelled (the transaction is rolled back). Carefully consider any data being updated before cancelling an ODBC function that is being executed asynchronously.

(4) Coding example

The following is an example of coding for asynchronous execution:

```
SQLSetStmtOption(hstmst, SQL_ASYNC_ENABLE,
SQL_ASYNC_ENABLE_ON);
          ...
 Retrieval processing with  SQLFetch
rc=SQLFetch(hstmt);
while(rc==SQL_STILL_EXECUTING)
{
          ...
 Continue processing of UAP being executed asynchronously
          ...
  if(process cancel request was issued)
  {
        rc=SQL_Cancel(hstmt);
        if(rc==SQL_ERROR){  To error processing for cancel
                            request failure  }
  }
 rc=SQLFetch(hstmt);
}
if(rc == SQL_ERROR){  To error processing  }
 To retrieval data manipulation processing
        ...
```

13.7 Setting cursor libraries

A cursor library must be set before SQLExtendedFetch can be used in an ODBC UAP. A cursor library can be set in two ways:

When the SetConnectOption ODBC function is used:

Use the SetConnectOption ODBC function and specify SQL_ODBC_CURSORS in the fOption argument and SQL_CUR_USE_ODBC in the vParam argument.

When RDO of Visual Basic is used:

Specify rdUseOdbc in the CursorDriver property of the rdoEnvironment object. The following is an example of coding when RDO of Visual Basic is used:

```
Dim mrdoEnv as rdoEnvironment
Set mrdoEnv = rdoEngine.rdoCreateEnvironment("","","")
    mrdoEnv.CursorDriver = rdUseOdbc
    Src = "DSN=host1; UID=USER_A;PWD=USER_A"
    Set mrdoConn = mrdoEnv.OpenConnection
                   ("", rdDriverComplete, False, Src)
        ...
        ...
```

See the simple sample UAPs found in the Sampleap directory of the installation floppy disk for the ODBC driver.

13.8 File DSNs

When an application program uses a file DSN, it can connect to a data source without obtaining information from `ODBC.INI` or the registry because the DSN file stores information for connecting to the data source.

By sharing the file, multiple users can connect to the HiRDB system without having to register the data source (formerly the machine data source) to each machine. A file DSN can be used when the ODBC component version is 3.0 or higher.

File DSNs can be created by the ODBC data source administrator.

Creating file DSNs

To create a file DSN, select a file DSN, add the file DSN, select a driver (HiRDB 32-bit driver), and then specify the storage file name. A connection request is then issued to the HiRDB system, and the driver manager creates the file based on the complete connection character string returned by `SQLDriverConnect`. However, in this case, the password is not stored in the file DSN. If the password is to be shared, add the line `PWD=`*password* to the created file.

13.9 Executing a UAP in Unicode

This section explains the ODBC functions that can be used by a UAP in Unicode.

(1) ODBC functions that can be used by a UAP in Unicode

Table 13-6 shows the ODBC functions that can be used by a UAP in Unicode.

Table 13-6: ODBC functions that can be used by a UAP in Unicode

Classification	Function name	Function
Connection with data source	`SQLConnectW`	Connects to a specific driver based on the data source name, authorization identifier, and password.
	`SQLDriverConnectW`	Connects to a specific driver based on the connection character string. Also, requests to the driver manager and driver that a connection dialog box be displayed for the user.
	`SQLBrowseConnectW`	Returns the continuous level connection attributes and valid attribute values. If a value is specified for each connection attribute, connects to the data source.
	`SQLDriversW`	Returns the installed driver and a list of its attributes.
Driver and data source information	`SQLDataSources`	Returns a list of data sources that can be used.
	`SQLGetInfoW`	Returns a specific driver and data source information.
Setting and acquisition of driver options	`SQLSetConnectAttrW`	Sets the connection attributes.
	`SQLGetConnectAttrW`	Returns the connection attribute values.
	`SQLSetStmtAttrW`	Sets the statement attribute.
	`SQLGetStmtAttrW`	Returns the statement attribute value.
Descriptor setting and acquisition	`SQLSetDescFieldW`	Sets one descriptor field.
	`SQLGetDescFieldW`	Returns one descriptor field value.
	`SQLSetDescRecW`	Sets multiple descriptor fields.

Classification	Function name	Function
	`SQLGetDescRecW`	Returns multiple descriptor field values.
	`SQLPrepareW`	Prepares an SQL statement to be executed later.
SQL request creation	`SQLSetCursorNameW`	Specifies a cursor name.
	`SQLGetCursorNameW`	Returns the cursor name related to the statement handle.
SQL execution	`SQLExecDirectW`	Executes a statement.
	`SQLNativeSqlW`	Returns the text of the SQL statement that the driver converted.
Acquisition of execution results and execution results information	`SQLDescribeColW`	Describes the results set columns.
	`SQLColAttributeW`	Describes the attributes of the results set columns.
	`SQLGetDiagFieldW`	Returns additional diagnosis information (one field of the diagnosis data structure).
	`SQLGetDiagRecW`	Returns additional diagnosis information (multiple fields of the diagnosis data structure).
	`SQLColumnPrivilegesW`	Returns a list of columns and privileges related to one or more tables.
Acquisition of data source system information	`SQLColumnsW`	Returns a list of column names of specified tables.
	`SQLForeignKeysW`	Returns a list of column names that compose an external key when there is an external key in a specified table.
	`SQLPrimaryKeysW`	Returns a list of column names that compose a main key of a specified table.
	`SQLProcedureColumnsW`	Returns a list of input or output parameters and columns that compose the results set of a specified procedure.
	`SQLProceduresW`	Returns a list of procedure names in a specified data source.

Classification	Function name	Function
	`SQLSpecialColumnsW`	Returns the optimum column for identifing lines in a specified table or the column information that is corrected automatically when line values are changed by a transaction.
	`SQLStatisticsW`	Returns statistical information related to a single table and a list of indexes related to the table.
	`SQLTablePrivilegesW`	Returns a list of tables and the privileges related to each table.
	`SQLTablesW`	Returns a list of table names in the specified data source.

(2) Notes

The following notes apply when `UCS2_UJIS` or `UCS2_UTF8` is set in `PDCLTCNVMODE` of the client environment definition:

- The SQL data type returned when the column attribute is acquired is as follows:

  When the HiRDB data type is `CHAR`, `MCHAR`, or `NCHAR`: `SQL_WCHAR`

  When the HiRDB data type is `VARCHAR`, `MVARCHAR`, or `NVARCHAR`: `SQL_WVARCHAR`

- When the column attribute is acquired, if the HiRDB data type is character string system data type, the column definition length x 2 is set for the column length. For example, in case of `char(10), 20` is returned for the column length.

13.10 Tuning and troubleshooting

This section explains how to tune and troubleshoot ODBC UAPs.

(1) Poor performance in a UAP that retrieves multiple rows

Use the block transfer facility. To use this facility, specify the `PDBLKF` operand in the client environment definition. A specification value between 40 and 50 is recommended. Specifying a larger value has little effect in reducing the number of communications and instead may delay processing because of the increased processing overhead. For details about the block transfer facility, see *4.7 Block transfer facility*.

(2) If a UAP executes connect and disconnect processing frequently

Use the high-speed connection facility. To use this facility, specify the `PDFESHOST`, `PDSERVICEPORT`, and `PDSERVICEGRP` operands in the client environment definition. The high-speed connection facility shortens the time for connection to HiRDB. For details about the `PDFESHOST`, `PDSERVICEPORT`, and `PDSERVICEGRP` operands of the client environment definition, see *6.6.4 Environment definition information*.

(3) Checking SQL statements requested of HiRDB

If a UAP accesses the HiRDB system via ODBC, the SQL statements specified in the UAP may differ from the SQL statements requested of the HiRDB system, depending on the environment in which the UAP was created. To check what kind of SQL statements are issued to the HiRDB system, use the SQL trace facility. To use this facility, specify the `PDSQLTRACE` operand in the client environment definition. It is recommended to also specify the trace output destination directory in the `PDCLTPATH` operand at this time. For details about the SQL trace facility, see *10.1.1 SQL tracing*.

(4) Other

- If an application program, such as Microsoft Access, specifies the lock option in a retrieval SQL statement, a syntax error may occur in that application program. If this happens, examine whether the problem can be corrected by specifying the `PDISLLVL` operand of the client environment definition.
- If you use the Microsoft Jet database engine to access HiRDB, a lock error may occur in HiRDB during updating, depending on how the UAP was created. This occurs when the Microsoft Jet database engine establishes multiple connections to HiRDB and referencing or updating is executed on the same line from different connections. To avoid this, specify `0` or `1` for the `PDISLLVL` operand in the client environment definition. In the `Sampleap` directory on the installation floppy disk for the ODBC driver, there is a sample UAP that uses DAO (Data Access Object) of Visual Basic and no lock error occurs during access; refer to this UAP.

13.11 Facilities that cannot be used when HiRDB is accessed with ODBC

When an application program accesses the HiRDB system with ODBC, some of the facilities cannot be used.

- Access using the row interface

  Queries with `ROW` specifications, `UPDATE` statements, and `INSERT` statements cannot be executed.

- Update and deletion using a cursor

  Update and deletion using `CURRENT OF` *cursor-name* cannot be executed. However, if the cursor library facility is used, the cursor library can sometimes execute such operations to change `CURRENT OF` *cursor-name* to a `WHERE` condition.

- Portable cursors

  Portable cursors (cursors with the `WITH HOLD` specification or cursors defined by queries with the `UNTIL DISCONNECT` specification) cannot be used.

Chapter

14. HiRDB Access from OLE DB Application Programs

This chapter provides an overview of the OLE DB and discusses its connection interface, schema information, and error handling procedures.

This chapter contains the following sections:

14.1 Overview

(1) What is OLE DB?

OLE DB is an API, like ODBC, for accessing a wide range of data sources. Unlike ODBC, OLE DB contains interface definitions suitable for accessing data other than SQL data.

(2) HiRDB OLE DB Provider

To access HiRDB from an OLE DB-supported application program, you need a HiRDB OLE DB provider. The HiRDB OLE DB provider is included in HiRDB/Run Time and HiRDB/Developer's Kit.

(3) Installing the HiRDB OLE DB provider

To install the HiRDB OLE DB provider when installing HiRDB/Run Time or HiRDB/Developer's Kit, in the Setup Type dialog box, choose **Custom**, and in the Select Components dialog box, select **OLE DB provider for HiRDB**.

When you install the HiRDB OLE DB provider, the following files are created:

- `PDOLEDB.DLL`
- `PDCLTL32.DLL`

(4) HiRDB OLE DB provider name

The name of the HiRDB OLE DB provider (provider program ID) is `HiRDBProvider`. When using an interface that requires the provider name (such as ActiveX Data Object (ADO)), you can use the HiRDB OLE DB provider by specifying this provider name in the `connection` object's `Provider` property.

14.2 Connection interface

This section explains the registry information and connection property.

14.2.1 Registry information

(1) Adding to the HKEY_CLASSES_ROOT key

(a) Provider program ID = provider name

```
"HiRDBProvider"="Hitachi HiRDB OLE DB Provider"
```

(b) Provider class ID

```
"HiRDBProvider\\ClSID"
  ="{6A708561-748A-11d3-B810-0000E2212E58}"
```

(2) Adding to the HKEY_CLASSES_ROOT\CLSID subkey

(a) Provider program ID

```
{"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}"
  ="HiRDBProvider"
```

(b) Provider name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}\\ProgID"
  ="HiRDBProvider"
```

(c) Program ID by version

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\VersionIndependentProgID"="HiRDBProvider"
```

(d) Provider DLL name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\InprocServer32"="pdoledb.dll"
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\InprocServer32\\ThreadingModel"="Both"
```

(e) Comment

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\OLE DB Provider"="Hitachi HiRDB OLE DB Provider"
```

(f) Extended error name

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\ExtendedErrors"="Hitachi HiRDB OLE DB Provider"
```

(g) Extended error comment

```
"CLSID\\{6A708561-748A-11d3-B810-0000E2212E58}
  \\ExtendedErrors\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  = "Hitachi HiRDB OLE DB Provider"
```

(3) Adding to the HKEY_CLASSES_ROOT key

(a) Provider error program ID

```
" HiRDBProviderErrors"="Hitachi HiRDB OLE DB Provider"
```

(b) Provider error class ID

```
"HiRDBProviderErrors\\ClSID"
  ="{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
```

(4) Adding to the HKEY_CLASSES_ROOT\CLSID subkey

(a) Provider error program ID

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}"
  ="HiRDBProvider Error Lookup"
```

(b) Provider error lookup name

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}\\ProgID"
  ="HiRDBProvider Error Lookup"
```

(c) Error lookup program ID by version

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\VersionIndependentProgID"="HiRDBProvider Error Lookup"
```

(d) Provider error lookup DLL name

```
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\InprocServer32"="pdoledb.dll"
"CLSID\\{5F6D492E-40BA-11D3-BD66-0000E21F878E}
  \\InprocServer32\\ThreadingModel"="Both"
```

14.2.2 Connection properties

Three `Initialization` properties are used for connection. These three properties are optional.

(1) DBPROP_INIT_DATASOURCE

This is the client's environment variable group name. If this property is omitted, the system assumes `HiRDB.INI`. For details about the client's environment variable group, see *6.7 Registering an environment variable group*.

(2) DBPROP_AUTH_USERID

This is the authorization identifier used for connection.

If this property is omitted, the authorization identifier is acquired from `PDUSER` of the applicable client environment variables group. If there is no `DBPROP_INIT_DATASOURCE` specification, the authorization identifier is acquired from `HiRDB.INI`.

(3) DBPROP_AUTH_PASSWORD

This is the password to be used for connection. If this property is omitted, but DBPROP_INIT_DATASOURCE is specified, the system obtains the password from PDUSER in the corresponding client environment variable group. If DBPROP_INIT_DATASOURCE is also omitted, the system obtains the password from HiRDB.INI.

14.3 Schema information

Table 14-1 lists the schema information provided by the HiRDB OLE DB provider.

Table 14-1: Schema information provided by the HiRDB OLE DB provider

Type of OLE DB schema information	Description	Provided
ASSERTIONS	Assertion information	--
CATALOGS	Catalog information	--
CHARACTER_SETS	Character set identification	--
CHECK_CONSTRAINTS	CHECK constraint identification	--
COLLATIONS	Character collation identification	--
COLUMN_DOMAIN_USAGE	Domain-dependent column information	--
COLUMN_PRIVILEGES	Column privilege information	--
COLUMNS	Column information	P (required)
CONSTRAINT_COLUMN_USAGE	Various constraint (reference, UNIQUE, CHECK) column information	--
CONSTRAINT_TABLE_USAGE	Various constraint (reference, UNIQUE, CHECK) table information	--
FOREIGN_KEYS	External key information	--
INDEXES	Index information	P
KEY_COLUMN_USAGE	Key column information	--
PRIMARY_KEYS	Primary key information	--
PROCEDURE_COLUMNS	Column information for row set returned by procedure	--
PROCEDURE_PARAMETERS	Procedure parameter information	P
PROCEDURES	Procedure information	P
PROVIDER_TYPES	Provider data type identification	P (required)
REFERENTIAL_CONSTRAINTS	Reference constraints	--
SCHEMATA	Schema information	P

Type of OLE DB schema information	Description	Provided
SQL_LANGUAGES	Match level for processing SQL installation and language type	—
STATISTICS	Statistical information	—
TABLE_CONSTRAINTS	Table constraints	—
TABLE_PRIVILEGES	Table privilege information	P
TABLES	Table information	P (required)
TRANSLATIONS	Character conversion identification	—
USAGE_PRIVILEGES	User privilege information	—
VIEW_COLUMN_USAGE	View column information	—
VIEWS	View information	—

P: Provided.

—: Not provided.

14.4 Data type correspondences

Table 14-2 shows the correspondences between the HiRDB data types and the OLE DB type indicators.

Table 14-2: Correspondences between the HiRDB data types and the OLE DB type indicators

HiRDB data types	OLE DB type indicators
CHAR, MCHAR, and NCHAR	DBTYPE_STR
VARCHAR, MVARCHAR, and NVARCHAR	
DECIMAL(*p*,*s*)	DBTYPE_NUMERIC
SMALLINT (signed)	DBTYPE_I2
INTEGER (signed)	DBTYPE_I4
REAL	DBTYPE_R4
SMALLFLT	
FLOAT	DBTYPE_R8
DOUBLE PRECISION	
BLOB	DBTYPE_BYTES
BINARY	
DATE	DBTYPE_DBDATE
TIME	DBTYPE_DBTIME
TIMESTAMP	DBTYPE_DBTIMESTAMP
INTERVAL YEAR TO DAY	DBTYPE_DECIMAL
INTERVAL HOUR TO SECOND	DBTYPE_DECIMAL

14.5 Error handling procedures

14.5.1 Troubleshooting facility

This facility collects trace information about the OLE DB interface (for each method) issued by consumers.

(1) Collection method

Specify appropriate values in the following registry keys:
`HKEY_LOCAL_MACHINE`

Trace information is collected only when the value of `Software\HITACHI\HiRDB\oleprovtrc` is `1`.

Specify the absolute path of the output file name in `Software\HITACHI\HiRDB\oletrcfile`. (If `oletrcfile` is omitted, the system outputs trace information to `c:\temp\pdoletrc.txt`.)

Trace information is output to `Software\HITACHI\HiRDB\oletrcdumpsize` with `GetData()` and input with `Execute()`. Specify the `void*` type data dump output size in bytes. (If `oletrcdumpsize` is omitted, the system assumes `256`.)

14.6 Notes

(1) About a cursor in ADO

HiRDB does not allow you to use the server cursor (specify `adUseServer` in the `CursorLocation` property of the `Recordset` object) in ADO. To use a cursor in ADO, use the client cursor (specify `adUseClient` in the `CursorLocation` property of the `Recordset` object).

Chapter

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

This chapter describes the installation and functions of HiRDB.NET Data Provider, which is required to access HiRDB from ADO.NET-compatible application programs.

15.1 Overview

15.1.1 HiRDB.NET Data Provider

.NET Framework provides a common-language runtime that does not depend on the platform or development language being used. It also provides the .NET Framework class libraries. ADO.NET is a library that can be used when .NET Framework applications that access databases are created.

HiRDB provides HiRDB.NET Data Provider, which is required to access HiRDB using ADO.NET. HiRDB.NET Data Provider complies with ADO.NET specifications.

HiRDB.NET Data Provider provides the common basic interface group that is provided in .NET Framework's System.Data address space. It also provides the `INSERT` facility using arrays and accesses to repetition columns as unique extended functions.

15.1.2 Prerequisite programs for HiRDB.NET Data Provider

(1) Supported platforms

- Windows 2000
- Windows XP
- Windows Server 2003

(2) Required programs

In the application program execution environment, the following programs are required:

- Microsoft Internet Explorer 5.01 or later
- .NET Framework version 1.1 package that can be re-distributed (can be installed by Windows Update)

When application programs are developed, the following programs are also required:

- Microsoft Visual Studio .NET 2003

  Operation with Microsoft Visual Studio .NET 2002 + .NET Framework SDK version 1.1 cannot be guaranteed.

15.2 Installing HiRDB.NET Data Provider

15.2.1 Installation procedure

To install HiRDB.NET Data Provider, during installation of HiRDB/Run Time or HiRDB/Developer's Kit, in the Setup Method window, choose **Custom**, and then in the Select Component window, select **HiRDB Data Provider**.

15.2.2 Files that are installed

When HiRDB.NET Data Provider is installed, the following files are created:

- `pddndp.dll`
- `pddndpcore.dll`

15.2.3 Checking the version information

You can check the version information of HiRDB.NET Data Provider by displaying the DLL properties provided by HiRDB.NET Data Provider.

15.3 List of classes provided by HiRDB.NET Data Provider

HiRDB.NET Data Provider complies with ADO.NET specifications.

Table 15-1 lists and describes the classes provided by HiRDB.NET Data Provider.

Table 15-1: List of HiRDB.NET Data Provider classes

Class	Function
HiRDBCommand	Represents an SQL statement or stored procedure that is executed on a database.
HiRDBCommandBuilder	Automatically creates a single table command to make a change to DataSet that has been associated with a database.
HiRDBConnection	Represents an open connection to a database.
HiRDBDataAdapter	Represents a series of data commands and database connections that are used to store data in DataSet and update a database.
HiRDBDataReader	Provides a method for reading a forward stream in data rows from a database.
HiRDBException	Represents an exception that is created when a warning or error is returned from HiRDB.NET Data Provider.
HiRDBParameter	Represents a HiRDBCommand parameter and a map for DataColumn as an option.
HiRDBParameterCollection	Represents a parameter collection associated with HiRDBCommand and a map of each parameter for DataSet columns.
HiRDBRowUpdatedEventArgs	Provides data for a RowUpdated event.
HiRDBRowUpdatingEventArgs	Provides data for a RowUpdating event.
HiRDBTransaction	Represents a transaction that is executed on a database.

15.4 List of members provided by HiRDB.NET Data Provider

This section presents a list of interface members provided by HiRDB.NET Data Provider.

15.4.1 List of HiRDBCommand members

(1) Constructor

`HiRDBCommand`

(2) Inheritance classes

`Component`, `IDbCommand`, `ICloneable`

(3) Properties

Member	Function
`CommandText`	Acquires or sets the text command that is executed on a database.
`CommandTimeout`	Acquires or sets the wait time before command retries are cancelled and an error is generated.
`CommandType`	Acquires or sets a value that indicates how to interpret the `CommandText` property.
`Connection`	Acquires or sets the `HiRDBConnection` that is used by this `HiRDBCommand`.
`Parameters`	Acquires `HiRDBParameterCollection`.
`Transaction`	Acquires or sets the `HiRDBTransaction` on which this `HiRDBCommand` is executed.
`UpdatedRowSource`	Acquires or sets how to apply the command result to `DataRow` when `HiRDBDataAdapter`'s `Update` method uses the command result.

(4) Methods

Member	Function
`Cancel`	Cancels execution of `HiRDBCommand`.
`Clone`	Creates a new object which is a copy of the current instance.
`CreateParameter`	Creates a new instance of the `HiRDBParameter` object.
`ExecuteNonQuery`	Executes an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.
`ExecuteReader`	Executes `CommandText` on `HiRDBConnection` and creates `HiRDBDataReader` using one of the `CommandBehavior` values.

Member	Function
`ExecuteScalar`	Executes a query and returns the first column of the first row in the result set returned by the query. Any excess column or row will be ignored.
`Prepare`	Creates a prepared version of a command (compiled) in a database.

15.4.2 List of HiRDBCommandBuilder members

(1) Constructor

`HiRDBCommandBuilder`

(2) Inheritance class

`Component`

(3) Property

Member	Function
`DataAdapter`	Acquires or sets the `HiRDBDataAdapter` object for which an SQL statement is to be created automatically.

(4) Methods

Member	Function
`GetDeleteCommand`	Acquires the automatically created `HiRDBCommand` object for executing deletion processing on the database.
`GetInsertCommand`	Acquires the automatically created `HiRDBCommand` object for executing insertion processing on the database.
`GetUpdateCommand`	Acquires the automatically created `HiRDBCommand` object for executing update processing on the database.
`RefreshSchema`	Updates database schema information to create the `INSERT`, `UPDATE`, or `DELETE` statement.

15.4.3 List of HiRDBConnection members

(1) Constructor

`HiRDBConnection`

(2) Inheritance classes

`Component`, `IDbConnection`, `ICloneable`

(3) Properties

Member	Function
`ConnectionString`	Acquires or sets the character string that is used to open a database.
`ConnectionTimeout`	Acquires the wait time for establishing a connection before retries are cancelled and an error is generated.
`Database`	Acquires the name of the current database or the database that is used when a connection is established.
`LifeTime`	Acquires or sets the time remaining before actual disconnection occurs.
`Pooling`	Acquires or sets whether or not pooling is to be performed.
`State`	Acquires the current connection status.

(4) Methods

Member	Function
`BeginTransaction`	Starts the database transaction using the specified `IsolationLevel` value.
`ChangeDatabase`	Changes the current database for the open `HiRDBConnection` object.
`Clone`	Creates a new object which is a copy of the current instance.
`Close`	Closes the connection to the database.
`CreateCommand`	Creates and returns the `HiRDBCommand` object associated with the connection.
`Dispose`	Releases all resources used by `HiRDBConnection`.
`Open`	Opens the database connection with the settings specified in the `ConnectionString` property of the `HiRDBConnection` object.

15.4.4 List of HiRDBDataAdapter members

(1) Constructor

`HiRDBDataAdapter`

(2) Inheritance classes

`DbDataAdapter`, `IDbDataAdapter`

(3) Properties

Member	Function
`DeleteCommand`	Acquires or sets the SQL statement for deleting records from a data set.
`InsertCommand`	Acquires or sets the SQL statement for inserting new records in a database.

Member	Function
`SelectCommand`	Acquires or sets the SQL statement for selecting records in a database.
`UpdateCommand`	Acquires or sets the SQL statement for updating records in a database.

15.4.5 List of HiRDBDataReader members

(1) Constructor

`HiRDBDataReader`

(2) Inheritance classes

`MarshalByRefObject`, `IEnumerable`, `IDataReader`, `IDisposable`, `IDataRecord`

(3) Properties

Member	Function
`Depth`	Acquires the value indicating the nesting level of the current row.
`FieldCount`	Acquires the number of columns in the current row.
`IsClosed`	Acquires the value indicating whether or not the data reader is closed.
`RecordsAffected`	Acquires the number of rows changed, inserted, or deleted by execution of the SQL statement.

(4) Methods

Member	Function
`Close`	Closes the `HiRDBDataReader` object.
`GetBoolean`	Acquires the value of the specified column as a Boolean value.
`GetByte`	Acquires an unsigned 8-bit integer value in the specified column.
`GetBytes`	Reads a byte stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.
`GetChar`	Acquires the character string value in the specified column.
`GetChars`	Reads a character stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.
`GetData`	The purpose of this member is to support the .NET Framework infrastructure. It cannot be used directly in a unique coding that has been created.
`GetDataTypeName`	Acquires data-type information for the specified field.

Member	Function
`GetDateTime`	Acquires or sets the date and time data value in the specified field.
`GetDecimal`	Acquires the fixed position value in the specified field.
`GetDouble`	Acquires the double-precision floating-point number in the specified field.
`GetEnumerator`	Returns the enumerator that can perform iterative operation on a collection.
`GetFieldArrayCount`	Acquires the size of field array.
`GetFieldType`	Acquires `Type` information corresponding to the type of `Object` that is returned from `GetValue`.
`GetFloat`	Acquires the single-precision floating-point number in the specified field.
`GetGuid`	Returns the `GUID` value of the specified field.
`GetInt16`	Acquires a signed 16-bit integer value in the specified field.
`GetInt32`	Acquires a signed 32-bit integer value in the specified field.
`GetInt64`	Acquires a signed 64-bit integer value in the specified field.
`GetName`	Acquires the name of the field to be searched.
`GetOrdinal`	Returns the index of the specified field.
`GetSchemaTable`	Returns the `DataTable` that describes `HiRDBDataReader`'s column metadata.
`GetString`	Acquires a character string in the specified field.
`GetValue`	Returns a value in the specified field.
`GetValues`	Acquires all attribute fields in the current record collection.
`IsDBNull`	Returns a value indicating whether or not the specified field is set to `null`.
`NextResult`	Advances the data reader to the next result when the result of a batch SQL statement is read.
`Read`	Advances `HiRDBDataReader` to the next record.

15.4.6 List of HiRDBException members

(1) Constructor

```
HiRDBException
```

(2) Inheritance class

```
Exception
```

(3) Properties

Member	Function
`ErrorCode`	Acquires the error code part as an `int`.
`Message`	Acquires text with a complete error.

15.4.7 List of HiRDBParameter members

(1) Constructor

`HiRDBParameter`

(2) Inheritance classes

`MarshalByRefObject`, `IDbDataParameter`, `IDataParameter`, `ICloneable`

(3) Properties

Member	Function
`DbType`	Acquires or sets `DbType` for a parameter. When `DbType` is to be set, this member sets the corresponding data type in the `HiRDBType` property according to Table 15-3.
`Direction`	Acquires or sets a value indicating whether the parameter is input only, output only, bidirectional, or the stored procedure's return value.
`HiRDBType`	Acquires or sets an enumeration indicating the data type in HiRDB. When an enumeration is to be set, this member sets the corresponding data type in the `DbType` property according to Table 15-4. `HiRDBType` enumeration: `Integer`, `SmallInt`, `Decimal`, `Float`, `SmallFlt`, `Char`, `VarChar`, `NChar`, `NVarChar`, `MChar`, `MVarChar`, `Date`, `Time`, `TimeStamp`, `IntervalYearToDay`, `IntervalHourToSecond`, `Blob`, `Binary`
`IsNullable`	Acquires a value indicating whether or not the parameter accepts the `null` value.
`ParameterName`	Acquires or sets the name of `HiRDBParameter`.
`Precision`	Acquires or sets the number of significant digits for a numeric parameter.
`Repetition`	Acquires or sets an array structure in HiRDB.
`Scale`	Acquires or sets the number of decimal places for a numeric parameter.
`Size`	Sets the size of the column definition length or character string data that can be stored after charcter code conversion. Also acquires the current setting. For `TIMESTAMP (DateTime)`, this value is the number of digits in the fractional part.
`SourceColumn`	Acquires or sets the name of the source column that has been assigned to `DataSet` and is used to read or return `Value`.
`SourceVersion`	Acquires or sets the `DataRowVersion` that is used to read `Value`.

Member	Function
Value	Acquires or sets a parameter value.

(4) Method

Member	Function
Clone	Creates a new object which is a copy of the current instance.

15.4.8 List of HiRDBParameterCollection members

(1) Constructor

HiRDBParameterCollection

(2) Inheritance classes

MarshalByRefObject, IDataParameterCollection, IList, ICollection, IEnumerable

(3) Properties

Member	Function
Count	Acquires the number of HiRDBParameter objects stored in HiRDBParameterCollection.
IsFixedSize	Acquires a value indicating whether the size of HiRDBParameterCollection is fixed.
IsReadOnly	Acquires a value indicating whether or not HiRDBParameterCollection is read only.
IsSynchronized	Acquires a value indicating whether or not an access to HiRDBParameterCollection is synchronized (thread-safe).
SyncRoot	Acquires an object that can be used to synchronize an access to HiRDBParameterCollection.

(4) Methods

Member	Function
Add	Adds items to HiRDBParameterCollection.
Clear	Deletes all items from HiRDBParameterCollection.
Contains	Acquires a value indicating whether or not HiRDBParameter is in the collection.
CopyTo	Copies the elements of HiRDBParameterCollection to Array using Array's specific index as the start position.
GetEnumerator	Returns an enumerator that can perform iterative operation on a collection.

Member	Function
IndexOf	Acquires the location of HiRDBParameter in a collection.
Insert	Inserts an item at the specified location in HiRDBParameterCollection.
Remove	Deletes the first occurrence of the specified object in HiRDBParameterCollection.
RemoveAt	Deletes HiRDBParameter from the collection.

15.4.9 List of HiRDBRowUpdatedEventArgs members

(1) Constructor

HiRDBRowUpdatedEventArgs

(2) Inheritance class

RowUpdatedEventArgs

(3) Property

Member	Function
Command	Acquires the HiRDBCommand that is executed when Update is called.

15.4.10 List of HiRDBRowUpdatingEventArgs members

(1) Constructor

HiRDBRowUpdatingEventArgs

(2) Inheritance class

RowUpdatingEventArgs

(3) Property

Member	Function
Command	Acquires or sets the HiRDBCommand that is executed during Update processing.

15.4.11 List of HiRDBTransaction members

(1) Constructor

HiRDBTransaction

(2) Inheritance classes

MarshalByRefObject, IDbTransaction, IDisposable

(3) Properties

Member	Function
`Connection`	Acquires the `HiRDBConnection` object used to associate a transaction.
`IsCompleted`	Acquires a value indicating whether or not the transaction is completed.
`IsolationLevel`	Specifies this transaction's `IsolationLevel`.

(4) Methods

Member	Function
`Commit`	Commits a database transaction.
`Rollback`	Rolls back a database transaction from the hold status.

15.5 Interfaces of HiRDB.NET Data Provider

15.5.1 HiRDBCommand

(1) Constructor

(a) HiRDBCommand

`void HiRDBCommand ()`

Description: Initializes a new instance of `HiRDBCommand`.

`void HiRDBCommand (string)`

Argument

`string cmdText`: SQL text (`CommandText` property)

Description: Specifies an SQL text to initialize a new instance of the `HiRDBCommand` class.

`void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection)`

Arguments

`string cmdText`: SQL text (`CommandText` property)

`HiRDBConnection rConnection`: `HiRDBConnection` object representing the connection to the database (`Connection` property)

Description: Uses an SQL text and `HiRDBConnection` object to initialize a new instance of the `HiRDBCommand` class.

`void HiRDBCommand (string, Hitachi.HiRDB.HiRDBConnection, Hitachi.HiRDB.HiRDBTransaction)`

Arguments

`string cmdText`: SQL text (`CommandText` property)

`HiRDBConnection rConnection`: `HiRDBConnection` object representing the connection to the database (`CommandText` property)

`HiRDBTransaction rTransaction`: `HiRDBTransaction` object that executes `HiRDBCommand` (`Transaction` property)

Description: Uses an SQL text and the `HiRDBConnection` and `HiRDBTransaction` objects to initialize a new instance of the `HiRDBCommand` class.

(2) Properties

(a) CommandText

Type: `string`

Default value: `""`

Description: Acquires or sets the text command that is executed on a database.

(b) CommandTimeout

Type: `int`

Default value: `30`

Description: Acquires or sets the wait time before command retries are cancelled and an error is generated.

Exception: `HiRDBException`

(c) CommandType

Type: `System.Data.CommandType`

Default value: `CommandType.Text`

Description: Acquires or sets how to interpret the `CommandText` property.

(d) Connection

Type: `HiRDBConnection`

Default value: `null`

Description: Acquires or sets the `HiRDBConnection` that is used by this `HiRDBCommand`.

Exception: `HiRDBException`

(e) Parameters

Type: `HiRDBParameterCollection`

Description: Acquires `HiRDBParameterCollection` (read only).

(f) Transaction

Type: `HiRDBTransaction`

Default value: `null`

Description: Acquires or sets the `HiRDBTransaction` on which this `HiRDBCommand` is executed.

(g) UpdatedRowSource

Type: `System.Data.UpdateRowSource`

Default value: `UpdatedRowSource.None`

Description: Acquires or sets how to apply the command result to `DataRow` when `HiRDBDataAdapter`'s `Update` method uses the command result.

Exception: `HiRDBException`

(3) Methods

(a) Cancel

`void Cancel ()`

Return: `void`

Description: Cancels execution of `HiRDBCommand`.

(b) Clone

`object Clone ()`

Return

`object`: New object which is a copy of this instance

Description: Creates a new object which is a copy of the current instance.

(c) CreateParameter

`Hitachi.HiRDB.HiRDBParameter CreateParameter ()`

Return

`HiRDBParameter`: `HiRDBParameter` object

Description: Creates a new instance of the `HiRDBParameter` object.

(d) ExecuteNonQuery

`int ExecuteNonQuery ()`

Return

`int`: Number of affected rows

Description: Executes an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.

Exception: `HiRDBException`

`int ExecuteNonQuery (int)`

Argument

`int nArraySize`: Number of array elements

Return

`int`: Number of affected rows

Description: Uses the `INSERT` facility using arrays to execute an SQL statement on the `HiRDBConnection` object and returns the number of affected rows.

Exception: `HiRDBException`

(e) ExecuteReader

`Hitachi.HiRDB.HiRDBDataReader ExecuteReader ()`

Return

`HiRDBDataReader`: `HiRDBDataReader` object

Description: Executes `CommandText` on `HiRDBConnection` to create `HiRDBDataReader`.

Exception: `HiRDBException`

`ExecuteReader (System.Data.CommandBehavior)`

Argument

`System.DataCommandBehavior behavior`: One of the `CommandBehavior` values

Return

`HiRDBDataReader`: `HiRDBDataReader` object

Description: Executes `CommandText` on `HiRDBConnection` and creates `HiRDBDataReader` using one of the `CommandBehavior` values.

Exception: `HiRDBException`

(f) ExecuteScalar

`object ExecuteScalar ()`

Return

`object`: First column of the first row in the result set

Description: Executes a query and returns the first column of the first row in the result set returned as .NET Framework's data type by that query. Any remaining column or row will be ignored.

Exception: `HiRDBException`

(g) Prepare

`void Prepare ()`

Return: `void`

Description: Creates a prepared version of a command (compiled) in a database.

Exception: `HiRDBException`

15.5.2 HiRDBCommandBuilder

(1) Constructor

(a) HiRDBCommandBuilder

`void HiRDBCommandBuilder ()`

Description: Initializes a new instance of `HiRDBCommandBuilder`.

`void HiRDBCommandBuilder (HiRDBDataAdapter adapter)`

Argument

`HiRDBDataAdapter adapter`: `HiRDBDataAdapter` object (`DataAdapter` property)

Description: Specifies the `HiRDBDataAdapter` object and initializes a new instance of `HiRDBCommandBuilder`.

(2) Properties

(a) DataAdapter

Type: `HiRDBDataAdapter`

Default value: `null`

Description: Acquires or sets the `HiRDBDataAdapter` object for which an SQL statement is to be created automatically.

(3) Methods

(a) GetDeleteCommand

`HiRDBCommand GetDeleteCommand (string)`

Argument

`string s TableName`: Table name

Return

`HiRDBCommand`: `HiRDBCommand` object that was automatically created to execute deletion processing

Description: Acquires the automatically created `HiRDBCommand` object for executing deletion processing on the database.

Exception: `HiRDBException`

(b) GetInsertCommand

`HiRDBCommand GetInsertCommand (string)`

Argument

`string s TableName`: Table name

Return

`HiRDBCommand`: `HiRDBCommand` object that was automatically created to execute insertion processing

Description: Acquires the automatically created `HiRDBCommand` object for executing insertion processing on the database.

Exception: `HiRDBException`

(c) GetUpdateCommand

`HiRDBCommand GetUpdateCommand (string)`

Argument

`string s TableName`: Table name

Return

`HiRDBCommand`: `HiRDBCommand` object that was automatically created to execute update processing

Description: Acquires the automatically created `HiRDBCommand` object for executing update processing on the database.

Exception: `HiRDBException`

(d) RefreshSchema

`void RefreshSchema (string)`

Argument

`string s TableName`: Table name

Return: `void`

Description: Updates database schema information to create the `INSERT`, `UPDATE`, or `DELETE` statement.

Exception: `HiRDBException`

15.5.3 HiRDBConnection

(1) Constructor

(a) HiRDBConnection

`void HiRDBConnection ()`

Description: Initializes a new instance of `HiRDBConnection`.

`void HiRDBConnection (string)`

Argument

`string ConnectionString`: Character string storing the connection settings (`ConnectionString` property)

Description: Specifies a connection character string and initializes a new instance of the `HiRDBConnection` class.

(2) Properties

(a) ConnectionString

Type: `string`

Default value: `""`

Description: Acquires or sets the character string that is used to open a database.

Exception: `HiRDBException`

For this property, you must specify one `string`-type argument. The character string to be specified is called a *connection character string*. This is the same type of connection character string as those used for `Connection` in ADO and ADO.NET. The following table lists and describes the character strings that can be specified:

Character string	Description
• `datasource` • `dsn` • `env`	Settings for the registry to be used. Specify the name of the environment variable group that was created using the tool for registering HiRDB client environment variables.
• `uid` • `userid`	Authorization identifier used for DB connection
• `password` • `Pwd`	Password to be used for the database connection
• `PD*`	Settings in the client environment definition

If nothing is specified, the default setting (`HiRDB.ini`) is used to establish the connection. If a client environment variable group name is available, this name is used. If the authorization identifier, password, and client environment definition are specified, their use takes precedence. This character string is not case sensitive. To distinguish upper-case letters from lower-case letters, enclose the applicable part in quotation marks. All spaces and tabs are ignored (except those enclosed in quotation marks).

If the specified character string is not one of the connection character strings listed above, an exception occurs. However, for `Provider`, the specified invalid character string is ignored; no exception occurs. This maintains compatibility with `OleDb Data`

`Provider` in the `DataProvider` layer.

(b) ConnectionTimeout

Type: `int`

Default value: `15`

Description: Acquires the wait time for establishing a connection before retries are cancelled and an error is generated (read only).

(c) Database

Type: `string`

Default value: `""`

Description: Acquires the name of the current database or the database that is used when a connection is established (read only).

(d) LifeTime

Type: `int`

Default value: `60`

Description: Acquires or sets the time remaining before actual disconnection occurs.

Exception: `HiRDBException`

(e) Pooling

Type: `bool`

Default value: `true`

Description: Acquires or sets whether or not pooling is to be performed. If pooling is performed, the value is `true`; if not, the value is `false`.

Exception: `HiRDBException`

(f) State

Type: `System.Data.ConnectionState`

Default value: `ConnectionState.Closed`

Description: Acquires the current connection status (read only).

(3) Methods

(a) BeginTransaction

`BeginTransaction ()`

Return

`HiRDBTransaction`: Object representing a new transaction

Description: Starts the database transaction.

Exception: `HiRDBException`

`BeginTransaction (System.Data.IsolationLevel)`

Argument

`System.Data.IsolationLevel`: One of the `IsolationLevel` values

Return

`HiRDBTransaction`: Object representing a new transaction

Description: Starts the database transaction using the specified `IsolationLevel` value.

Exception: `HiRDBException`

(b) ChangeDatabase

`void ChangeDatabase (string)`

Argument

`string databaseName`: Name of the database to be changed

Return: `void`

Description: Changes the current database for the open `HiRDBConnection` object.

Exception: `HiRDBException`

(c) Clone

`object Clone ()`

Return

`object`: New object which is a copy of this instance

Description: Creates a new object which is a copy of the current instance.

(d) Close

`void Close ()`

Return: `void`

Description: Closes the connection to the database.

(e) CreateCommand

`Hitachi.HiRDB.HiRDBCommand CreateCommand ()`

Return

`HiRDBCommand`: `HiRDBCommand` object

Description: Creates and returns the `HiRDBCommand` object associated with the connection.

(f) Dispose

`void Dispose ()`

Return: `void`

Description: Releases all resources used by `HiRDBConnection`. Because `Disconnect` is called from within this method, when the `HiRDBConnection` object disappears, the database is automatically disconnected.

(g) Open

`void Open ()`

Return: `void`

Description: Opens the database connection with the settings specified in the `ConnectionString` property of the `HiRDBConnection` object.

Exception: `HiRDBException`

15.5.4 HiRDBDataAdapter

(1) Constructor

(a) HiRDBDataAdapter

`void HiRDBDataAdapter ()`

Description: Initializes a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (Hitachi.HiRDB.HiRDBCommand)`

Argument

`HiRDBCommand selectCommand`: `HiRDBCommand` object representing the `SQL SELECT` statement (`SelectCommand` property)

Description: Uses the specified `HiRDBCommand` to initialize a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (string, Hitachi.HiRDB.HiRDBConnection)`

Arguments

`string selectCommandText`: `SQL SELECT` statement

`HiRDBConnection selectConnection`: `HiRDBConnection` object representing the connection

Description: Uses the `HiRDBConnection` specifying the `SQL SELECT` statement to create `HiRDBCommand` (`SelectCommand` property). This constructor initializes a new instance of the `HiRDBDataAdapter` class.

`void HiRDBDataAdapter (string, string)`

Arguments

`string selectCommandText`: `SQL SELECT` statement

`string selectConnectionString`: connection character string

Description: Uses a connection character string to create `HiRDBConnection`. The constructor then uses the created `HiRDBConnection` to create `HiRDBCommand` (`SelectCommand` property). This constructor initializes a new instance of the `HiRDBDataAdapter` class.

(2) Properties

(a) DeleteCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for deleting records from a data set.

(b) InsertCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for inserting new records in a database.

(c) SelectCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for selecting records in a database.

(d) UpDateCommand

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the SQL statement for updating records in a database.

15.5.5 HiRDBDataReader

(1) Constructor

`HiRDBDataReader`

Description: To create `HiRDBDataReader`, you must call the `ExecuteReader` method of the `HiRDBCommand` object without directly using the constructor.

(2) Properties

(a) Depth

Type: `int`

Default value: `0`

Description: Acquires the value indicating the nesting level of the current row.

(b) FieldCount

Type: `int`

Description: Acquires the number of columns in the current row.

(c) IsClosed

Type: `bool`

Default value: `false`

Description: Acquires the value indicating whether or not the data reader is closed. If the data reader is closed, the value is `true`; if not, the value is `false`.

(d) RecordsAffected

Type: `int`

Default value: `0`

Description: Acquires the number of rows changed, inserted, or deleted by execution of an SQL statement.

(3) Methods

(a) Close

```
void Cancel ()
```

Return: `void`

Description: Closes the `HiRDBDataReader` object.

(b) GetBoolean

```
bool GetBoolean (int)
```

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`bool`: Column value

Description: Acquires the value of the specified column as a Boolean value.

Exception: `HiRDBException`

(c) GetByte

`byte GetByte (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`byte`: Unsigned 8-bit integer value in the specified column

Description: Acquires an unsigned 8-bit integer value in the specified column.

Exception: `HiRDBException`

(d) GetBytes

`long GetBytes (int, long, byte[ ], int,int)`

Arguments

`int i`: Ordinal number of the column that begins at 0

`long fieldOffset`: Index of the row where the read operation begins

`byte[] buffer`: Buffer for reading byte streams

`int bufferoffset`: Index of `buffer` where the read operation begins

`int length`: Number of bytes to be read

Return

`long`: Number of bytes actually read

Description: Reads a byte stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.

Exception: `HiRDBException`

(e) GetChar

`char GetChar (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return

`char`: Character value in the specified column

Description: Acquires the character string value in the specified column.

Exception: `HiRDBException`

(f) GetChars

`long GetChars (int, long,char[ ], int, int)`

Arguments

`int i`: Ordinal number of the column that begins at 0

`long fieldOffset`: Index of the row where the read operation begins

`char[] buffer`: Buffer for reading byte streams

`int bufferoffset`: Index of `buffer` where the read operation begins

`int length`: Number of bytes to be read

Return

`long`: Number of characters actually read

Description: Reads a character stream as array into the buffer starting at the specified column offset relative to the specified buffer offset, which is the start position.

Exception: `HiRDBException`

(g) GetData

`GetData (int)`

Argument

`int i`: Ordinal number of the column that begins at 0

Return: Currently not supported.

Description: The purpose of this member is to support the .NET Framework infrastructure. It cannot be used directly in a unique coding that has been created.

(h) GetDataTypeName

`string GetDataTypeName (int)`

Argument

`int i`: Index of the field to be searched

Return

`string`: Data-type information for the specified field

Description: Acquires data-type information for the specified field.

Exception: `HiRDBException`

(i) GetDateTime

`System.DateTime GetDateTime (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.DateTime`: Date and time data value in the specified field

Description: Acquires or sets the date and time data value in the specified field.

Exception: `HiRDBException`

(j) GetDecimal

`decimal GetDecimal (int)`

Argument

`int i`: Index of the field to be searched

Return

`decimal`: Fixed position value in the specified field

Description: Acquires the fixed position value in the specified field.

Exception: `HiRDBException`

(k) GetDouble

`double GetDouble (int)`

Argument

`int i`: Index of the field to be searched

Return

`double`: Double-precision floating-point number in the specified field

Description: Acquires the double-precision floating-point number in the specified field.

Exception: `HiRDBException`

(l) GetEnumerator

`System.Collections.IEnumerator GetEnumerator ()`

Return

`System.Collections.IEnumerator`: `IEnumerator` that can be used to perform iterative operation on a collection

Description: Returns the enumerator that can perform iterative operation on a collection.

(m) GetFieldArrayCount

`int GetFieldArrayCount (int)`

Argument

`int i`: Index of the field to be searched

Return

`int`: Size of field array

Description: Acquires the size of field array.

Exception: `HiRDBException`

(n) GetFieldType

`System.Type GetFieldType (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.Type`: `Type` information corresponding to the type of `object` that is returned from `GetValue`

Description: Acquires `Type` information corresponding to the type of `Object` that is returned from `GetValue`.

Exception: `HiRDBException`

(o) GetFloat

`float GetFloat (int)`

Argument

`int i`: Index of the field to be searched

Return

`float`: Single-precision floating-point number in the specified field

Description: Acquires the single-precision floating-point number in the specified field.

Exception: `HiRDBException`

(p) GetGuid

`System.Guid GetGuid (int)`

Argument

`int i`: Index of the field to be searched

Return

`System.Guid`: `GUID` value of the specified field

Description: Returns the `GUID` value of the specified field.

(q) GetInt16

`short GetInt16 (int)`

Argument

`int i`: Index of the field to be searched

Return

`short`: Signed 16-bit integer value in the specified field

Description: Acquires a signed 16-bit integer value in the specified field.

Exception: `HiRDBException`

(r) GetInt32

`int GetInt32 (int)`

Argument

`int i`: Index of the field to be searched

Return

`int`: Signed 32-bit integer value in the specified field

Description: Acquires a signed 32-bit integer value in the specified field.

Exception: `HiRDBException`

(s) GetInt64

`long GetInt64 (int)`

Argument

`int i`: Index of the field to be searched

Return

`long`: Signed 64-bit integer value in the specified field

Description: Acquires a signed 64-bit integer value in the specified field.

Exception: `HiRDBException`

(t) GetName

`string GetName (int)`

Argument

`int i`: Index of the field to be searched

Return

`string`: Field name (if there is no value to be returned, returns the `null` character string (`""`))

Description: Acquires the name of the field to be searched.

Exception: `HiRDBException`

(u) GetOrdinal

`int GetOrdinal (string)`

Argument

`string name`: Name of the field to be searched

Return

`int`: Index of the specified field

Description: Returns the index of the specified field.

Exception: `HiRDBException`

(v) GetSchemaTable

`System.Data.DataTable GetSchemaTable ()`

Return

`System.Data.DataTable`: `DataTable` that describes column metadata

Description: Returns the `DataTable` that describes `HiRDBDataReader`'s column metadata.

Exception: `HiRDBException`

(w) GetString

`string GetString (int)`

Argument

`int i`: Index of the field to be searched

Return

`string`: Character string in the specified field

Description: Acquires a character string in the specified field.

Exception: `HiRDBException`

(x) GetValue

`object GetValue (int)`

Argument

`int i`: Index of the field to be searched

Return

`object`: `Object` for storing the returned field value, if any

Description: Returns a value in the specified field.

Exception: `HiRDBException`

`object GetValue (int, int)`

Arguments

`int i`: Index of the field to be searched

`int j`: Index of the field to be searched

Return

`object`: `Object` for storing the returned field value, if any

Description: Returns a value in the specified field (for array).

Exception: `HiRDBException`

(y) GetValues

`int GetValues (object[ ])`

Argument

`object values`: `Object` array which is the target of a copy operation on the attribute field

Return

`int`: Number of `Object` instances in array

Description: Acquires all attribute fields in the current record collection.

(z) IsDBNull

`bool IsDBNull (int)`

Argument

`int i`: Index of the field to be searched

Return

`bool`: If the specified field is set to `null`, the value is `true`; if not, the value is false.

Description: Returns a value indicating whether or not the specified field is set to null.

Exception: HiRDBException

(aa) NextResult

`bool NextResult ()`

Return

bool: If there are further rows, the value is true; if not, the value is false.

Description: Advances the data reader to the next result when the result of a batch SQL statement is read.

Exception: HiRDBException

(ab) Read

`bool Read ()`

Return

bool: If there are further rows, the value is true; if not, the value is false.

Description: Advances HiRDBDataReader to the next record.

Exception: HiRDBException

15.5.6 HiRDBException

(1) Properties

(a) ErrorCode

Type: int

Default value: 0

Description: Acquires an error code as an int.

(b) Message

Type: String

Default value: ""

Description: Acquires text with a complete error.

15.5.7 HiRDBParameter

(1) Constructor

(a) HiRDBParameter

`void HiRDBParameter ()`

Description: Initializes a new instance of the `HiRDBParameter` class.

`void HiRDBParameter (string, object)`

Arguments

`string name`: Name of the parameter to be allocated (`ParameterName` property)

`object value`: Value of the new `HiRDBParameter` object (`Value` property)

Description: Specifies the parameter name and `HiRDBParameter` object to initialize a new instance of the `HiRDBParameter` class.

`void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType)`

Arguments

`string name`: Name of the parameter to be allocated (`ParameterName` property)

`Hitachi.HiRDB.HiRDBType dataType`: One of the `HiRDBType` values (`HiRDBType` property)

Description: Specifies a parameter name and data type to initialize a new instance of the `HiRDBParameter` class.

`void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int)`

Arguments

`string name`: Name of the parameter to be allocated (`ParameterName` property)

`Hitachi.HiRDB.HiRDBType dataType`: One of the `HiRDBType` values (`HiRDBType` property)

`int size`: Parameter width (`Size` property)

Description: Uses a parameter name, data type, and length to initialize a new instance of the `HiRDBParameter` class.

`void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int, string)`

Arguments

`string name`: Name of the parameter to be allocated (`ParameterName` property)

`Hitachi.HiRDB.HiRDBType dataType`: One of the `HiRDBType` values (`HiRDBType` property)

`int size`: Parameter width (`Size` property)

`string srcColumn`: Name of the source column (`SourceColumn` property)

Description: Specifies a parameter name, data type, length, and source column name to initialize a new instance of the `HiRDBParameter` class.

```
void HiRDBParameter (string, Hitachi.HiRDB.HiRDBType, int,
System.Data.ParameterDirection, Byte, Byte, String,
System.Data.DataRowVersion, Object)
```

Arguments

`string parameterName`: Parameter name (`ParameterName` property)

`Hitachi.HiRDB.HiRDBType dataType`: One of the `HiRDBType` values (`HiRDBType` property)

`int size`: Parameter width (`Size` property)

`System.Data.ParameterDirection direction`: One of the `ParameterDirection` values (`Direction` property)

`byte precision`: Total length in digits used to resolve `Value` (`Precision` property)

`byte scale`: Length of the fractional part in digits used to resolve `Value` (`Scale` property)

`string srcColumn`: Name of the source column (`SourceColumn` property)

`System.Data.DataRowVersion srcVersion`: One of the `DataRowVersion` values (`SourceVersion` property)

`object value`: `Object` which is the value of `HiRDBParameter` (`Value` property)

Description: Specifies a parameter name, data type, length, source column name, parameter direction, precision of numeric value, and other properties to initialize a new instance of the `HiRDBParameter` class.

(2) Properties

(a) DbType

Type: `System.Data.DbType`

Default value: `DbType.String`

Description: Acquires or sets `DbType` for a parameter. When `DbType` is to be set, this member sets the corresponding data type in the `HiRDBType` property according to Table 15-3.

(b) Direction

Type: `System.Data.ParameterDirection`

Default value: `ParameterDirection.Input`

Description: Acquires or sets a value indicating whether the parameter is input only, output only, bidirectional, or the stored procedure's return value.

(c) HiRDBType

Type: `Hitachi.HiRDB.HiRDBType`

Default value: `HiRDBType.MVarChar`

Description: Acquires or sets an enumeration indicating the data type in HiRDB. When the enumeration is to be set, this member sets the corresponding data type in the `DbType` property according to Table 15-4.

`HiRDBType` enumeration:

> `Integer`, `SmallInt`, `Decimal`, `Float`, `SmallFlt`, `Char`, `VarChar`, `NChar`, `NVarChar`, `MChar`, `MVarChar`, `Date`, `Time`, `TimeStamp`, `IntervalYearToDay`, `IntervalHourToSecond`, `Blob`, `Binary`

(d) IsNullable

Type: `bool`

Default value: `true` (fixed)

Description: Acquires a value indicating whether or not the parameter accepts the `null` value (read only). If the `null` value is accepted, the value is `true`; if not, the value is `false`.

(e) ParameterName

Type: `string`

Default value: `""`

Description: Acquires or sets the name of the `HiRDBParameter`.

(f) Precision

Type: `byte`

Default value: `0`

Description: Acquires or sets the number of significant digits for a numeric parameter.

(g) Repetition

Type: `short`

Default value: `1`

Description: Acquires or sets an array structure in HiRDB.

(h) Scale

Type: `byte`

Default value: `0`

Description: Acquires or sets the number of decimal places for a numeric parameter.

(i) Size

Type: `int`

Default value: `0`

Description: Sets the size of the column definition length or character string data that can be stored after character code conversion. Also acquires the current setting. For `TIMESTAMP (DateTime)`, this value is the number of digits in the fractional part.

(j) SourceColumn

Type: `string`

Default value: `""`

Description: Acquires or sets the name of the source column that has been assigned to `DataSet` and is used to read or return `Value`.

(k) SourceVersion

Type: `System.Data.DataRowVersion`

Default value: `DataRowVersion.Default`

Description: Acquires or sets the `DataRowVersion` that is used to read `Value`.

(l) Value

Type: `object`

Default value: `null`

Description: Acquires or sets a parameter value.

(3) Methods

(a) Clone

```
object Clone ()
```

Return

`object`: New object which is a copy of this instance

Description: Creates a new object which is a copy of the current instance.

15.5.8 HiRDBParameterCollection

(1) Constructor

(a) HiRDBParameterCollection

`void HiRDBParameterCollection ()`

Description: Initializes a new instance of the `HiRDBParameterCollection` class.

(2) Properties

(a) Count

Type: `int`

Default value: `0`

Description: Acquires the number of `HiRDBParameter` objects stored in `HiRDBParameterCollection` (read only).

(b) IsFixedSize

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether the size of `HiRDBParameterCollection` is fixed (read only). If the size of the value is fixed, the value is `true`; if not, the value is `false`.

(c) IsReadOnly

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether or not `HiRDBParameterCollection` is read only (read only). If it is read only, the value is `true`; if not, the value is `false`.

(d) IsSynchronized

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether or not an access to `HiRDBParameterCollection` is synchronized (thread-safe) (read only). If the access is synchronized, the value is `true`; if not, the value is `false`.

(e) SyncRoot

Type: `object`

Default value: `null`

Description: Acquires an object that can be used to synchronize an access to `HiRDBParameterCollection` (read only).

(3) Methods

(a) Add

`int Add (object)`

Argument

`object value`: `HiRDBParameter` object to be added to `HiRDBParameterCollection`

Return

`int`: Index in the new `HiRDBParameter` object's collection

Description: Adds items to `HiRDBParameterCollection`.

`int Add (Hitachi.HiRDB.HiRDBParameter)`

Argument

`HiRDBParameter value`: `HiRDBParameter` to be added to `HiRDBParameterCollection`

Return

`int`: Index of the new `HiRDBParameter`

Description: Adds items to `HiRDBParameterCollection`.

`int Add (string, object)`

Arguments

`string parameterName`: Parameter name

`object parameterValue`: Parameter value

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name and value of the parameter to add items to `HiRDBParameterCollection`.

`int Add (string, HiRDBType)`

Arguments

`string parameterName`: Parameter name

`HiRDBType dataType`: One of the `HiRDBType` values

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name and data type of the parameter to add items to `HiRDBParameterCollection`.

`int Add (string, HiRDBType, int)`

Arguments

`string parameterName`: Parameter name

`HiRDBType dataType`: One of the `HiRDBType` values

`int size`: Parameter size

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name, data type, and size of the parameter to add items to `HiRDBParameterCollection`.

`int Add (string, HiRDBType, int, string)`

Arguments

`string parameterName`: Parameter name

`HiRDBType dataType`: One of the `HiRDBType` values

`int size`: Parameter size

`string srcColumn`: Name of the source column

Return

`int`: Index of the new `HiRDBParameter`

Description: Specifies the name, data type, size, and source column of the parameter to add items to `HiRDBParameterCollection`.

(b) Clear

`void Clear ()`

Return: `void`

Description: Deletes all items from `HiRDBParameterCollection`.

(c) Contains

`bool Contains (string)`

Argument

`string parameterName`: Parameter name

Return

`bool`: If the parameter is stored in the collection, the value is `true`; if not, the value is `false`.

Description: Acquires a value indicating whether or not `HiRDBParameter` is in the collection.

`bool Contains (object)`

Argument

`object value`: `Object` that is searched for in `HiRDBParameterCollection`

Return

`bool`: If `Object` is in `HiRDBParameterCollection`, the value is `true`; if not, the value is `false`.

Description: Acquires a value indicating whether or not `HiRDBParameter` is in the collection.

(d) CopyTo

`void CopyTo (System.Array, int)`

Arguments

`System.Array array`: One-dimensional `Array` to which elements are copied from `HiRDBParameterCollection`

`int index`: Index number, beginning at 0, at the location where `value` is inserted

Return: `void`

Description: Copies the elements of `HiRDBParameterCollection` to `Array` using `Array`'s specific index as the start position.

(e) GetEnumerator

`System.Collections.IEnumerator GetEnumerator ()`

Return

`System.Collections.Ienumerator`: `IEnumerator` that can be used to perform iteration processing on a collection

Description: Returns the enumerator that can perform iterative operation on a collection.

(f) IndexOf: overload

`int IndexOf (string)`

Argument

`string parameterName`: Parameter name

Return

`int`: Location of `HiRDBParameterCollection` in the collection that begins at 0

Description: Acquires the location of `HiRDBParameter` in a collection.

Exception: `HiRDBException`

`int IndexOf (object)`

Argument

`object value`: `Object` that is searched for in `HiRDBParameterCollection`

Return

`int`: If the object is in the list, the value is the index of `value`; if not, the value is `-1`.

Description: Acquires the location of `HiRDBParameter` in a collection.

(g) Insert

`void Insert (int, Hitachi.HiRDB.HiRDBParameter)`

Arguments

`int index`: Index number, which begins at 0, at the location where `value` is inserted

`HiRDBParameter value`: `HiRDBParameter` to be added to `HiRDBParameterCollection`

Return: `void`

Description: Inserts an item at the specified location in `HiRDBParameterCollection`.

(h) Remove

`void Remove (object)`

Argument

`object value`: `HiRDBParameter` to be deleted from `HiRDBParameterCollection`

Return: `void`

Description: Deletes the first occurrence of the specified object in `HiRDBParameterCollection`.

(i) RemoveAt

`void RemoveAt (string)`

Argument

`string parameterName`: Parameter name

Return: `void`

Description: Deletes `HiRDBParameter` from a collection.

Exception: `HiRDBException`

`void RemoveAt (int)`

Argument

`int index`: Index of the item to be deleted that begins at 0

Return: `void`

Description: Deletes `HiRDBParameter` from a collection.

15.5.9 HiRDBRowUpdatedEventArgs

(1) Constructor

(a) HiRDBRowUpdatedEventArgs

```
void HiRDBRowUpdatedEventArgs (System.Data.DataRow,
System.Data.IdbCommand, System.Data.StatementType,
System.Data.Common.DataTableMapping)
```

Arguments

`System.Data.DataRow dataRow`: `DataRow` that was sent through `Update`

`System.Data.IDbCommand command`: `IDbCommand` that was executed when `Update` was called

`System.Data.StatementType statementType`: Type of SQL statement that was executed

`System.Data.Common.DataTableMapping tableMapping`: `DataTableMapping` that was sent through `Update`

Description: Initializes a new instance of the `HiRDBRowUpdatedEventArgs` class.

(2) Properties

(a) Command

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires the `HiRDBCommand` that is executed when `Update` is called (read only).

15.5.10 HiRDBRowUpdatingEventArgs

(1) Constructor

(a) HiRDBRowUpdatingEventArgs

```
void HiRDBRowUpdatingEventArgs (System.Data.DataRow,
System.Data.IDbCommand, System.Data.StatementType,
System.Data.Common.DataTableMapping)
```

Arguments

`System.Data.DataRow dataRow`: `DataRow` that executes `Update`

`System.Data.IDbCommand command`: `IDbCommand` that is executed when `Update` is called

`System.Data.StatementType statementType`: Type of SQL statement to be executed

`System.Data.Common.DataTableMapping tableMapping`: `DataTableMapping` that is sent through `Update`

Description: Initializes a new instance of the `HiRDBRowUpdatingEventArgs` class.

(2) Properties

(a) Command

Type: `HiRDBCommand`

Default value: `null`

Description: Acquires or sets the `HiRDBCommand` that is executed during `Update` processing.

15.5.11 HiRDBTransaction

(1) Constructor

(a) HiRDBTransaction

```
void HiRDBTransaction (Hitachi.HiRDB.HiRDBConnection)
```

Argument

`HiRDBConnection rConnection`: `Connection` object (`Connection` property)

Description: Initializes a new instance of the `HiRDBTransaction` class.

```
void HiRDBTransaction (Hitachi.HiRDB.HiRDBConnection,
System.Data.IsolationLevel)
```

Arguments

`HiRDBConnection rConnection`: `Connection` object (`Connection` property)

`System.Data.IsolationLevel eIsolationLevel`: Transaction lock operation (`IsolationLevel` property)

Description: Initializes a new instance of the `HiRDBTransaction` class.

(2) Properties

(a) Connection

Type: `HiRDBConnection`

Default value: `null`

Description: Specifies the `HiRDBConnection` object used to associate a transaction (read only).

(b) IsCompleted

Type: `bool`

Default value: `false`

Description: Acquires a value indicating whether or not the transaction is completed (read only). If the transaction is completed, the value is `true`; if not, the value is `false`.

(c) IsolationLevel

Type: `System.Data.IsolationLevel`

Default value: `IsolationLevel.ReadCommitted`

Description: Specifies this transaction's `IsolationLevel` (read only).

(3) Methods

(a) Commit

```
void Commit ()
```

Return: `void`

Description: Commits a database transaction.

Exception: `HiRDBException`

(b) Rollback

`void Rollback ()`

Return: `void`

Description: Rolls back a database transaction from the hold status.

Exception: `HiRDBException`

15.6 Notes about HiRDB.NET Data Provider

Table 15-2 gives notes about HiRDB.NET Data Provider.

Table 15-2: Notes about HiRDB.NET Data Provider

Object	Method or property	Details
HiRDBCommand	CommandTimeout property	The setting is ignored because the timeout value during execution depends on the settings in the client environment definition (PDCWAITTIME, PDSWAITTIME, PDSWATCHTIME).
	Cancel method	System.NotSupportedException is returned because there is no cancellation function.
	ExecuteReader method	When the CommandBehavior.KeyInfo, CommandBehavior.SchemaOnly, or CommandBehavior.SequentialAccess argument is specified, it is treated as CommandBehavior.Default because a function for acquiring only column or primary key information is not available.
	UpdatedRowSource property	When UpdatedRowSource.Both or UpdatedRowSource.FirstReturnedRecord is specified, HiRDBException is returned because there is no batch query function that returns rows.
HiRDBConnection	Database property	The null character always results because there is no function for acquiring database names.
	State property	ConnectionState.Connecting, ConnectionState.Executing, ConnectionState.Fetching, or ConnectionState.Broken will never result because this property is a reserved value for future product versions.
	BeginTransaction method	IsolationLevel is ignored, if specified, because this method is set for each SQL statement or acquired from HiRDB environment variables.
	ChangeDatabase method	System.NotSupportedException is returned because a function for changing the connected database is not available.
HiRDBDataReader	Depth property	Always 0 because there is no hierarchy concept.
	GetBoolean method	NotSupportedException is returned because there is no corresponding type.

Object	Method or property	Details
	GetByte method	NotSupportedException is returned because there is no corresponding type.
	GetChar method	NotSupportedException is returned because there is no corresponding type.
	GetData method	NotSupportedException is returned because there is no corresponding type.
	GetGuid method	NotSupportedException is returned because there is no corresponding type.
	NextResult method	false is returned because there is no multiple record set function.
HiRDBParameter	DbType property	If DbType.Boolean, DbType.Currency, DbType.Guid, or DbType.VarNumeric is specified, HiRDBException is returned because there is no corresponding type.
	Direction property	If HiRDBCommand class's ExecuteNonQuery, ExecuteReader, ExecuteScalar, or Prepare method is executed while Direction.ReturnValue is specified, HiRDBException is returned because there is no function for acquiring the stored procedure's return value.
	IsNullable property	Acquisition only; setting is not available. (The null value can always be specified.)

15.7 Data types of HiRDB.NET Data Provider

15.7.1 DbType and HiRDBType properties

When the `DbType` property of the `HiRDBParameter` class is set, the `HiRDBType` property of the same class is automatically set. When the `HiRDBType` property is set, the `DbType` property is automatically set. Table 15-3 lists the `HiRDBType` property values that are automatically set when the `DbType` property is set, and Table 15-4 lists the `DbType` property values that are automatically set when the `HiRDBType` property is set.

Table 15-3: HiRDBType property values that are automatically set when the DbType property is set

DbType property	HiRDBType property
`AnsiString`	`VarChar`
`AnsiStringFixedLength`	`Char`
`Binary`	`Binary`
`Boolean`	[`NotSupportedException` exception]
`Byte`	`SmallInt`
`Currency`	[`NotSupportedException` exception]
`Date`	`Date`
`DateTime`	`TimeStamp`
`Decimal`	`Decimal`
`Double`	`Float`
`Guid`	[`NotSupportedException` exception]
`Int16`	`SmallInt`
`Int32`	`Integer`
`Int64`	`Decimal`
`Object`	`Binary`
`SByte`	`SmallInt`
`Single`	`SmallFlt`
`String`	`MvarChar`

DbType property	HiRDBType property
StringFixedLength	Mchar
Time	Time
UInt16	Integer
UInt32	Decimal
UInt64	Decimal
VarNumeric	[NotSupportedException exception]

Table 15-4: DbType property values that are automatically set when the HiRDBType property is set

HiRDBType property	DbType property
Binary	Object
Blob	Object
Char	AnsiStringFixedLength
Date	Date
Decimal	Decimal
Float	Double
Integer	Int32
IntervalYearToDay	String
IntervalHourToSecond	String
MChar	StringFixedLength
MVarChar	String
NChar	StringFixedLength
NVarChar	String
SmallFlt	Single
SmallInt	Int16
Time	Time
TimeStamp	DateTime
VarChar	AnsiString

15.7.2 Data types and accessories used by a UAP

Table 15-5 lists the data types that are set in the `Value` property of the `HiRDBParameter` class, for example during execution of the `INSERT` and `Get`*XXXX* methods of the `HiRDBDataReader` class that are used during execution of `SELECT`. Note that HiRDB's `NULL` is represented by `DBNull.Value` of the .NET Framework type.

Table 15-5: Data types and accessories for HiRDB-type UAPs

Classification	HiRDB data type	.NET Framework type used by UAPs, for example in INSERT	Accessory used by UAP for SELECT
Character	CHAR[ACTER]	String	GetString()
	VARCHAR/CHAR[ACTER]VARYING	String	GetString()
	NCHAR/NATIONAL CHAR[ACTER]	String	GetString()
	NVARCHAR/NCHAR VARYING	String	GetString()
	MCHAR	String	GetString()
	MVARCHAR	String	GetString()
Numeric value	[LARGE]DEC[IMAL]/NUMERIC	Decimal	GetDecimal()
	SMALLINT	Int16	GetInt16()
	INT[EGER]	Int32	GetInt32()
	SMALLFLT/REAL	Single	GetFloat()
	FLOAT/DOUBLE PRECISION	Double	GetDouble()
Date and time	DATE	DateTime	GetDateTime()
	TIME	DateTime	GetDateTime()
	TIMESTAMP	DateTime	GetDateTime()
Other	BINARY	Byte[]	GetBytes()
	BLOB	Byte[]	GetBytes()
	INTERVAL YEAR TO DAY	String	GetString()
	INTERVAL HOUR TO SECOND	TimeSpan	GetString()

15.7.3 Type conversion by HiRDB.NET Data Provider

When no .NET Framework type or accessory listed in Table 15-5 is used, type conversion takes place automatically within the HiRDB data provider. No .NET Framework type or accessory is used when `Int32`-type data is inserted in a table that contains items with the `CHAR` attribute or the `GetInt32` method is used for acquisition.

Tables 15-6 and 15-7 list the type conversions for `INSERT`, and Tables 15-8 and 15-9 list the type conversions for `SELECT`.

For the definition of symbols used in tables 15-6 through 15-9, see *15.7.3(1) Definition of symbols*.

Table 15-6: List of type conversions for INSERT (1/2)

.NET Framework type	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`Boolean`	E1	E1	E1	E1	E1	E1	E1	E1	E1
`Int16`	N	N	N	N	N	N	N	E1	E1
`Int32`	N	C1	N	N	N	N	N	E1	E1
`Int64`	C2	C1	N	N	N	N	N	E1	E1
`UInt16`	N	N	N	N	N	N	N	E1	E1
`UInt32`	N	C1	N	N	N	N	N	E1	E1
`UInt64`	C2	C1	N	N	N	N	N	E1	E1
`Single` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Single` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Double` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Double` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Decimal` data with fractional part	C4	C3	N	N	N	N	N	E1	E1
`Decimal` data with no fractional part	C2	C1	N	N	N	N	N	E1	E1
`Char`	N1	N1	E1	E1	E1	N	N	N	N
`Char[]`	E1	E1	E1	E1	E1	E1	E1	E1	E1

.NET Framework type	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
String	C2	C1	N	N	N	N	N	N	N
DateTime	E1	E1	E1	E1	E1	N	N	E1	E1
TimeSpan	E1	E1	E1	E1	E1	N	N	E1	E1
Guid	E1	E1	E1	E1	E1	N	N	E1	E1
Byte	N	N	N	N	N	N	N	E1	E1
Byte[]	E1	E1	E1	E1	E1	E1	E1	E1	E1
Sbyte	N	N	N	N	N	N	N	E1	E1
SByte[]	E1	E1	E1	E1	E1	E1	E1	E1	E1

Table 15-7: List of type conversions for INSERT (2/2)

.NET Framework type	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
Boolean	E1	E1	E1	E1	E1	E1	E1	E1	E1
Int16	N	N	E1	E1	E1	E2	E2	E1	E1
Int32	N	N	E1	E1	E1	E2	E2	E1	E1
Int64	N	N	E1	E1	E1	E2	E2	E1	E1
UInt16	N	N	E1	E1	E1	E2	E2	E1	E1
UInt32	N	N	E1	E1	E1	E2	E2	E1	E1
UInt64	N	N	E1	E1	E1	E2	E2	E1	E1
Single data with fractional part	N	N	E1	E1	E1	E3	E3	E1	E1
Single data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
Double data with fractional part	N	N	E1	E1	E1	E3	E3	E1	E1
Double data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
Decimal data with fractional part	N	N	E1	E1	E1	E3	E3	E1	E1

.NET Framework type	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
`Decimal` data with no fractional part	N	N	E1	E1	E1	E2	E2	E1	E1
`Char`	N	N	E1	E1	E1	E3	E3	E1	E1
`Char[]`	E1	E1	E1	E1	E1	E3	E3	E1	E1
`String`	N	N	N	N	N	N	N	E1	E1
`DateTime`	N	N	N	N	N	E3	E3	E1	E1
`TimeSpan`	N	N	E1	E1	E1	E3	N	E1	E1
`Guid`	N	N	E1	E1	E1	E3	E3	E1	E1
`Byte`	N	N	E1	E1	E1	E3	E3	N	N
`Byte[]`	E1	E1	E1	E1	E1	E3	E3	N	N
`Sbyte`	N	N	E1	E1	E1	E3	E3	E1	E1
`SByte[]`	E1	E1	E1	E1	E1	E3	E3	E1	E1

Note 1: `INSERT` operation on `NCHAR/NVARCHAR`

If the size of data obtained after S-JIS conversion consists of an odd number of bytes, the `[Hitachi.HiRDB.HiRDBException]KFPZ24026-E` format conversion error occurs.

Note 2: During array `INSERT`

If the type is not an `Object` array type, the `[Hitachi.HiRDB.HiRDBException] KFPZ24026-E` format conversion error occurs. Because no array can be inserted in `BLOB`, the same error occurs if an attempt is made.

Table 15-8: List of type conversions for SELECT (1/2)

Accessory	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
`GetBoolean`	E4	E4	E4	E4	E4	E4	E4	E4	E4
`GetByte`	E4	E4	E4	E4	E4	E4	E4	E4	E4
`GetBytes`	N	N	E1	N	N	N	N	N	N
`GetChar`	E4	E4	E4	E4	E4	E4	E4	E4	E4

Accessory	HiRDB data type								
	I	SI	DE	F	SF	C	VC	NC	NVC
GetChars	N	N	N	N	N	N	N	N	N
GetData	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetDateTime	E1	E1	E1	E1	E1	C6	C6	C6	C6
GetDecimal	N	N	N	N	N	C7	C7	C7	C7
GetDouble	N	N	N	N	N	C8	C8	C8	C8
GetFloat	N	N	N	N	N	C9	C9	C9	C9
GetGuid	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetInt16	C1	N	C1	C1	C1	C1	C1	C1	C1
GetInt32	N	N	C2	C2	C2	C2	C2	C2	C2
GetInt64	N	N	C10	C10	C10	C10	C10	C10	C10
GetString	N	N	N	N	N	N	N	N	N
GetValue	N	N	N	N	N	N	N	N	N
GetValues	N	N	N	N	N	N	N	N	N

Table 15-9: List of type conversions for SELECT (2/2)

Accessory	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
GetBoolean	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetByte	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetBytes	N	N	E1	E1	E1	E1	E1	N	N
GetChar	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetChars	N	N	E1	E1	E1	E1	E1	E1	E1
GetData	E4	E4	E4	E4	E4	E4	E4	E4	E4
GetDateTime	C6	C6	N	N	N	E1	E1	E1	E1
GetDecimal	C7	C7	E1	E1	E1	E1	E1	E1	E1
GetDouble	C8	C8	E1	E1	E1	E1	E1	E1	E1

Accessory	HiRDB data type								
	MC	MVC	DA	T	TS	IY	IHS	BI	BL
`GetFloat`	C9	C9	E1	E1	E1	E1	E1	E1	E1
`GetGuid`	E4	E4	E4	E4	E4	E4	E4	E4	E4
`GetInt16`	C1	C1	E1	E1	E1	E1	E1	E1	E1
`GetInt32`	C2	C2	E1	E1	E1	E1	E1	E1	E1
`GetInt64`	C10	C10	E1	E1	E1	E1	E1	E1	E1
`GetString`	N	N	N	N	N	N	N	N	N
`GetValue`	N	N	N	N	N	N	N	N	N
`GetValues`	N	N	N	N	N	N	N	N	N

Note 1: During `DATE` acquisition

When the `GetDateTime` method is used, `00:00:00` is set in the time field. When the `GetString` method is used, the value is set in the format *YYYY*/*MM*/*DD*.

Note 2: During `TIME`/`TIMESTAMP` acquisition

When the `GetDateTime` method is used, the current date is set in the date field. When the `GetString` method is used, the value is set in the following format:

`TIME`: *hh*:*mm*:*ss*

`TIMESTAMP(0)`: *YYYY*/*MM*/*DD* *hh*:*mm*:*ss*

`TIMESTAMP(2)`: *YYYY*/*MM*/*DD* *hh*:*mm*:*ss*.*nn*

`TIMESTAMP(4)`: *YYYY*/*MM*/*DD* *hh*:*mm*:*ss*.*nnnn*

`TIMESTAMP(6)`: *YYYY*/*MM*/*DD* *hh*:*mm*:*ss*.*nnnnnn*

Note 3: During `INTERVALYEARTODAY` acquisition

When the `GetString` method is used, the value is set in the format ±*YYYY*/*MM*/*DD*.

Note 4: During `INTERVALHOURTOSECOND` acquisition

When the `GetString` method is used, the value is set in the format ±*hh*:*mm*:*ss*.

(1) Definition of symbols

(a) HiRDB data types

The following table defines the symbols used for the HiRDB data types:

Symbol	Definition
I	INTEGER
SI	SMALLINT
DE	DECIMAL and LARGE DECIMAL
F	FLOAT/DOUBLE PRECISION
SF	SMALLFLT and REAL
C	CHARACTER
VC	VARCHAR
NC	NCHAR and NATIONAL CHARACTER
NVC	NVARCHAR
MC	MCHAR
MVC	MVARCHAR
DA	DATE
T	TIME
TS	TIMESTAMP
IY	INTERVAL YEAR TO DAY
IHS	INTERVAL HOUR TO SECOND
BI	BINARY
BL	BLOB

(b) Whether or not type conversion is supported

N indicates normal; C indicates a conditional; and E indicates error. Some of these letters are followed by a number; they are defined as follows:

Symbol	Definition
N	Numeric character code is set.
C1	-32768 to 32767: Normal 0 to 32767: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error

Symbol	Definition
C2	-2147483648 to 2147483647: Normal 0 to 2147483647: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C3	-32768 to 32767: Normal (rounded) Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C4	-2147483648 to 2147483647: Normal (rounded) Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C5	0 to 255: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C6	DateTime format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C7	Decimal format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C8	Double format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C9	Float format data: Normal Other: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
C10	-9223372036854775808 to 9223372036854775807: Normal Out of range: [Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
E1	[Hitachi.HiRDB.HiRDBException] KFPZ24026-E format conversion error
E2	[Hitachi.HiRDB.HiRDBException] KFPZ24107-E Decimal, date and time, time interval type overflow
E3	[Hitachi.HiRDB.HiRDBException] KFPZ24106-E date and time, time interval type format error
E4	[System.NotSupportedException] unsupported error

15.8 Example of a UAP using HiRDB.NET Data Provider

This section describes an example of a UAP using HiRDB.NET Data Provider.

Although the sample program is coded in Visual C# .NET, its contents are almost identical in Visual Basic.NET. If necessary, change the information as appropriate.

15.8.1 Connecting to the database

The following example connects to HiRDB and then disconnects from HiRDB:

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {

            try
            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");  ...1

                // Connect to the database
                cn.Open(); ..........................................2

                // Disconnect from the database
                cn.Close(); .........................................3
            }

            catch (HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            } ......................................................4
        }
    }
}
```

Explanation

1. First, create a `HiRDBConnection` object. This object manages all communications with HiRDB. Because `Disconnect` is called from within the `HiRDBConnection: Dispose` method, when this object disappears,

the database is automatically disconnected.

For this method, you must specify one `string`-type argument. The character string to be specified is called a *connection character string*. This is the same type of connection character string as those used for `Connection` in ADO and ADO.NET. For details about the character strings that can be specified, see *15.5.3(2)(a) ConnectionString*.

2. To connect to the database, use the `Open` method.
3. To disconnect from the database, use the `Close` method. Using the `Close` method while a connection is not established does not result in an exception.
4. An exception occurs if the server is not running, communication is disabled, the SQL statement is invalid, or in similar cases. Basically, a block using HiRDB.NET Data Provider detects exceptions by `try` through `catch`, and then displays an exception message.

   In the case of an overall HiRDB error, `System.Exception` occurs, and in the case of a HiRDB.NET Data Provider-specific error, `HiRDBException` occurs. Make sure that `System.Exception` is not abbreviated as `Exception`.

   A HiRDB Client Library or HiRDB.NET Data Provider-specific error code is stored in the `ErrorCode` property of the exception object that is created by HiRDB.NET Data Provider.

   A 3-digit (*–XXX*) or 4-digit (*–XXXX*) error code indicates `KFPA1`*XXXX* and a 5-digit error code (`-24`*XXX*) indicates `KFPZ24`*XXX*.

15.8.2 Executing the SQL statement

This example creates a table named `ex`:

```
using System;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)
        {
            try
```

```
            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");  ...1

                // Connect to the database
                cn.Open();

                // Create a Command object
                HiRDBCommand cm = new HiRDBCommand();


                // Create a table
                cm.Connection = cn;
                cm.CommandText = "create table ex (a int)";
                cm.ExecuteNonQuery();  ................................1

                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)

            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

Explanation

1. To execute an SQL statement, use the `Execute` method. Specify a `string`-type SQL statement as is in the `CommandText` property of `HiRDBCommand`. This method can execute most SQL statements. Special SQL statements such as `commit` cannot be executed by this method, as well as statements such as `select` that must receive a result set. To execute these SQL statements, use dedicated methods.

15.8.3 Executing a transaction

This example inserts data `1` to the `ex` table:

```
using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)

        {
            // Create a Connection object
            HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

            // Connect to the database
            cn.Open();

            // Create a Transaction object
            HiRDBTransaction tran;
            // Start of transaction
            tran = cn.BeginTransaction(IsolationLevel.ReadCommitted);  ..1
            // Create a Command object
            HiRDBCommand cm = new HiRDBCommand();
            cm.Connection = cn;
            cm.Transaction = tran;
            try

            {
                // Insert data in the table
                cm.CommandText = "insert into ex values (1)";
                cm.ExecuteNonQuery();

                // Transaction was successful
                tran.Commit();  .........................................2

                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)
            {
                // Transaction failed
                tran.Rollback();  .......................................3


                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                // Transaction failed
                tran.Rollback();  .......................................3

                Console.WriteLine(ex);
            }
        }
    }
}
```

Explanation

1. To start a transaction, use the `BeginTransaction` method.
2. To complete the transaction, call the `Commit` method.
3. To restore, call the `Rollback` method.

15.8.4 Executing a search statement

This example displays all table data:

```
using System;
using System.Data;
using Hitachi.HiRDB;

namespace test_C
{
    class Sample
    {
        [STAThread]
        static void Main(string[] args)

        {
            try
            {
                // Create a Connection object
                HiRDBConnection cn = new HiRDBConnection("dsn=pc;");

                // Connect to the database
                cn.Open();

                // Create a Command object
                HiRDBCommand cm = new HiRDBCommand();
                cm.Connection = cn;
                cm.CommandText = "select a from ex";


                // Create a DataReader object
                HiRDBDataReader rd = cm.ExecuteReader();  ...............1
                int i;
                while(rd.Read())
                {
                    for (i = 0 ; i < rd.FieldCount ; i++)
                    {
                     Console.WriteLine(rd.GetName(i) + " - " +rd.GetValue(i));
                    }
                }  ......................................................2
```

```
                // Disconnect from the database
                cn.Close();
            }
            catch (HiRDBException ex)
            {
                Console.WriteLine(ex);
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

Explanation

1. To execute a search, use the `ExecuteReader` method to create a `HiRDBDataReader`.
2. Use the `Read` method to move on to the next row. Use the `GetName` method to acquire a column name, and use the `GetValue` method to acquire a column value.

15.8.5 Executing the INSERT facility using arrays

This example inserts `123`, `200`, and `null` in the `ex` table:

```
// Create objects such as a connection object
HiRDBConnection    pConn = new HiRDBConnection("connection-character-string");
HiRDBCommand    pCom = pConn.CreateCommand();

// Connect to the database
pConn.Open();

// Create a parameter object
HiRDBParameter    pPar = pCom.CreateParameter();

// Set parameters
pPar.Direction = ParameterDirection.Input;
pPar.HiRDBType = HiRDBType.Integer;
object [] aValue = new object[3];
aValue[0] = 123;
aValue[1] = 200;
aValue[2] = null;
pPar.Value = aValue;
pCom.Parameters.Add(pPar);  ........................................1
```

```
// Use parameters to execute SQL statement
pCom.CommandText = "insert into ex values(?)";
pCom.ExecuteNonQuery(aValue.Length);  ...............................2

// Disconnect from the database
pConn.Close();
```

Explanation

1. Set the parameter value in the `value` parameter. Because `value` is the `object` type, it can reference all types. The `Int32` type is specified in normal `INSERT` statements, but in the `INSERT` statement using an array, the array of `object` is set in `value`, and each element of the `object` array is set to point to the `Int32` type. The same applies when other types are used; always set an array of `object` in `value`.
2. To execute the SQL statement, use `:overload` of `ExecuteNonQuery`. The normal `ExecuteNonQuery` has no argument, but when the `INSERT` statement using an array is used, specify the size of the array.

Note

The codes for setting `value` for parameters and for executing SQL statements vary depending on whether or not an array is used.

15.8.6 Executing a repetition column

This example inserts `123`, `456`, and `789` in the first column of the `ex` table:

```
// Create objects such as a connection object
HiRDBConnection    pConn = new HiRDBConnection("connection-character-string");
HiRDBCommand    pCom = pConn.CreateCommand();

// Connect to the database
pConn.Open();

// Create a table
pCom.Connection = pConn;
pCom.CommandText = "create table ex(a int array[3])";
pCom.ExecuteNonQuery();
```

```
// Create a parameter object
HiRDBParameter    pPar = pCom.CreateParameter();

// Set parameters
pPar.Direction = ParameterDirection.Input;
pPar.HiRDBType = HiRDBType.Integer;
object [] aValue = new object[3];
aValue[0] = 123;
aValue[1] = 456;
aValue[2] = 789;    pPar.Value = aValue;
pPar.Repetition = (short)aValue.Length;
pCom.Parameters.Add(pPar);  .......................................1


// Use parameters to execute SQL statement
pCom.CommandText = "insert into ex values(?)";
pCom.ExecuteNonQuery();

// Execute the select statement
pCom.CommandText = "select * from ex";
HiRDBDataReader pReader = pCom.ExecuteReader();


// Fetch until there is no more data
while (pReader.Read())
{
    for (int i = 0; i < pReader.FieldCount; ++ i)
        for (int j = 0; j < pReader.GetFieldArrayCount(i); ++ j)
            Console.WriteLine(pReader.GetValue(i, j));
}  .................................................................2

// Disconnect from the database
pConn.Close();
```

Explanation

1. The `object` array is set in `value` for the same reason as for the `INSERT` statement using an array. For a repetition column, also set the `Repetition` extended property. This property specifies the number of repetition columns. There is no argument during the execution of the SQL statement.
2. For `FETCH`, an extended method for repetition columns is also provided with `DataReader`. First, use `GetFieldArrayCount` to acquire the number of repetition columns for the data obtained by `FETCH`. To acquire the value of the data obtained by `FETCH`, use `:overload` of `GetValue`. In the second argument, specify the number of the repetition column. An indexer `[int,int]` equivalent to this method is also provided.

Note

The usage of repetition columns is similar to that of the `INSERT` facility using arrays. The differences occur in the part that specifies the repetition count in the parameter and the part that executes the SQL statement.

Chapter

16. Type2 JDBC Driver

This chapter explains the JDBC driver installation, environment setup, and JDBC functions. Note that the JDBC driver cannot be used in the Linux for AP8000 version of a client.

Hereafter in this chapter, the Type2 JDBC driver is referred to as *JDBC driver*.

16.1 Installation and environment setup

16.1.1 Installing

You can select the installation of a JDBC driver when installing HiRDB.

Table 16-1 shows the JDBC driver's installation directory and file.

Table 16-1: JDBC driver's installation directory and file

Platform	Type	Installation directory	File
UNIX	HiRDB server	`$PDDIR/client/lib/`	`pdjdbc.jar`[1] `libjjdbc.sl`(`libjjdbc.so`)[2]
	HiRDB client	<u>*/HiRDB*</u>`/client/lib/`	`pdjdbc.jar`[1] `libjjdbc.sl`(`libjjdbc.so`)[2]
Windows	HiRDB server	`%PDDIR%\CLIENT\UTL\`	`pdjdbc.jar` `jjdbc.dll`
	HiRDB client	<u>*\HiRDB*</u>`\CLIENT\UTL\`	`pdjdbc.jar` `jjdbc.dll`

Note

The underline indicates the HiRDB client's installation directory.

[1] For the 32-bit mode HP-UX (IPF) version, the file is `pdjdbc32.jar`.

[2] For the 32-bit mode HP-UX (IPF) version, the file is `libjjdbc32.so`.

To use the JDBC driver in an HP-UX (IPF), Linux (IPF), or Windows Server 2003 (IPF) environment, you need J2SDK v1.4.2. Note that J2SDK v1.4.2 must be run on an IPF-compliant Java Virtual Machine.

16.1.2 Environment setup

The following shows the environment variable definition required for JDBC driver operation.

(1) UNIX environment

Specify the following information in the environment variable for the execution environment:

`CLASSPATH=$CLASSPATH:[`*installation-directory*`]/pdjdbc.jar`*

* For the 32-bit mode HP-UX (IPF) version, the file is `pdjdbc32.jar`. Do not set `pdjdbc.jar` and `pdjdbc32.jar` at the same time.

(2) Windows environment

From **Control Panel**, choose **System**, then in the System Properties dialog box, choose **Advanced**, and specify the following information as the environment variable:
`CLASSPATH=%CLASSPATH%;[`*installation-directory*`]\pdjdbc.jar`

16.1.3 Abbreviation of methods

- The following methods are referred to collectively as the `get`*XXX* method:

  `getArray` method, `getAsciiStream` method, `getBigDecimal` method, `getBinaryStream` method, `getBlob` method, `getBoolean` method, `getByte` method, `getBytes` method, `getCharacterStream` method, `getClob` method, `getDate` method, `getDouble` method, `getFloat` method, `getInt` method, `getLong` method, `getObject` method, `getRef` method, `getShort` method, `getString` method, `getTime` method, and `getTimestamp` method

- The following methods are referred to collectively as the `set`*XXX* method:

  `setArray` method, `setAsciiStream` method, `setBigDecimal` method, `setBinaryStream` method, `setBlob` method, `setBoolean` method, `setByte` method, `setBytes` method, `setCharacterStream` method, `setClob` method, `setDate` method, `setDouble` method, `setFloat` method, `setInt` method, `setLong` method, `setNull` method, `setObject` method, `setRef` method, `setShort` method, `setString` method, `setTime` method, and `setTimestamp` method

16.2 JDBC1.0 facility

16.2.1 Driver class

(1) Overview

The `Driver` class provides the following functions:

- Database connection
- Validity checking on a specified URL
- Acquisition of the connection properties specified with the `DriverManager.getConnection` method
- Acquisition of driver version information

For details about and usage of each method provided with the `Driver` class, see the applicable JDBC manual. This section explains the database connection procedure and the URL syntax unique to this JDBC driver.

(2) Database connection using the DriverManager

To execute DB connection using the `DriverManager` class provided by the Java execution environment:

1. Register the `Driver` class in the Java Virtual Machine.
2. Call the `DriverManager.getConnection` method using the connection information as the argument.

(a) Registering in Java Virtual Machine with the Driver class

Register the `Driver` class in the Java Virtual Machine by using the `Class.forName` method or by registering in the system properties. The package name and `Driver` class name of the JDBC driver specified for registration are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`Driver` class name: `PrdbDriver`

- **Using the Class.forName method**

  Call the `Class.forName` method from within the application as follows:

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

- **Registering in the system properties**

  Call the `System.setProperty` method from within the application as follows:

```
System.setProperty("jdbc.drivers","JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver");
```

(b) Defining the connection information and establishing a database connection

To connect to the database, use one of the following methods:

- **Using the DriverManager.getConnection method**

```
Connection con = DriverManager.getConnection(String url, String user, String password)
;
or
Connection con = DriverManager.getConnection(String url, Properties info) ;
```

- **Specification with an internal driver**

  When an internal driver is used, the information called by the routine at the HiRDB side is assumed as the connection information (such as the authorization identifier). However, when a trace is acquired within the JDBC driver, `INNER` is assumed as the authorization identifier.

```
Specification for internal driver only:
Connection con = DriverManager.getConnection(String url) ;
```

- **Directly calling the connect method in the Driver class**

```
Driver drv = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDriver();
Connection con = drv.connect(String url, Properties info) ;
```

In the arguments of the previous methods, specify the information required for database connection.

If a database connection is successful, the JDBC driver returns a `Connection` object as a result of the method call. If required information is not specified in each argument, or invalid information is specified, the JDBC driver throws an `SQLException` as a result of the method call.

Table 16-2 lists the arguments of the `getConnection` method, and Table 16-3 lists the information to be specified for `Properties info`.

Table 16-2: Arguments of the getConnection method

Argument	Description	Specification
`String url`	URL; For URL, see *(3) URL syntax*.	R
`String user`	Authorization identifier[1]	R[2]
`String password`	Password	O
`Properties info`	See *Table 16-3*.	—

Legend:

R: Required.

O: Optional.

— : Not applicable.

[1] If null or space characters are specified for the authorization identifier, this method throws an `SQLException`. The method also throws an `SQLException` if the driver-converted character codes and, as a result, the size of the character string specified for the authorization identifier exceed 30 bytes. For details about character code conversion, see *16.12.2 Character code conversion facility*.

[2] The argument can be omitted, if specified with the internal driver.

Table 16-3: Information to be specified for Properties info

Key	Description	Specification
`user`	Authorization identifier[1]	R[2]
`password`	Password	O
`ENCODELANG`	In a Java program, Unicode is used for the character codes. Therefore, during character data processing with HiRDB, the JDBC driver performs mutual character code conversion between HiRDB's character data and Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the Java Virtual Machine. You must specify the character set names specified by the JDBC driver for the provided encoder and decoder. The settings can be for any character set (such as MS932) supported by Java. For details about this operation if you specify OFF or have not specified anything in `Properties info` (including the settings using the `DataSource.setEncodeLang` method and `ENCODELANG` of the URL), see *16.11.5 setEncodeLang*.	O
`COMMIT_BEHAVIOR`	When HiRDB commits, this key specifies whether or not the following classes are to remain valid after commit has executed: • `ResultSet` class • `Statement` class, `PreparedStatement` class, and `CallableStatement` class For details about the specification values, see *16.11.19 setCommit_Behavior*. Note: See *Notes on COMMIT_BEHAVIOR* following this table.	O

Key	Description	Specification
BLOCK_UPDATE	Specifies whether or not multiple parameters are to be processed at one time when the ? parameter is used to update databases. When this information is omitted, FALSE is assumed. TRUE: Processes multiple parameters at one time. FALSE: Processes parameter sets individually. Other: Assumes that FALSE is specified. Notes: • When TRUE is set, the batch update function supports HiRDB facilities using arrays. • Only INSERT, UPDATE, and DELETE SQL statements can use facilities using arrays. All other SQL statements are processed sequentially, not in batch mode. • Even the SQL statements that can use facilities using arrays are processed sequentially, not in batch mode, if they do not satisfy the conditions for facilities using arrays. • To use facilities using arrays, see *16.3.2 Batch updating*. • For details about the facilities using arrays, see *4.8 Facilities using arrays*. • This function can also be specified using the HiRDB_for_Java_BLOCK_UPDATE system property. However, when BLOCK_UPDATE is set, the HiRDB_for_Java_BLOCK_UPDATE system property setting is ignored.	O
LONGVARBINARY_ ACCESS	Specifies the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY). When this key is omitted, REAL is assumed. REAL: Accesses real data from HiRDB. LOCATOR: Uses the HiRDB locator. Other: Assumes that REAL is specified.	O

Key	Description	Specification
HiRDB_for_Java_SQL_IN_NUM	Specifies the maximum number of input or input/output ? parameters in the SQL statements to be executed. This is the number of input or input/output ? parameter information items that is acquired during SQL preprocessing. If the actual number of input or input/output ? parameters is greater than this property value, the input or input/output ? parameter information is acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or non-numeric value results in an error. Notes: • This item can also be specified using the HiRDB_for_Java_SQL_IN_NUM system property. However, when HiRDB_for_Java_SQL_IN_NUM is specified for Properties info, the system property setting is ignored. • If you do not execute any SQL statement that uses input or input/output ? parameters, we recommend that you specify a value of 1. • This property value is applicable only when the version of the connected HiRDB server is 07-02 or later.	O
HiRDB_for_Java_SQL_OUT_NUM	Specifies the maximum number of output items for the SQL statement to be executed. This is the number of output items that is acquired during SQL preprocessing. If the actual number of output items is greater than this property value, the output items are acquired after the SQL preprocessing. The permitted value range is from 1 to 30,000 (default is 64). Specifying any other value or non-numeric value results in an error. Note: • This item can also be specified using the HiRDB_for_Java_SQL_OUT_NUM system property. However, when HiRDB_for_Java_SQL_OUT_NUM is specified for Properties info, the system property setting is ignored. • If you do not execute any SQL statement that contains a search item or output or input/output ? parameter, we recommend that you specify a value of 1. • This property value is applicable only when the version of the connected HiRDB server is 07-02 or later.	O
HiRDB_for_Java_SQLWARNING_LEVEL	Specifies the retention level of warning information that has been issued during execution of the SQL statement. The permitted warning retention levels are as follows: • IGNORE • SQLWARN (default) • ALLWARN In this method, information specified in the arguments is not case sensitive. For details about the above values, see *16.2.9 SQLWarning class*.	O

Key	Description	Specification
HiRDB_for_Java _CLEAR_ENV	Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection. TRUE: Ignores the HiRDB client environment definition registered as OS environment variables when the database is connected for the first time after the process has started. When TRUE is specified, you can apply the value of the HiRDB client environment definition that has been set by a method other than the OS environment variables (such as environment variable groups). FALSE (default): Does not ignore the HiRDB client environment definition registered as OS environment variables. Notes: • In this method, information specified in the arguments is not case sensitive. • Once the database is connected, the HiRDB client environment definition set as OS environment variables is not ignored even if an attempt is made to specify TRUE within a native method installed by a method such as C language. • Once the database is connected with TRUE specified, the client environment definition value remains ignored even if FALSE is specified the next time the database is connected.	O

Legend:

R: Required.

O: Optional.

[1] If null or space characters are specified for the authorization identifier, this method throws an SQLException. This method also throws an SQLException if the driver-converted character codes and, as a result, the size of the character string specified for the authorization identifier exceed 30 bytes. For details about character code conversion, see *16.12.2 Character code conversion facility*.

[2] The key can be omitted, if specified with the internal driver.

Notes on COMMIT_BEHAVIOR

- If another user specifies CLOSE or PRESERVE to execute a definition SQL on a resource (such as a table or index) that is being accessed by SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL, and PDDDLDEAPRP in the client environment definition is set to NO, the definition SQL goes into lock-release wait status until the connection to the resource is disconnected.

  If PDDDLDEAPRP in the client environment definition is set to YES, the preprocessing result becomes invalid. If an SQL for which the preprocessing result has been invalidated in this manner is executed, an `SQLException` exception occurs (the value acquired by the `getErrorCode` method is `-1542`).

- When `PRESERVE` is specified, the JDBC driver uses HiRDB's holdable cursor.

- By specifying[1] `CLOSE` or `PRESERVE`, the only precompiled SQL statements that are valid after commit[2] are `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `PURGE TABLE`, and `CALL` (SQL statements can be precompiled by executing the `Connection.prepareStatement` method or the `Connection.prepareCall` method).

  Other precompiled SQL statements become invalid during commit even though you specify `CLOSE` or `PRESERVE` for `COMMIT_BEHAVIOR`.

  When SQL statements that include these invalid SQL statements are executed with the `PreparedStatement` class object or `CallableStatement` object, an error occurs. An example of such an error is shown below:

  Example

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();         //No error occurs.
con.commit();
pstmt2.execute();         //An error occurs.
pstmt1.close();
pstmt2.close();
```

Explanation

Because the SQL statement to be executed is a `LOCK` statement, even though `COMMIT_BEHAVIOR` specifies `CLOSE`, `PreparedStatement` becomes invalid after commit and an error occurs.

[1] Refers to specification of one of the following:

- `COMMIT_BEHAVIOR=CLOSE` specified for the URL specified by the `getConnection` method.

- `COMMIT_BEHAVIOR=PRESERVE` specified for the URL specified by the `getConnection` method.

- `setCommit_Behavior` method of the `JdbhDataSource`, `JdbhConnectionPoolDataSource`, or `JdbhXADataSource` class used to specify `CLOSE`.

- `setCommit_Behavior` method of the `JdbhDataSource`, `JdbhConnectionPoolDataSource`, or `JdbhXADataSource` class used to specify `PRESERVE`.

[2] Means one of the following:

- Explicit commit using the commit method
- Implicit commit by automatic commit
- Execution of a definition SQL statement
- Execution of a `PURGE TABLE` statement
- Explicit rollback by rollback method
- Implicit rollback by an SQL execution error

(3) URL syntax

This section explains the URL syntax supported by the JDBC driver. Do not place any space inside each item or between items in a URL. To specify both an additional connection information item and a database host name item, separate them by a comma (`,`).

(a) URL syntax

```
jdbc:hitachi:PrdbDrive[://[DBID=additional-connection-information]
                      [[{://|,}]DBHOST=database-host-name]
                      [[{://|,}]ENCODELANG=conversion-character-set]
                      [[{://|,}]COMMIT_BEHAVIOR=cursor-operation-mode]
                      [[{://|,}]CLEAR_ENV=environment-variable-invalidation-setting]]
```

(b) URL items

`jdbc:hitachi:PrdbDrive`

This is the protocol name and the subprotocol name. This item is required.

additional-connection-information

Specify HiRDB's port number (this corresponds to `PDNAMEPORT` in the client definitions). Alternatively, specify a HiRDB environment variable group.

If this item is omitted, the default value for `PDNAMEPORT` is assumed.

Notes about specifying a HiRDB environment variable group in additional connection information

- When you specify the name of a HiRDB environment variable group, place @ at the beginning of the group name.

- If the environment variable name contains single-byte spaces or single-byte @ characters, enclose the name in single-byte quotation marks ("). When an environment variable group name is enclosed in single-byte quotation marks, all characters following the last single-byte quotation mark up to the next item or all characters through the end of the character string are ignored. An environment variable group name containing single-byte quotation marks or single-byte commas cannot be specified.
- The environment variables registered in an environment variable group have precedence over the user environment variables and the environment variables registered by HiRDB.INI.
- The following priority applies to the specification of additional connection information and database host name:

  1. HiRDB environment variable group specified in the additional connection information

  2. Database host name or the port number specified in the additional connection information

  For example, if a HiRDB environment variable group name has been specified in DBID, information about the HiRDB environment variable group takes effect. A database host name does not take effect even if it is specified in DBHOST in the URL. In this case, if PDHOST is omitted in the HiRDB environment variable group, a connection error results.

database-host-name

Specify HiRDB's host name. This corresponds to PDHOST in the client definitions.

If this item is omitted, the default value for PDHOST is assumed.

conversion-character-set

Specify the conversion character set to be used for character type conversion.

cursor-operation-mode

Specify whether the cursor is valid following COMMIT.

environment-variable-invalidation-setting

Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection. For details about the specification value and notes, see HiRDB_for_Java_CLEAR_ENV in Table 16-3.

(c) Example of specifying a HiRDB environment variable group name in additional connection information

- In UNIX

In this example, the path of the HiRDB environment variable group name is /HiRDB_P/Client/HiRDB.ini:

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini";
```

- In Windows

1. In this example, the environment variable group name registered using the tool for registering HiRDB client environment variables is HiRDB_ENV_GROUP:

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=HiRDB_ENV_GROUP";
```

2. In this example, the path of the HiRDB environment variable group name is C:\HiRDB_P\Client\HiRDB.ini:

```
String url = "jdbc:hitachi:PrdbDrive://
DBID=@HIRDBENVGRP=C:\\HiRDB_P\\Client\\HiRDB.ini";
```

3. In this example, the path of the HiRDB environment variable group name is C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini (∆: single-byte space character):

```
String url = "jdbc:hitachi:PrdbDrive://DBID=@HIRDBENVGRP=" +
            "\"C:\Program∆Files\\HITACHI\\HiRDB\\HiRDB.ini\"";
```

16.2.2 Connection class

(1) Overview

The Connection class provides the following functions:

- Creation of objects in the Statement, PreparedStatement, and CallableStatement classes
- Transaction settlement (COMMIT or ROLLBACK)
- Specification of AUTO commit mode

For details about and usage of each method provided with the Connection class, see the applicable JDBC manual.

(2) Notes

(a) Catalog

The JDBC driver does not use a catalog regardless of the connected database type. Therefore, the `getCatalog` method unconditionally returns the `null` value, and the `setCatalog` method does nothing.

(b) Access mode

The JDBC driver does not allow the access mode to be changed. Therefore, the `isReadOnly` method unconditionally returns `false`, and the `setReadOnly` method processes nothing.

(c) Transaction isolation mode

The JDBC driver does not allow the transaction mode to be changed. Therefore, the `getTransactionIsolation` method unconditionally returns `TRANSACTION_READ_COMMITTED`, and the `setTransactionIsolation` method does nothing.

16.2.3 Statement class

(1) Overview

The `Statement` class provides the following functions:

- SQL execution
- Creation of a result set (`ResultSet` object) as a retrieval result
- Return of the number of updated rows as an updating result

For details about and usage of each method provided with the `Statement` class, see the applicable JDBC manual.

(2) Notes

(a) Multi-thread

To use a single `Statement` object with multiple threads, a series of processing, such as SQL execution, acquisition of result set, and closing of the result set, needs to be serialized per thread. If they are processed in parallel, operation cannot be guaranteed. Therefore, you should allocate a separate `Statement` object for each thread.

(b) Cursor name

The JDBC driver does not support positioned updating or deletion. Therefore, the `setCursorName` method does nothing.

(c) Limitation of retrieval time

The JDBC driver does not support the monitoring of a retrieval time. Therefore, the `setQueryTimeout` method, if specified, is ignored.

(d) Specification of the maximum number of rows to be retrieved

The maximum number of rows to be retrieved cannot be specified in the JDBC driver.

16.2.4 PreparedStatement class

(1) Overview

The `PreparedStatement` class provides the following functions:

- Execution of SQL specifying the `?` parameter
- Specification of the `?` parameter
- Generation and return of the `ResultSet` object as a search result
- Return of the number of updated rows as an updating result

Because the `PreparedStatement` class is a subclass of the `Statement` class, it inherits all of the `Statement` class functions.

For details about and usage of each method provided with the `PreparedStatement` class, see the applicable JDBC manual.

(2) Notes

All of the notes about the `Statement` class are applicable to the `PreparedStatement` class, because the `PreparedStatement` class is a subclass of the `Statement` class. The following describes the other notes about the `PreparedStatement` class.

(a) Specification of the ? parameter

For details about the `set`*XXX* method used to set the `?` parameter, see *16.3.3(2) Data mapping when the ? parameter is specified*. For details about the JDBC SQL types supported by the connected database, see *16.12 Data types and character codes*.

(b) Multiple result sets

The function for returning multiple result sets is not available. Therefore, the `getMoreResults` method unconditionally returns `false` and closes any currently open result set.

16.2.5 CallableStatement class

(1) Overview

The `CallableStatement` class provides the following functions:

- Execution of Java stored routines

- Specification of `IN` and `INOUT` parameters (a `set`*XXX* method of the `PreparedStatement` class is used)
- Registration of `OUT` and `INOUT` parameters
- Acquisition of `OUT` and `INOUT` parameters
- Acquisition of a result set

Because the `CallableStatement` class is a subclass of the `PreparedStatement` class, it inherits all of the `PreparedStatement` class functions and the `Statement` class functions. Note that the result set obtained with the `DatabaseMetaData` class within a Java stored routine can be used only within the Java stored routine. The `CallableStatement` class's `getResultSet` cannot acquire it as a dynamic result set.

For details about and usage of each method provided with the `CallableStatement` class, see the applicable JDBC manual.

(2) Notes

1. All of the notes about the `PreparedStatement` and `Statement` classes applicable to the `CallableStatement` class, because the `CallableStatement` class is a subclass of the `PreparedStatement` class.
2. Parameter information is erased when the `clearParameters` method is executed. If the `clearParameters` method is executed after execution of the `execute` method but before execution of the `get`*XXX* method, the `KFPJ20506-E` message is output when the `get`*XXX* method is executed.
3. If you use the `INOUT` parameter of a Java stored routine, the `java.sql.Types` specified using the `registerOutParameter` method and the data type set using the `set`*XXX* method must be the same.

16.2.6 ResultSet class

(1) Overview

The `ResultSet` class provides the following functions:

- Moving within a result set in units of rows
- Returning resulting data
- Issuing a message indicating whether or not the retrieval result data is the `NULL` value

For details about and usage of each method provided with the `ResultSet` class, see the applicable JDBC manual.

(2) Notes

(a) Multi-thread

If a single `ResultSet` object is used with multiple threads in parallel, operation cannot be guaranteed. Therefore, you should process a single `ResultSet` object by a single thread.

(b) Data mapping (conversion)

For details about the `get`*XXX* method used during the acquisition of results, see *16.3.3(1) Data mapping during retrieval data acquisition*. For details about the JDBC SQL types supported by the connected database, see *16.12 Data types and character codes*.

16.2.7 ResultSetMetaData class

(1) Overview

The `ResultSetMetaData` class provides the following functions:

- Returning meta-information, such as data type and length of each column, in `ResultSet` (result set)

(2) Details of method

(a) isSearchable (int column) method

`true` is returned if the column specified by the `column` parameter can be used for the `WHERE` clause; otherwise, `false` is returned as the return value. In the case of the `WHERE` clause, `true` is always returned to enable use of all data type columns. However, for the first column of `ResultSet`, which is the return value of the `Array.getResultSet` method, `false` is returned. For details about `getResultSet`, see *16.6 Array class*.

Example:

Column `C1` is in table `T1`. Regardless of its data type, `C1` can be used in the `WHERE` clause as shown below:

```
SELECT * FROM T1 WHERE LENGTH (C1) > 5
```

(b) getColumnDisplaySize (int column) method

The return value is the maximum number of characters when the column specified by the `column` parameter is expressed in a character string. However, for the first column of `ResultSet`, which is the return value of the `Array.getResultSet` method, 10 is returned. Table 16-4 lists the return values for this method for each SQL data type in HiRDB.

Table 16-4: Return values of the getColumnDisplaySize method for each SQL data type in HiRDB

SQL data type in HiRDB	Return value (int)	Return value format
`INTEGER`	11	1 sign character + 10 digits, which is the maximum number of digits
`SMALLINT`	6	1 sign character + 5 digits, which is the maximum number of digits

`DECIMAL(`*m*`,`*n*`)`
`NUMERIC(`*m*`,`*n*`)`
• *m*: Accuracy (total number of digits)
• *n*: Decimal scaling position (number of digits after decimal point)
	m + 2	1 sign character + accuracy m + 1 decimal point digit

`FLOAT` `DOUBLE PRECISION`	23	1 sign character + 17 digits, which is the maximum number of significant digits + 1 decimal point character + 4, which is the maximum number of characters in the index area
`SMALLFLT` `REAL`	13	1 sign character + 8 digits, which is the maximum number of significant digits + 1 decimal point character + 3, which is the maximum number of characters in the index area
`CHAR(`*n*`)` • *n*: Number of bytes of the definition length	*n*	NA

`VARCHAR(`*n*`)`
`CHAR VARYING(`*n*`)`
• *n*: Number of bytes of the maximum length
	n	NA

`NCHAR(`*n*`)`
`NATIONAL CHAR(`*n*`)`
• *n*: Number of characters of the definition length
	n	NA

SQL data type in HiRDB	Return value (int)	Return value format
NVARCHAR(*n*) NATIONAL CHAR VARYING(*n*) NCHAR VARYING(*n*) • *n*: Number of characters of the maximum length	*n*	NA
MCHAR(*n*) • *n*: Number of bytes of the maximum length	*n*	NA
MVARCHAR(*n*) • *n*: Number of bytes of the maximum length	*n*	NA
DATE	10	*yyyy*-*mm*-*dd*, which is 10 characters
TIME	8	*hh*:*mm*:*ss*, which is 8 characters
TIMESTAMP(*p*) • *p*: Number of digits of the fractional part of the second	(1) When *p* is 0: 19 (2) When *p* is 2, 4, or 6: 20 + *p*	(1) *yyyy*-*mm*-*dd* *hh*:*mm*:*ss*, which is 19 characters (2) 19 characters shown above + 1 decimal point character + *p*, which is the number of digits of the decimal part
BLOB(*n*[*K*|*M*|*G*]) • *n*: Maximum length • *K*: Kilobytes • *M*: Megabytes • *G*: Gigabytes If the unit is omitted, bytes is assumed.	When the unit specification is omitted: *n* When *K* is specified for the unit: *n* x 1024* When *M* is specified for the unit: *n* x 1024 x 1024* When *M* is specified for the unit: *n* x 1024 x 1024 x 1024#	NA
BINARY(*n*) • *n*: Number of bytes of the maximum length.	N	NA

Legend:

NA: Not applicable.

* Calculation result 2147483648 becomes 2147483647.

16.2.8 DatabaseMetaData class

The `DatabaseMetaData` class provides the following functions:

- Returning various information about a connected database
- Storing or returning listing information (such as a list of tables or columns) in `ResultSet` (result set)

Note that the result set obtained with the `DatabaseMetaData` class within a Java stored routine can be used only within the Java stored routine.

For details about the methods provided by the `DatabaseMetaData` class and how to use them, see the applicable JDBC documentation. For details about the values that are actually returned, see *16.13 Classes and methods with limitations*. Note that the value returned by each method is information related to the HiRDB server, whose version has to be the same as that of the JDBC driver being used.

16.2.9 SQLWarning class

(1) Overview

The `SQLWarning` class provides the following function:

- Providing information about database access warnings

The `SQLWarning` object is accumulated without an issuance of exception in the method object that caused the warning.

(2) Notes

(a) Releasing the accumulated SQLWarning object

The `SQLWarning` object is accumulated by chain from the method object that caused the warning (`Connection`, `Statement`, `PreparedStatement`, `CallableStatement`, or `ResultSet`).

To explicitly release the accumulated `SQLWarning` object, you must execute `clearWarnings` from the object connecting the chain.

(b) SQLWarning object generation conditions

If the warnings caused by execution of SQL statements are to be retained in the JDBC driver according to the warning retention level specification, the `SQLWarning` objects are generated and the warning information is retained. The following table describes the `SQLWarning` generation conditions:

Execution result of SQL statement	Warning retention level		
	IGNORE	SQLWARN	ALLWARN
`SQLCODE` is greater than 0 and is not 100, nor 110, nor 120	N	N	Y

Execution result of SQL statement	Warning retention level		
	IGNORE	SQLWARN	ALLWARN
SQLWARN0 in the SQL Communications Area is W (except when SQLWARN6 is W)	N	Y	Y
Warning occurred in the JDBC driver	N	Y	Y

Legend:

Y: Generated

N: Not generated

Note

You can specify the warning retention level using the HiRDB_for_Java_SQLWARNING_LEVEL property or the setSQLWarningLevel method. The default is SQLWARN.

(c) Warning message

The following table presents the messages that can be acquired from SQLWarning:

Condition	Message acquired by getMessage
SQLWARN0 is W	KFPJ01074-W
SQLWARN0 is '∆' and SQLCODE is greater than 0 (except when SQLCODE=100, 110, or 120)	KFPA*XXXXX*-X
Warning occurred in the JDBC driver	KFPJ*XXXXX*-W

(d) Batch updating

When warning occurs during updating of multiple rows during batch updating, only one SQLWarning is generated.

16.3 JDBC2.0 basic facility

16.3.1 Result set enhancements

The JDBC2.0 basic standard has added scroll and parallel processing as the extended features of result sets (`ResultSet` class).

(1) Scroll types

There are three different scroll types for result sets:

(a) Forward-only type

This is the standard scroll type from JDBC1.0. It allows a result set to be scrolled in the forward direction only (from top to bottom).

(b) Scroll-insensitive type

This is a new scroll type added with JDBC2.0. It allows a result set to be scrolled in a forward or backward direction. It also allows a movement specifying a location relative to the current location or a movement to an absolute location.

Scroll-insensitive means that a change made while a result set is open does not take effect on the result set. In other words, the scroll-insensitive type provides a static view of base data. The rows contained in a result set, their order, and column values are all fixed when the result set is created.

(c) Scroll-sensitive type

This is a new scroll type added with JDBC2.0. While a result set is open, any change made takes effect on the result set.

Changes that take effect may be made directly to the current result set, or made by another result set within the same transaction, or made by another transaction. The number of changes applied depends on the driver's implementation level and DBMS transaction cut-off level.

(2) Parallel processing type

There are two different parallel processing types for result sets:

(a) Read-only type

This is the standard parallel processing type from JDBC1.0. It does not allow data to be updated from its result set.

(b) Updatable type

This is a new parallel processing type added with JDBC2.0. It allows data to be updated (`UPDATE`, `INSERT`, and `DELETE`) from its result set.

(3) Types of result set

When the scroll type and parallel processing type are combined, there are six result set types. Specify the result set type to acquire an instance of the `Statement` class (or its subclass) using the `createStatement` method, `prepareStatement` method, or `prepareCall` method of the `Connection` class.

Table 16-5 shows the availability of the result set type when you use the JDBC driver.

Table 16-5: Availability of result set types with JDBC driver

Result set type		Availability with JDBC driver
Scroll type	**Parallel processing type**	
Forward-only	Read-only	A
	Updatable	NA
Scroll-insensitive	Read-only	A
	Updatable	NA
Scroll-sensitive	Read-only	NA
	Updatable	NA

Legend:

A: Available.

NA: Not available.

Notes

1. An error occurs if an unavailable result set is specified. In this case, the JDBC driver creates an instance of the `Statement` class (or its subclass) using the result set that is closest to the specified type, then stores a warning message in `SQLWarning` of the `Connection` class.
2. Some of the methods in the `ResultSet` class are not available because the JDBC driver does not provide the updatable parallel processing type. If such an unavailable method is called, the JDBC driver unconditionally throws an `SQLException`. For details about the unavailable methods, see *16.13 Classes and methods with limitations*.

(4) Notes about using scroll-type result sets

A scroll-type result set caches all retrieval data in the JDBC driver. If there is a large amount of data, a memory shortage or performance reduction may occur. Therefore, to use a scroll-type result set, you must suppress the retrieval data volume in advance by adding a condition to an SQL statement, for example.

16.3.2 Batch updating

The JDBC2.0 basic standard adds the batch updating feature to the `Statement`, `PreparedStatement`, and `CallableStatement` classes. The batch update facility enables multiple SQL statements or multiple parameter values to be registered for batch execution.

To use the batch update facility, you need to set the `Connection` class's `AUTO` commit mode to off. This is because, if an error occurs during the batch updating, the application needs to control the transaction's validity. If the `AUTO` commit mode is on (initial status) and an error occurs during the batch updating, the SQL execution immediately preceding the error takes effect.

When you execute batch updating, you can use HiRDB facilities using arrays.

The facilities using arrays are useful for updating a large amount of HiRDB data at high speed. For details about the facilities using arrays, see *4.8 Facilities using arrays*.

Notes about using the facilities using arrays

1. The facilities using arrays are supported by HiRDB version 07-01 or later.
2. During `Connect`, you must specify the `BLOCK_UPDATE=TRUE` property (if `DataSource` is used, specify `setBlockUpdate(true)`) or `setBlockUpdate(true)` in `JdbcDbpsvPreparedStatement`.
3. If you specify the `HiRDB_for_Java_BLOCK_UPDATE=TRUE` system property, you can enable the array facilities. For details about `HiRDB_for_Java_BLOCK_UPDATE`, see *BLOCK_UPDATE* in Table 16-3.
4. The SQL statement to be executed must contain at least one `?` parameter (this does not apply to stored procedures). Additionally, you must use the `addBatch()` method of the `CallableStatement` class or the `PreparedStatement` class (using the `addBatch(String sql)` method of the `Statement` class results in a HiRDB error).

   Executable SQL statements include `INSERT`, `UPDATE`, and `DELETE`. All other SQL statements are executed sequentially, not in batch mode.
5. There must be two or more parameter sets that have been registered by the `addBatch()` method. If there is only one parameter set, it is processed normally, not in batch mode. If there are more than 30,000 parameter sets, each group of 30,000 parameter sets is executed at one time.
6. If the length of `BINARY` data specified in the `?` parameter is 32,001 bytes or greater, sequential execution takes place because facilities using arrays are not applied.
7. If the length of data specified for HiRDB `BLOB`-type columns is 32,001 bytes or greater, sequential execution takes place because facilities using arrays are

not applied.[2]

8. Make sure that the same data type is specified for each and every column.[1]
9. When `DECIMAL`-type data is inserted, the precision and scaling of the `DECIMAL`-type data specified for array are replaced by HiRDB's table definition attributes. If the length of integer part of the `DECIMAL`-type data specified for array is greater than that of the HiRDB table definition attribute, an overflow occurs, resulting in an error.
10. If you specify HiRDB's repetition column in the `?` parameter, you cannot use the facilities using arrays.
11. If an error occurs during batch updating with facilities using arrays, the execution results immediately preceding the error are ignored.
12. Facilities using arrays cannot be used from the basic Cosminexus J2EE server mode.
13. When facilities using arrays are used from Cosminexus, the `setBlockUpdate` method of `PreparedStatement` is not available.
14. When a large amount of data is updated using the `addBatch` function, a large amount of Java memory is used. Depending on the performance of Java memory, the advantages of batch updating may not be obtained. When you use a large amount of data, specify a heap size at the start of Java (`java -Xms32m JavaUP`: set the Java heap at the start of Java to 32 megabytes).

[1] For example, if you use `setInt()` to specify the first `addBatch` for the column 1 data, you must also use `setInt()` for the subsequent `addBatch`.

[2] If you use facilities using arrays and specify the `?` parameter for HiRDB's `BLOB`-type columns, note the following:

- If the length of data specified in the `?` parameter is less than 32,001 bytes, the data is treated as `BINARY`-type data in the JDBC driver, thereby executing facilities using arrays. If the length is 32,001 bytes or greater, facilities using arrays are not executed.

(1) Batch updating with the Statement class

Following are notes about batch updating with the `Statement` class:

- Use the `addBatch` method to register multiple updating SQL statements.
- Use the `executeBatch` method to execute the registered updating SQL statements in batch mode.
- An array of the number of rows updated by each updating SQL statement is returned as the batch execution result.

- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If a retrieval SQL statement is registered, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.

The JDBC driver executes registered SQL statements sequentially because it cannot execute them in batch mode.

(2) Batch updating with the PreparedStatement class

Following are notes about batch updating with the `PreparedStatement` class:

- Use a normal procedure (`set`*XXX* method) to specify the `?` parameter for an updating SQL statement that is specified during the creation of a `PreparedStatement` instance.
- Use the `addBatch` method to register `?` parameter sets.
- Use the `executeBatch` method to execute the registered multiple`?` parameter sets in batch mode.
- An array of the number of rows updated by each `?` parameter set is returned as the batch execution result.
- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If a retrieval SQL statement is specified during the creation of a `PreparedStatement` instance, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.

When facilities using arrays are used, the JDBC driver can execute multiple lines of `?` parameters in batch mode. When facilities using arrays are not used, multiple lines of `?` parameters are executed sequentially.

Notes

- If you use HiRDB facilities using arrays, see the notes in *16.3.2 Batch updating*.
- In the second or subsequent `addBatch`, if there are not enough parameters to be specified in the `set`*XXX* method, the previous values are inherited. The following shows an example.

Example: When there are 2 `INTEGER`-type columns (columns 1 and 2)

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
```

```
prepstmt.executeBatch();
```

Explanation

- The values that are set in the first `addBatch` are `100` for both columns 1 and 2.

  If there are not enough parameters in the first `addBatch`, an error occurs.

- The values that are set in the second `addBatch` are `200` for column 1 and `100` for column 2.

  Because information for column 2 has not been updated by the second `addBatch`, information for the first `addBatch` is inherited.

(3) Batch updating with the CallableStatement class

Following are notes about batch updating with the `CallableStatement` class:

- Use a normal procedure (`set`*XXX* method) to specify input parameters for the Java stored routine that is specified during the creation of a `CallableStatement` instance.
- Use the `addBatch` method to register input parameter sets.
- Use the `executeBatch` method to execute the registered multiple input parameter sets in batch mode.
- An array of the values (number of updated rows) that are returned by the Java stored routine executed by each input parameter set is returned as the batch execution result.
- If an error occurs during batch execution, the JDBC driver throws a `BatchUpdateException`.
- If the Java stored routine specified during the creation of a `CallableStatement` instance does not return the number of updated rows, the JDBC driver throws a `BatchUpdateException` when calling the `executeBatch` method.
- If the Java stored routine specified during the creation of a `CallableStatement` instance has an output parameter or input/output parameter, the JDBC driver throws a `BatchUpdateException` when calling the `addBatch` method.

The JDBC driver cannot execute multiple lines of `?` parameters in stored procedures; therefore, multiple lines of `?` parameters in stored procedures are executed sequentially.

Notes

- Batch updating of stored procedures is supported only for the `IN` parameter. If an `OUT` parameter, `INOUT` parameter, or result set (`ResultSet`) is used, an error results.

- In the case of a stored procedure that returns a result set (`ResultSet`), whether or not it returns a result set is unknown until the stored procedure is executed during batch updating. Therefore, if data is updated within the stored procedure, updated information may be applied.[1]
- Facilities using arrays are not supported in stored procedures. They are supported only in the SQL statements with `?` parameters.
- In the second or subsequent `addBatch`, if there are not enough parameters to be specified in the `set`*XXX* method, the previous values are inherited.[2]
- If you use the facilities using arrays, see the notes in *16.3.2 Batch updating*.

[1] For example, if a stored procedure that searches and acquires the result of updating is executed during batch updating, `BatchUpdateException` occurs, but updated information may still be applied.

[2] Example: When there are 2 `INTEGER`-type columns (columns 1 and 2)

```
callstmt.setInt(1,100);
callstmt.setInt(2,100);
callstmt.addBatch();
callstmt.setInt(1,200);
callstmt.addBatch();
callstmt.executeBatch();
```

Explanation

- The values that are set in the first `addBatch` are `100` for both columns 1 and 2.

  If there are not enough parameters in the first `addBatch`, an error occurs.
- The values that are set in the second `addBatch` are `200` for column 1 and `100` for column 2.

  Because information for column 2 has not been updated by the second `addBatch`, information for the first `addBatch` is inherited.

16.3.3 Added data types

Several new JDBC SQL types have been added to JDBC2.0 basic standard. They are as follows:

- `BLOB`
- `CLOB`

- `ARRAY`
- `REF`
- `DISTINCT`
- `STRUCT`
- `JAVA OBJECT`

Note that the JDBC driver can use only the `ARRAY` JDBC SQL type.

(1) Data mapping during retrieval data acquisition

Tables 16-6 and 16-7 show the mapping between the `get`*XXX* methods and JDBC SQL types of `ResultSet` and `CallableStatement`.

If a `get`*XXX* method is called for an unsupported JDBC SQL type, the JDBC driver throws an `SQLException`. For details about the JDBC SQL types supported by the connected database, see *16.12 Data types and character codes*.

Note that the `getCharacterStream` method has been added because the `getUnicodeStream` method is no longer recommended in the JDBC2.0 basic standard.

Table 16-6: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (1/2)

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
`getByte`	M	M	M	M	M	M[2]
`getShort`	R	M	M	M	M	M[2]
`getInt`	M	R	M	M	M	M[2]
`getLong`	M	M	M	M	M	M[2]
`getFloat`	M	M	M	R	M	M[2]
`getDouble`	M	M	R	M	M	M[2]
`getBigDecimal`	M	M	M	M	R	M[2]
`getBoolean`	M	M	M	M	M	M
`getString`	M	M	M	M	M	R
`getBytes`	—	—	—	—	—	—
`getDate`	—	—	—	—	—	M[2]

getXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
getTime	—	—	—	—	—	M[2]
getTimestamp	—	—	—	—	—	M[2]
getAsciiStream	—	—	—	—	—	M
getUnicodeStream	—	—	—	—	—	M
getBinaryStream	—	—	—	—	—	—
getObject	M	M	M	M	M	M
getCharacterStream	—	—	—	—	—	M
getArray	—	—	—	—	—	—
getBlob	—	—	—	—	—	—
getClob[1]	—	—	—	—	—	—
getRef[1]	—	—	—	—	—	—

Legend:

R: Mapping is recommended.

M: Can be mapped.

—: Cannot be mapped.

[1] Not supported by the JDBC driver.

[2] When the data from the character string is converted, any single-byte spaces that precede or follow the character string data acquired from the database are stripped, before the data is converted to the Java data type that is to be returned by the `get`*XXX* method.

Note the following when you convert data to a Java data type:

- If there is an expression following the decimal point in text string data and the `getByte`, `getInt`, `getShort`, or `getLong` method is executed, the data following the decimal point is truncated and only the integer digits are converted and returned.
- If character string data contains double-byte characters, the method throws an `SQLException`. Double-byte characters include double-byte spaces filled when a character string shorter than the definition length of a column

is stored in an NCHAR column.

- If overflow occurs when character string data is converted to the Java data type, the method throws an `SQLException`.
- If the execution environment of the UAP is JDK or JRE 1.2 and the character string data uses exponential notation (such as `1.23E-23`) and the `getLong` method or `getBigDecimal` method is executed, the method throws an `SQLException`.

Table 16-7: Mapping between the getXXX methods and JDBC SQL types of ResultSet and CallableStatement (2/2)

getXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`getByte`	M[2]	—	—	—	—	—
`getShort`	M[2]	—	—	—	—	—
`getInt`	M[2]	—	—	—	—	—
`getLong`	M[2]	—	—	—	—	—
`getFloat`	M[2]	—	—	—	—	—
`getDouble`	M[2]	—	—	—	—	—
`getBigDecimal`	M[2]	—	—	—	—	—
`getBoolean`	M[2]	—	—	—	—	—
`getString`	R	M	M	M	M	—
`getBytes`	—	—	—	—	M	—
`getDate`	M[2]	R[3]	—	M	—	—
`getTime`	M[2]	—	R	M	—	—
`getTimestamp`	M[2]	M	—	R	—	—
`getAsciiStream`	M	—	—	—	M	—
`getUnicodeStream`	M	—	—	—	M	—
`getBinaryStream`	—	—	—	—	R	—
`getObject`	M	M	M	M	M	M

getXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`getCharacterStream`	M	—	—	—	M	—
`getArray`	—	—	—	—	—	R
`getBlob`	—	—	—	—	M	—
`getClob`[1]	—	—	—	—	—	—
`getRef`[1]	—	—	—	—	—	—

Legend:

R: Mapping is recommended.

M: Can be mapped.

—: Cannot be mapped.

[1] Not supported by the JDBC driver.

[2] In data conversion from character string data, any single-byte spaces that exist before and after the character string data is acquired from the database are removed, and then the data is converted to the Java data type returned by the `get`*XXX* method.

Note the following when you convert data to a Java data type:

- If there is an expression following the decimal point in text string data and the `getByte`, `getInt`, `getShort`, or `getLong` method is executed, the data following the decimal point is truncated and only the integer digits are converted and returned.
- If character string data contains double-byte characters, the method throws an `SQLException`. Double-byte characters include double-byte spaces filled when a character string shorter than the definition length of a column is stored in an NCHAR column.
- If overflow occurs when character string data is converted to the Java data type, the method throws an `SQLException`.
- If the execution environment of the UAP is JDK or JRE 1.2 and the character string data uses exponential notation (such as `1.23E-23`) and the `getLong` method or `getBigDecimal` method is executed, the method throws an `SQLException`.

[3] When the JDBC SQL type is the `DATA` type and conversion is executed by specifying a `java.util.Calendar` object for the `setDate` method, the specified

java.util.Calendar object is used for data conversion, time data is truncated, and only date data is stored in the database. In such a case, even if you specify the java.util.Calendar object for the getDate method to acquire the data stored using the setDate method, a different date than the date specified for the setDate method may be acquired.

Example:

The following is an example of when a java.util.Calendar object using Universal Time (UTC) is specified for the setDate and getDate methods in a UAP that uses Japanese standard time as the default time zone.

When you specify a java.sql.Date object that represents 2005-10-03 for the setDate method and then execute it, the JDBC driver adds 00:00:00 in the time part, and then stores the date part as 2005-10-02 in the database by delaying 9 hours because of the time zone difference. If this data is acquired using the getDate method, the date part 2005-10-02 is acquired from the database and 00:00:00 is added for the time part, and then 2005-10-02 09:00:00 is set by advancing 9 hours because of the time zone difference. Because of this, 2005-10-02 is set in the java.sql.Date object return value of the getDate method, which is different from the 2005-10-03 date specified for the setDate method.

(2) Data mapping when the ? parameter is specified

Table 16-8 shows set*XXX* methods and JDBC SQL types to be mapped for the PreparedStatement class and CallableStatement. For an unsupported JDBC SQL type, the set*XXX* method throws an SQLException. For details about the JDBC SQL types supported by the connected database, see *16.12 Data types and character codes*.

Note that the setCharacterStream method has been added because the setUnicodeStream method is no longer recommended in the JDBC2.0 basic standard.

Table 16-8: setXXX methods and JDBC SQL types to be mapped for PreparedStatement class

PreparedStatement class's setXXX method	JDBC SQL type to be mapped
setCharacterStream	CHAR, VARCHAR, or LONGVARCHAR
SetRef*	REF
setBlob	LONGVARBINARY
setClob*	CLOB
setArray	ARRAY

* Not supported by the JDBC driver.

Tables 16-9 and 16-10 show the mapping between the `set`*XXX* methods and JDBC SQL types of `PreparedStatement` and `CallableStatement`.

Table 16-9: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (1/2)

setXXX method	JDBC SQL type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	CHAR
`setByte`	M	M	M	M	M	M
`setShort`	R	M	M	M	M	M
`setInt`	M	R	M	M	M	M
`setLong`	M	M	M	M	M	M
`setFloat`	M	M	M	R	M	M
`setDouble`	M	M	R	M	M	M
`setBigDecimal`	M	M	M	M	R	M
`setBoolean`	M	M	M	M	M	M
`setString`	M	M	M	M	M	R
`setBytes`	—	—	—	—	—	—
`setDate`	—	—	—	—	—	M
`setTime`	—	—	—	—	—	M
`setTimestamp`	—	—	—	—	—	M
`setAsciiStream`	—	—	—	—	—	M
`setUnicodeStream`	—	—	—	—	—	M
`setBinaryStream`	—	—	—	—	—	—
`setObject`	M	M	M	M	M	M
`setCharacterStream`	—	—	—	—	—	M
`setArray`	—	—	—	—	—	—
`setBlob`	—	—	—	—	—	—
`setClob*`	—	—	—	—	—	—
`setRef*`	—	—	—	—	—	—

Legend:

R: Mapping is recommended.

M: Can be mapped. Note that data may be missing or a conversion error may occur depending on the format of source data.

—: Cannot be mapped.

* Not supported by the JDBC driver.

Table 16-10: Mapping between the setXXX methods and JDBC SQL types of PreparedStatement and CallableStatement (2/2)

setXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
`setByte`	M	—	—	—	—	—
`setShort`	M	—	—	—	—	—
`setInt`	M	—	—	—	—	—
`setLong`	M	—	—	—	—	—
`setFloat`	M	—	—	—	—	—
`setDouble`	M	—	—	—	—	—
`setBigDecimal`	M	—	—	—	—	—
`setBoolean`	M	—	—	—	—	—
`setString`	R	M	M	M	M	—
`setBytes`	—	—	—	—	M	—
`setDate`	M	R[2]	—	M	—	—
`setTime`	M	—	R	M	—	—
`setTimestamp`	M	M	—	R	—	—
`setAsciiStream`	M	—	—	—	M	—
`setUnicodeStream`	M	—	—	—	M	—
`setBinaryStream`	—	—	—	—	R	—
`setObject`	M	M	M	M	M	M

setXXX method	JDBC SQL type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVARBINARY	ARRAY
setCharacterStream	M	—	—	—	M	—
setArray	—	—	—	—	—	R
setBlob	—	—	—	—	M	—
setClob[1]	—	—	—	—	—	—
setRef[1]	—	—	—	—	—	—

Legend:

R: Mapping is recommended.

M: Can be mapped. Note that data may be missing or a conversion error may occur depending on the format of source data.

—: Cannot be mapped.

[1] Not supported by the JDBC driver.

[2] When the JDBC SQL type is the `DATA` type and conversion is executed by specifying a `java.util.Calendar` object for the `setDate` method, the specified `java.util.Calendar` object is used for data conversion, time data is truncated, and only date data is stored in the database. In such a case, even if you specify the `java.util.Calendar` object for the `getDate` method to acquire the data stored using the `setDate` method, a different date than the date specified for the `setDate` method may be acquired.

Example:

The following is an example of what happens when a `java.util.Calendar` object using Universal Time (UTC) is specified for the `setDate` and `getDate` methods in a UAP that uses Japanese standard time as the default time zone.

When you specify a `java.sql.Date` object that represents `2005-10-03` for the `setDate` method, and then execute it, the JDBC driver adds `00:00:00` in the time part and then stores the date part as `2005-10-02` in the database by delaying 9 hours because of the time zone difference. If this data is acquired using the `getDate` method, the date part `2005-10-02` is acquired from the database and `00:00:00` is added for the time part, and then `2005-10-02 09:00:00` is set by advancing 9 hours because of the time zone difference. Because of this, `2005-10-02` is set in the `java.sql.Date` object return value of the `getDate` method, which is different from the `2005-10-03` date specified for the `setDate` method.

16.4 JDBC2.0 Optional Package

16.4.1 Database connection using DataSource and JNDI

Database connection using `DataSource` and JNDI can now be used by the JDBC2.0 Optional Package.

Although it is not essential to use JNDI, using it offers a benefit in that you need to specify the connection information only once. Because `DataSource` class interface definition and JNDI are not included in JDK as standard features, you need to obtain them from the JavaSoft web site when developing application programs.

To connect to a database using `DataSource` and JNDI:

1. Create a `DataSource` object.
2. Set up connection information.
3. Register `DataSource` in JNDI.
4. Obtain `DataSource` from JNDI.
5. Connect to the database.

If you do not use JNDI, the operations in Steps 3 and 4 are unnecessary.

If you use JNDI, execute the operations in Steps 1 through 3 only once. Afterwards, you can connect to the database by performing the operations in Steps 4 and 5 only. Furthermore, after the operation in Step 4, you can modify the connection information as needed.

(1) Creating a DataSource object

Generate the `DataSource` class objects provided by the JDBC driver.

The `DataSource` class name of the JDBC driver required to generate the `DataSource` class objects is `JdbhDataSource`.

A `DataSource` class object creation example follows:

```
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource() ;
```

(2) Setting up connection information

Call up a connection information setup method for the `DataSource` object and set up connection information. Because a connection information acquisition method can also be used, you can also check the current connection information. For details on the connection information setup/acquisition method, see *16.11 Connection information setup/acquisition interface*.

(3) Registering DataSource in JNDI

Register the `DataSource` object in JNDI.

In JNDI, you can select a service provider from several that are available.

An example of obtaining a `DataSource` object in JNDI is shown as follows (for Windows). Note that this obtaining example uses File System, which is one of the service providers. For information on other service providers, see the JNDI documentation.

```
// Generate a DataSource class object provided by the JDBC driver.
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource();

// Specify connection information.
        ...


// Obtain system properties.
Properties sys_prop = System.getProperties() ;

// Set up properties for the File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set up the directory to be used by the File System service provider.
// (In this case, the directory is registered under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");

// Update the system properties.
System.setProperties(sys_prop) ;


// Initialize JNDI.
Context ctx = new InitialContext();

// Register a DataSource class object provided by the HiRDB driver in JNDI
// under a logical name called jdbc/TestDataSource.
ctx.bind("jdbc" + "\\" + "TestDataSource", ds);
                       ...
```

Note that the JDBC2.0 specification recommends that the logical name to be registered in JNDI be registered under a subcontext called `jdbc` (`jdbc/TestDataSource` in the registration example).

(4) Obtaining DataSource from JNDI

Obtain the `DataSource` object from JNDI.

An example of obtaining a `DataSource` object from JNDI is shown as follows (for Windows). Note that this obtaining example uses File System, which is one of service providers. For information on other service providers, see the JNDI documentation.

```
// Obtain system properties.
Properties sys_prop = System.getProperties() ;

// Set up properties for the File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set up the directory to be used by the File System service provider.
// (In this case, the directory is registered under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");


// Update the system properties.
System.setProperties(sys_prop) ;

// Initialize JNDI.
Context ctx = new InitialContext();


// Obtain an object with a logical name jdbc/TestDataSource from JNDI.
Object obj = ctx.lookup("jdbc" + "\\" + "TestDataSource") ;

// Cast the extracted object into the DataSource class type.
DataSource ds = (DataSource)obj;
              ...
```

(5) Connecting to the database

Invoke the `getConnection` method for the `DataSource` object.

An example of calling the `getConnection` method follows:

```
DataSource ds

// Obtain a DataSource object from JNDI.
                        ...


// Issue the getConnection method.
Connection con = ds.getConnection();
  or
Connection con = ds.getConnection("USERID", "PASSWORD");*
```

* The method's arguments (authorization identifier and password) take precedence over the connection information set for the `DataSource` objects. If the necessary connection information is not set for the `DataSource` object and the connection information is invalid or connection with the HiRDB server fails, the `getConnection` method throws an `SQLException`.

You can set connection information again as necessary after the `Datasource` object is obtained from JNDI. In such a case, you must cast the `Datasource` object to the `DataSource` class type provided by the JDBC driver and then set the connection information.

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource hirdb_ds;

// Obtain a DataSource object from JNDI.
                     ...


// Cast the DataSource object to the DataSource class type provided by the JDBC driver.
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.JdbhDataSource)ds;

// Reset the connection information.
          ...
```

16.4.2 Connection pooling

A function is provided in the JDBC2.0 Optional Package for pooling connections to a database. An overview of connection pooling is provided below:

- Connection pooling has no effect on existing applications. This means that applications do not need to be aware of connection pooling. However, this assumes that the database is not connected by `DriverManager`, but rather by `DataSource` and JNDI provided by the JDBC2.0 Optional Package.
- The connection pooling function itself is outside the functional scope of the JDBC specifications. This is intended to allow the user to select a desired connection pooling function when building a system (the user can create one, use one provided by an APServer vendor, or use one provided by a JDBC vendor).
- With the connection pooling function, the `Datasource` class can be used as an interface with applications. This `DataSource` class is different from the `DataSource` class provided by the JDBC driver.
- With the JDBC driver, the `ConnectionPoolDataSource` class and `PooledConnection` class can be used as an interface with the connection pooling function.
- The `ConnectionPoolDataSource` class provided by the JDBC driver can use the connection information setting and acquisition methods in the same way as the `Datasource` class provided by the JDBC driver.

Table 16-11 shows classes related to connection pooling.

Table 16-11: Classes related to connection pools

Class	Overview
javax.sql.DataSource	• Provided by a connection pooling function. • Used as the interface to applications during database connection. • Normally, connection pools are controlled in this class. • Normally, registered in JNDI for use. • Different from the `DataSource` class provided by the JDBC driver.
javax.sql.ConnectionPoolDataSource	• Provided by the JDBC driver. • Can use a method for setting/acquiring connection information necessary for database connection. • Normally not used directly from an application, and is used by a connection pooling function. • Normally, registered in JNDI for use. • A connection pooling function acquires a `PooledConnection` object from this class of objects.
javax.sql.PooledConnection	• Provided by the JDBC driver. • Normally not used directly from an application, and is used by a connection pooling function. • A connection pooling function targets this class of objects for pooling. • A connection pooling function acquires a `Connection` object to be used by an application from this class of objects.
javax.sql.ConnectionEventListener	• Provided by a connection pooling function. • A connection pooling function senses a connection pooling trigger by detecting a disconnection or SQL error through this class of objects.

Depending on the JDK version, the interface definition of the classes shown in Table 16-11 might not be included in the JDK standard; you will need to check the JavaSoft website if you intend to use the connection pooling function.

The following are the package name and class names of the classes provided by the JDBC driver and shown in Table 16-11.

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`ConnectionPoolDataSource` class name: `JdbhConnectionPoolDataSource`

`PooledConnection` class name: `JdbhPooledConnection`

Note that the setting of the connection information of the `ConnectionPoolDataSource` class provided by the JDBC driver is the same as the setting of the connection information of the `DataSource` class provided by the JDBC driver.

16.4.3 Distributed transactions

In the JDBC2.0 Optional Package, distributed transactions in cooperation with the transaction manager (TM) based on the XA standard of X/Open are defined as an extension of the connection pooling function. The following provides an overview of distributed transactions:

- Connection pooling has almost no effect on existing applications. However, there are certain restrictions, such as that direct commit is not allowed. Also, as with connection pooling, it is a precondition that database connection is not performed by using `DriverManager`, but rather by using `DataSource` and JNDI introduced by the JDBC2.0 Optional Package.
- As with connection pooling, the transaction linkage function for linking with a TM is outside the functional scope of the JDBC specifications.
- Normally, a transaction linkage function is installed as an extension of a connection pooling function, and uses TM-provided JTA and JTS as the interface with the TM. Note that operations complying with JTA standard 1.0 are not guaranteed.
- In the transaction linkage facility, as with connection pooling, the `DataSource` class can be used as the interface with applications. This `DataSource` class is different from the one provided by the JDBC driver.
- The JDBC driver can use the `XADataSource` class and `XAConnection` class as an interface with the transaction linkage facility. Also, the JDBC driver can use `XAResource` class as an interface with TM.
- As with the `DataSource` class provided by the JDBC driver, the `XADataSource` class provided by the JDBC driver can use the connection information setting/ acquisition methods.

As with connection pooling, `Connection` objects used by applications are generated by the `XAConnection` class. However, there are certain differences compared with the `Connection` objects generated by the `DataSource` class provided by the `PooledConnection` class or JDBC driver.

- The method of invoking a `commit` method or `rollback` method for the `Connection` class is based on an `SQLException`. That is, an application cannot directly complete a transaction.
- The default mode for `AutoCommit` is `OFF`.
- Issuance of a `Connection` class's `setAutoCommit (true)` method that turns on the `AutoCommit` mode results in an `SQLException`.

Table 16-12 lists the classes related to distributed transactions.

Table 16-12: Classes related to distributed transactions

Class	Overview
`javax.sql.DataSource`	• Provided by a transaction linkage function. • Used as the interface to applications during database connection. • Normally, linkage to a TM and connection pools are controlled in this class. • Normally, registered in JNDI for use. • Different from the `DataSource` class provided by the JDBC driver.
`javax.sql.XADataSource`	• Provided by the JDBC driver. • Can use a method for setting/acquiring connection information necessary for database connection. • Normally not used directly from an application, and is used by a transaction linkage function. • Normally, registered in JNDI for use. • A transaction linkage function acquires an `XAConnection` object from this class of objects.
`javax.sql.XAConnection`	• Provided by the JDBC driver. • This is a subclass of the `PooledConnection` class. That is, it inherits all methods related to connection pooling. • Normally not used directly from an application, and is used by a transaction linkage function. • A transaction linkage function targets this class of objects for pooling. • A transaction linkage function acquires a `Connection` object to be used by an application from this class of objects.
`javax.sql.ConnectionEventListener`	• Provided by a transaction linkage function. • A transaction linkage function senses a connection pooling trigger by detecting a disconnection or SQL error through this class of objects.
`javax.transaction.xa.XAResource`	• Provided by the JDBC driver. • Can use the XA-related methods used by a TM.
`javax.transaction.xa.Xid`	• Provided by the JDBC driver and TM. • Used as the argument/return value of an `XAResource` class method.

Because the interface definition of the classes listed in Table 16-12 is not included in JDK as a standard feature, you must acquire them from the JavaSoft website when you develop a transaction linkage facility.

The following are the package names and class names of the classes provided by the JDBC driver and shown in Table 16-12.

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

`XADataSource` class name: `JdbhXADataSource`

`XAConnection` class name: `JdbhXAConnection`

`XAResource` class name: `JdbhXAResource`

`Xid` class name: `JdbhXid`

Note that the setting of the connection information of the `XADataSource` class provided by the JDBC driver is the same as the setting of the connection information of the `DataSource` class provided by the JDBC driver.

16.5 JAR file access facility

To use a Java stored routine, you need to register the JAR file in HiRDB. This processing takes place via the JDBC driver.

This section explains the class and method names used to register, delete, and re-register JAR files.

16.5.1 Class name

The class name follows:
`JP.co.Hitachi.soft.HiRDB.JDBC.Jdbh_JARAccess`

16.5.2 Method name

(1) Registering a JAR file in HiRDB

(a) Format

```
public void Jdbh_JARInstall(java.sql.Connection    con,
                            String                 JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to register the JAR file.

JarName

Specifies the name of the JAR file.

Specify either the absolute path name or relative path name. You cannot specify a file located in another server machine, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method registers the specified JAR file in HiRDB using the `Java.sql.Connection` object. If HiRDB already contains a file with the same name, an error occurs.

(2) Deleting a JAR file from HiRDB

(a) Format

```
public void Jdbh_JARUnInstall(java.sql.Connection   con,
                              String                JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to delete the JAR file.

JarName

Specifies the name of the JAR file.

You cannot specify an absolute path name, nor a relative path name, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method deletes the specified JAR file from HiRDB using the `Java.sql.Connection` object.

(3) Re-registering a JAR file in HiRDB

(a) Format

```
public void Jdbh_JARReInstall(java.sql.Connection   con,
                              String                JarName)
```

(b) Arguments

con

Specifies a `Java.sql.Connection` object to re-register the JAR file.

JarName

Specifies the name of the JAR file.

You cannot specify an absolute path name, nor a relative path name, nor a wildcard.

(c) Return value

None.

(d) Exception

`SQLException`

A database access error occurred.

(e) Function

This method re-registers the specified JAR file in HiRDB using the `Java.sql.Connection` object. If HiRDB already contains a file with the same name, it is overwritten (an error does not occur).

16.6 Array class

The JDBC driver can access repetition columns using the Array class. Note the following when using each method:

(1) getArray

- MAP cannot be used.
- Table 16-13 shows the object types returned by this method.

Table 16-13: Object types returned by getArray

HiRDB data type	Object type
INTEGER	java.lang.Integer[]
SMALLINT	java.lang.Short[]
DECIMAL	java.math.BigDecimal[]
FLOAT, DOUBLE PRECISION	java.lang.Double[]
SMALLFLT, REAL	java.lang.Float[]
CHAR	java.lang.String[]
VARCHAR	java.lang.String[]
NCHAR	java.lang.String[]
NVARCHAR	java.lang.String[]
MCHAR	java.lang.String[]
MVARCHAR	java.lang.String[]
DATE	java.sql.Date[]
TIME	java.sql.Time[]
TIMESTAMP	java.sql.Timestamp[]

(2) getResultSet

- MAP cannot be used.
- The result set returned by this method includes one row in each array element, and each row has two columns. The second column stores the value of the element, while the first column stores the index of the corresponding element inside the array (the index of the first array element is 1). Rows are arranged in ascending order based on the indexes.

- Closing a statement also closes the result returned by this method.
- Table 16-14 shows the attribute values of the result sets returned by this method.

Table 16-14: Attribute values of the result sets returned by getResultSet

ResultSetMetaData class method name	Values returned by the method	
	First column	Second column
`getCatalogName`	`null`	`null`
`getColumnClassName`	`java.lang.Integer`	Depends on the database column attribute.
`getColumnDisplaySize`	`10`	Depends on the database column length.
`getColumnLabel`	`JDBC_Array_Index`	Depends on the database column name.
`getColumnName`	`JDBC_Array_Index`	Depends on the database column name.
`getColumnType`	`java.sql.Types.INTEGER`	Depends on the database column attribute.
`getColumnTypeName`	`INTEGER`	Depends on the database column attribute.
`getPrecision`	`10`	Depends on the database column attribute and column length.
`getScale`	`0`	Depends on the database column attribute and column length.
`getSchemaName`	`null`	`null`
`getTableName`	`null`	`null`
`isAutoIncrement`	`true`	`false`
`isCaseSensitive`	`false`	Depends on the database column attribute.
`isCurrency`	`false`	`false`
`isDefinitelyWritable`	`false`	`false`
`isNullable`	`java.sql.ResultSetMetaData.columnNoNulls`	Depends on the database column attribute.
`isReadOnly`	`true`	`false`
`isSearchable`	`false`	`true`
`isSigned`	`false`	Depends on the database column attribute.
`isWritable`	`false`	`false`

16.7 Specifying a value when using a repetition column as the ? parameter

This section explains how to specify a value when using a repetition column as the `?` parameter.

To specify a value for the `?` parameter, use the `setObject` method to specify an object in the class in which the Array interface was installed or a column object.

(1) Specifying an object in the class in which the Array interface was installed

- Create an object in the class in which the Array interface was installed, and use the `setArray` or `setObject` method to specify that object.
- The JDBC driver uses the `Array.getBaseType` method to check the data type of that object. If the data type of the database turns out to be different from the data type of the object, the JDBC driver throws an `SQLException`. For details on database and object data types, see *16.12.1 Data types*.
- The actual data is acquired using the `Array.getArray()` method without any argument. Table 16-15 shows the object types that must be returned during this data acquisition. If the object type returned is different from those shown in Table 16-15, the JDBC driver throws an `SQLException`.

Table 16-15: Object types returned during data acquisition using the Array.getArray() method without any argument

Data type returned by the Array.getBaseType method	Object types returned during data acquisition using the Array.getArray() method without any argument
`java.sql.Types.INTEGER`	`int[]` or `java.lang.Integer[]`
`java.sql.Types.SMALLINT`	`short[]` or `java.lang.Short[]`
`java.sql.Types.DECIMAL`	`java.math.BigDecimal[]`
`java.sql.Types.FLOAT`	`double[]` or `java.lang.Double[]`
`java.sql.Types.REAL`	`float[]` or `java.lang.Float[]`
`java.sql.Types.CHAR`	`java.lang.String[]`
`java.sql.Types.VARCHAR`	`java.lang.String[]`
`java.sql.Types.DATE`	`java.sql.Date[]`
`java.sql.Types.TIME`	`java.sql.Time[]`
`java.sql.Types.LONGVARBINARY`	`java.io.DataInputStream[]`

Data type returned by the Array.getBaseType method	Object types returned during data acquisition using the Array.getArray() method without any argument
`java.sql.Types.TIMESTAMP`	`java.sql.Timestamp[]`

(2) Using the setObject method to specify an array object

- If the database data type is different from the array object data type, the JDBC driver throws an `SQLException`.
- If the data type of the SQL statement specified by the `setObject` method and the data type of the array object are different from those shown in Table 16-16, the JDBC driver throws an `SQLException`.

Table 16-16: Data type of the SQL statement specified by the setObject method and the data type of the array object

Data type of the SQL statement specified by the setObject method	Data type of the array object
`java.sql.Types.INTEGER`	`int[]` or `java.lang.Integer[]`
`java.sql.Types.SMALLINT`	`short[]` or `java.lang.Short[]`
`java.sql.Types.DECIMAL`	`java.math.BigDecimal[]`
`java.sql.Types.FLOAT`	`double[]` or `java.lang.Double[]`
`java.sql.Types.REAL`	`float[]` or `java.lang.Float[]`
`java.sql.Types.CHAR`	`java.lang.String[]`
`java.sql.Types.VARCHAR`	`java.lang.String[]`
`java.sql.Types.DATE`	`java.sql.Date[]`
`java.sql.Types.TIME`	`java.sql.Time[]`
`java.sql.Types.LONGVARBINARY`	`java.io.DataInputStream[]`
`java.sql.Types.TIMESTAMP`	`java.sql.Timestamp[]`

(3) Relationship between repetition column elements and the object specified as the ? parameter

The sequence of the array objects obtained by the `Array.getArray()` method from the objects in the class in which the Array interface was installed is the same as the sequence of the repetition columns. Consequently, the first element of the array object becomes the first element of the repetition column, and the second element of the array object becomes the second element of the repetition column.

The same also holds true for the array objects specified by the `setObject` method. You can also specify an array object consisting of only one element.

(4) Specifying a null value for an element in the middle of a repetition column

Regardless of whether an object is in the class in which the Array interface was installed or an array object, if you specify a null value for an element in the middle of an element, the element of the applicable array becomes null. Therefore, to set a null value for the second element of a repetition column, specify a null value for the second element of the array object obtained by the `Array.getArray()` method from the objects in the class in which the Array interface was installed.

The same also holds true for the array objects specified by the `setObject` method.

16.8 Functions provided by the HiRDB JDBC driver

This section describes the HiRDB JDBC driver functions that are not standardized by JDBC2.0.

16.8.1 Provided class

To use the functions provided only by the HiRDB JDBC driver, you must use the following class:

Interface name	Main function	Class name
`PreparedStatement`	• Executing SQL statements with the `?` parameter • Setting values for the `?` parameter • `Statement` functions (all functions are inherited because this is `Statement`'s subclass)	`JdbcDbpsvPreparedStatement`

16.8.2 setBlockUpdate

(a) Function

`setBlockUpdate` specifies whether or not multiple parameters are to be processed at one time when the `?` parameter is used to update databases.

(b) Format

```
public void setBlockUpdate(boolean Mode)
```

(c) Arguments

`boolean Mode`

Specifies whether or not multiple parameter sets are to be processed at one time. When this argument is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.*

* During database connection, if `BLOCK_UPDATE=TRUE` is specified in the argument of the `getConnection` method of the `DriverManager` class, the

default for this function is `true`. Also, when `HiRDB_for_Java_BLOCK_UPDATE=TRUE` is specified in the system property, the default for this function is `true`.

(d) Return value

None.

(e) Functional detail

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(f) Exception

None.

(g) Notes

For details about how to process multiple lines of `?` parameters in batch mode, see *Table 16-3 Information to be specified for Properties info* and *16.3.2 Batch updating*.

16.8.3 getBlockUpdate

(a) Function

This function acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

(b) Format

```
public boolean getBlockUpdate()
```

(c) Arguments

None.

(d) Return value

`boolean`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.*

* During database connection, if `BLOCK_UPDATE=TRUE` is specified in the argument of the `getConnection` method of the `DriverManager` class, the default for this function is `true`.

(e) Functional detail

This function acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

(f) Exception

None.

16.9 Notes on using the BLOB type

This section provides notes about the processing of methods when the BLOB type is used.

(1) Method processing and notes

Table 16-17 describes the processing of each method.

Table 16-17: Method processing and notes

Method name of Blob interface class	Processing and notes
getBinaryStream	Returns the InputStream class equipped with JdbhInputStream. The maximum length of data that can be acquired is 2,147,483,639.
getBytes(long pos, int length)	Returns the maximum length of data from the specified pos location using the byte[] object. If the database contents are the null value, no data can be acquired from the specified location; or if the data length is 0 bytes, the method returns null. The maximum value is 2,147,483,639. If this length is exceeded, the method throws an SQLException.
length()	Returns the actual data length.
position(Blob pattern,long start)	Executes position(pattern.getBytes(1, (int)(pattern.length())), start). If null is specified in pattern, the method throws a NullPointerException.
position(byte[] pattern,long start)	Returns the position corresponding to pattern from the specified start location. The return value is >=start. If there is no location that corresponds to pattern, the method returns -1. The maximum value of pattern.length is 2,147,483,639. If this value is exceeded, the method throws an SQLException. If null is specified in pattern, the method throws a NullPointerException.
setBinaryStream(long pos)	Unconditionally throws an SQLException.
setBytes(long pos,byte[] bytes)	
setBytes(long pos,byte[] bytes,int offset,int len)	
truncate(long len)	

Note

If you have acquired data using the locator facility and execute ResultSet.close() or Statement.close(), you can no longer acquire data.

(2) Specification method using the ? parameter

To specify a value in the `?` parameter, you can use the `PreparedStatement.setBlob()` and `CallableStatement.setBlob()` methods. This subsection provides notes about using these methods.

(a) When using objects equipped with the Blob interface

When using the `setBlob()` method, you must specify an object equipped with the Blob interface. Additionally, the UAP must create the object equipped with the Blob interface.

JDBC uses the `Blob.getBytes()` method to acquire the value to be set in the `byte[]` format. The following method is used to acquire the value to be used:

```
Blob.getBytes(1, (int)(Blob.length()))
```

In the UAP, the `getBytes()` and `length()` methods must return normal values. JDBC assumes that the values returned by these methods are correct.

(b) When using the Blob object acquired by the ResultSet.getBlob() or CallableStatement.getBlob() method

When the Blob object acquired by the `ResultSet.getBlob()` or `CallableStatement.getBlob()` method as the execution result from JDBC is to be used as is, operation depends on whether or not the object was acquired by using the locator facility for access.

- When the locator facility was not used for access

  The data acquired by the `ResultSet.getBlob()` or `CallableStatement.getBlob()` method is used as the value of the `?` parameter.

- When the locator facility was used for access

  When the `setBlob()` method is called, `Blob.getBytes(1, (int)(Blob.length()))` is executed internally. The data acquired by `Blob.getBytes(1, (int)(Blob.length()))` is used as the value of the `?` parameter.

16.10 Setting system properties

16.10.1 Setting the array facility

(1) Overview

If you set the `HiRDB_for_Java_BLOCK_UPDATE` system property during program execution, you can specify whether or not to process multiple parameter sets at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

(2) Setting method

During program execution, use the `-D` option of the `java` command to set the `HiRDB_for_Java_BLOCK_UPDATE` system property.

(a) Function

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, or `DELETE`).

(b) Format

`java -D`<*name*>`=`<*value*> *class-name*

(c) Description

`name`

`HiRDB_for_Java_BLOCK_UPDATE`

`value`

`TRUE`: Processes multiple parameter sets at one time.

`FALSE`: Processes one parameter set at a time.

Other: Processes one parameter set at a time.

(d) Functional detail

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(e) Notes

- When you specify `-D`<*name*>`=`<*value*>, make sure that there is no space in the specified information. The specified information cannot be set correctly if it has any of the following formats, where Δ indicates a space:

- -DΔ<*name*>=<*value*>
- -D<*name*>Δ=<*value*>
- -D<*name*>=Δ<*value*>

- If `BLOCK_UPDATE` is set during database connection (`setBlockUpdate` method during data source connection), `BLOCK_UPDATE` or the value set in the `setBlockUpdate` method takes effect.
- If you used the `PreparedStatement` class's `setBlockUpdate` method, you can change the setting as to whether or not multiple parameter sets are to be processed at one time.
- For details about how to process multiple lines of `?` parameters in batch mode, see *Table 16-3 Information to be specified for Properties info* and *16.3.2 Batch updating*.

(f) Example

The following shows an example of setting the `HiRDB_for_Java_BLOCK_UPDATE` system property:

```
java -DHiRDB_for_Java_BLOCK_UPDATE=TRUE TestUP
```

16.10.2 Setting the maximum number of SQL search items or ? parameters

(1) Overview

If you set the `HiRDB_for_Java_SQL_IN_NUM` or `HiRDB_for_Java_SQL_OUT_NUM` system property during program execution, you can specify the maximum number of search items, output `?` parameters, input `?` parameters, or input/output `?` parameters that are to be acquired during SQL preprocessing.

(2) Setting method

During program execution, set the system property `HiRDB_for_Java_SQL_OUT_NUM` or `HiRDB_for_Java_SQL_IN_NUM` or both in the `-D` option of the `java` command.

(a) Function

This function specifies the maximum number of search items, output `?` parameters, input `?` parameters, or input/output `?` parameters that are to be acquired during SQL preprocessing.

(b) Format

`java -D`<*name*>=<*value*> *class-name*

(c) Description

The following table describes the information that can be specified in <*name*> and <*value*>:

<name>	<value>
`HiRDB_for_Java_SQL_IN_NUM`	Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed. This is the number of input or input/output `?` parameter information items acquired during SQL preprocessing. If the actual number of input or input/output `?` parameters is greater than this property value, the input or input/output `?` parameter information is acquired after the SQL preprocessing. The permitted value range is from `1` to `30,000` (default is `64`). Specifying any other value or a non-numeric value results in an error during database connection.
`HiRDB_for_Java_SQL_OUT_NUM`	Specifies the maximum number of output items for the SQL statement to be executed. This is the number of output items acquired during SQL preprocessing. If the actual number of output items is greater than this property value, the output items are acquired after the SQL preprocessing. The permitted value range is from `1` to `30,000` (default is `64`). Specifying any other value or a non-numeric value results in an error during database connection.

(d) Functional detail

This function specifies the maximum number of search items, output `?` parameters, input `?` parameters, or input/output `?` parameters that are to be acquired during SQL preprocessing. A sufficient value enables you to acquire search item, output `?` parameter, input `?` parameter, or input/output parameter information during SQL preprocessing, thereby improving performance compared to when this information is acquired after preprocessing.

(e) Notes

- When you specify `-D`<*name*>=<*value*>, make sure that there is no space in the specified information. The specified information cannot be set correctly if it has any of the following formats, where **Δ** indicates a space:
  - `-D`**Δ**<*name*>=<*value*>
  - `-D`<*name*>**Δ**=<*value*>
  - `-D`<*name*>=**Δ**<*value*>

- If HiRDB_for_Java_SQL_IN_NUM is set during database connection (setSQLInNum method during data source connection), HiRDB_for_Java_SQL_IN_NUM or the value set in the setSQLInNum method takes effect.
- If HiRDB_for_Java_SQL_OUT_NUM is set during database connection (setSQLOutNum method during data source connection), HiRDB_for_Java_SQL_OUT_NUM or the value set in the setSQLOutNum method takes effect.
- To acquire search item, output ? parameter, input ? parameter, or input/output parameter information during SQL preprocessing, the version of the connected HiRDB server must be 07-02 or later.

(f) Example

The following shows an example of setting the HiRDB_for_Java_SQL_IN_NUM and HiRDB_for_Java_SQL_OUT_NUM system properties:

```
java -DHiRDB_for_Java_SQL_IN_NUM=128
-DHiRDB_for_Java_SQL_OUT_NUM=128 TestUP
```

16.11 Connection information setup/acquisition interface

The JdbhDataSource, JdbhConnectionPoolDataSource, and JdbhXADataSource classes, which are provided by the JDBC driver, provide methods of setting/acquiring the connection information necessary for database connection, besides the methods specified by the JDBC2.0 Optional Package specification.

Table 16-18 lists the methods of setting/acquiring connection information.

Table 16-18: Methods of setting/acquiring connection information

Method	Function
setDescription	Sets the additional connection information needed by the database to be connected.
getDescription	Acquires the additional connection information needed by the database to be connected.
setDBHostName	Sets the host name of the HiRDB to be connected.
getDBHostName	Acquires the host name of the HiRDB to be connected.
setEncodeLang	Uses the specified encoding character code to convert data.
getEncodeLang	Returns the encoding characters to be used for data conversion.
setUser	Sets the authorization identifier.
getUser	Acquires the authorization identifier.
setPassword	Sets a password.
getPassword	Acquires a password.
setXAOpenString*	Sets an XA_OPEN character string.
getXAOpenString*	Acquires an XA_OPEN character string.
setXACloseString*	Sets an XA_CLOSE character string.
getXACloseString*	Acquires an XA_CLOSE character string.
setRMID*	Sets a resource manager identifier.
getRMID*	Acquires a resource manager identifier.
setXAThreadMode*	Sets a thread mode for using XA.

Method	Function
getXAThreadMode*	Acquires a thread mode for using XA.
setCommit_Behavior	Sets whether a cursor remains valid following COMMIT.
getCommit_Behavior	Acquires whether a cursor remains valid following COMMIT.
setBlockUpdate	Specifies whether or not multiple parameter sets are to be processed at one time.
getBlockUpdate	Acquires a value indicating whether or not multiple parameter sets are to be processed at one time.
setLONGVARBINARY_Access	Specifies the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY).
getLONGVARBINARY_Access	Acquires the access method for a LONGVARBINARY database (column attribute is BLOB or BINARY).
setSQLInNum	Specifies the maximum number of input or input/output ? parameters in the SQL statements to be executed.
getSQLInNum	Acquires the maximum number of input or input/output ? parameters in the SQL statements to be executed that has been set by setSQLInNum.
setSQLOutNum	Specifies the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed.
getSQLOutNum	Acquires the maximum number of search items, output ? parameters, or input/output ? parameters in the SQL statements to be executed that has been set by setSQLOutNum.
setSQLWarningLevel	Specifies the warning retention level that occurred during execution of SQL statements.
getSQLWarningLevel	Acquires the warning retention level specified in setSQLWarningLevel.
setClear_Env	Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection.
getClear_Env	Acquires the environment variable invalidation setting specified by setClear_Env.

* These methods are provided by the JdbhXADataSource class only.

16.11.1 setDescription

(a) Function

Sets the additional connection information needed by the database to be connected.

(b) Format

```
public void setDescription (String description)
```

(c) Argument

String `description`

Specifies additional connection information.

(d) Return value

None.

(e) Functional detail

Sets the additional connection information needed by the database to be connected. Setting details and whether setting is required are shown as follows.

Setting	Setting details	Setting required?
HiRDB port number	Sets the HiRDB port number expressed as a character string.	Optional
HiRDB environment variable group name	Sets the HiRDB environment variable group name following `@HIRDBENVGRP=`, expressed as a character string. If the environment variable name contains single-byte spaces or single-byte `@` characters, enclose the name in single-byte quotation marks (`"`). When an environment variable group name is enclosed in single-byte quotation marks, all characters following the last single-byte quotation mark through the end of the character string are ignored. An environment variable group name containing single-byte quotation marks or single-byte commas cannot be specified.	Optional
HiRDB environment variable group identifier	Sets the HiRDB environment variable group identifier expressed as four alphanumeric characters.	Required during XA connection

Note 1

The environment variables registered in an environment variable group have precedence over the user environment variables and the environment variables registered by `HiRDB.INI`.

Note 2

Specification examples are shown below. In these examples, `ds` represents the name of a variable that has reference to the `JdbhDataSource` class's instance.

In UNIX:

Example 1: When the path of the HiRDB environment variable group name is `/HiRDB_P/Client/HiRDB.ini`

```
ds.setDescription("@HIRDBENVGRP=/HiRDB_P/Client/
HiRDB.ini");
```

In Windows

Example 1: When specifying the HiRDB port number

```
ds.setDescription("22200");
```

Example 2: When specifying the environment variable group name HiRDB_ENV_GROUP that has been registered using the tool for registering HiRDB client environment variables

```
ds.setDescription("@HIRDBENVGRP=HiRDB_ENV_GROUP");
```

Example 3: When the path of the HiRDB environment variable group name is C:\HiRDB_P\Client\HiRDB.ini

```
ds.setDescription("@HIRDBENVGRP=C:\\HiRDB_P\\Client\\Hi
RDB.ini");
```

Example 4: When the path of the HiRDB environment variable group name is C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini (∆: single-byte space)

```
ds.setDescription("@HIRDBENVGRP=\"C:\\Program∆Files\\H
ITACHI\\HiRDB\HiRDB.ini\"");
```

Example 5: When the HiRDB environment variable group identifier is HDB1

```
ds.setDescription("HDB1");
```

(f) Exception that occurs

If an environment variable group name beginning with `@` is specified during a connection other than the XA connection, and the specified information following `@` contains a single-byte space, this method throws an `SQLException`.

16.11.2 getDescription

(a) Function

Acquires the additional connection information needed by the database to be connected.

(b) Format

```
public String getDescription()
```

(c) Argument

None.

(d) Return value

String

This is the additional connection information. If none is set, `null` is returned.

(e) Functional detail

Returns the additional connection information needed by the database to be connected that was specified by the `setDescription` method.

(f) Exception that occurs

None.

16.11.3 setDBHostName

(a) Function

Sets the host name of the HiRDB to be connected.

(b) Format

```
public void setDBHostName (String db_host_name)
```

(c) Argument

String `db_host_name`

Sets a HiRDB host name.

(d) Return value

None.

(e) Functional detail

Sets the host name of the HiRDB to be connected (host name set in the `PDHOST` client environment definition).

If the connection is not XA and the environment variable group name of a HiRDB client is specified in the additional connection information, the value specified by this method will be ignored.

(f) Exception that occurs

None.

16.11.4 getDBHostName

(a) Function

Acquires the host name of the HiRDB to be connected.

(b) Format

```
public String getDBHostName()
```

(c) Argument

None.

(d) Return value

String

This is the HiRDB host name. If none is set, `null` is returned.

(e) Functional detail

Returns the host name of the HiRDB to be connected that was specified by the `setDBHostName` method.

(f) Exception that occurs

None.

16.11.5 setEncodeLang

(a) Function

Specifies the character set used for character code conversion in the JDBC driver.

(b) Format

```
public void setEncodeLang (String encode_lang)
```

(c) Argument

String `encode_lang`

Specifies a character set supported by Java (such as MS932).

If `OFF` is specified with this method or if nothing is specified (including in the `ENCODELANG` settings of `Properties info` and the URL), the following operation takes place.

`OFF`:

The JDBC driver determines the character set that corresponds to the character codes type of the connected HiRDB. The following table shows the correspondence between the connected HiRDB character codes type and the character encoding used by the JDBC driver:

HiRDB character codes type*	Character encoding used
`lang-c`	`8859_1`
`sjis`	Java Virtual Machine standard encoding
`ujis`	EUCJIS
`utf-8`	UTF-8
`chinese`	GB2312

* The specification value is in the `-c` option of the `pdsetup` command for UNIX and the `-c` option of the `pdntenv` command for Windows. For the character codes types when the `pdntenv` command is not executed, see the *HiRDB Version 8 Installation and Design Guide*.

`None`:

For UNIX:

The JDBC driver determines the character set that corresponds to the HiRDB character codes type.

For Windows:

The JDBC driver uses the following rules to determine the character set:

Java Virtual Machine standard encoding	HiRDB character codes type	
	SJIS	Other than SJIS
MS932	MS932	Character set corresponding to the HiRDB character codes type
Other than MS932	SJIS	

(d) Return value

None.

(e) Functional detail

In a Java program, Unicode is used for the character codes. Therefore, during character data processing with HiRDB, the JDBC driver performs mutual character code conversion between the HiRDB character data and Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the Java Virtual Machine. This method specifies the character set names specified by the JDBC driver for the encoder and decoder that are provided by the Java Virtual Machine.

(f) Exception that occurs

None.

16.11.6 getEncodeLang

(a) Function

Acquires the specified character set.

(b) Format

```
public String getEncodeLang()
```

(c) Argument

None.

(d) Return value

String

Returns the character set.

(e) Functional detail

Returns the character set specified by the `setEncodeLang` method. If no character set is specified, `null` is returned.

(f) Exception that occurs

None.

16.11.7 setUser

(a) Function

Sets the authorization identifier.

(b) Format

```
public void setUser (String user)
```

(c) Argument

String `user`

Sets the authorization identifier.

(d) Return value

None.

(e) Functional detail

Sets the authorization identifier.

You can specify the authorization identifier using an argument of the `DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method (which are referred to generically as the *DB connection methods*).

If this method is used to set an authorization identifier, and if a DB connection method that has an authorization identifier and a password set as arguments is also called, the authorization identifier setting specified by the DB connection method takes precedence.

For details about specifying an authorization identifier, see *Table 16-2 Arguments of the getConnection method*.

(f) Exception that occurs

None.

16.11.8 getUser

(a) Function

Acquires the authorization identifier.

(b) Format

```
public String getUser()
```

(c) Argument

None.

(d) Return value

String

Sets the authorization identifier. If no authorization identifier has been set, `null` is returned.

(e) Functional detail

Returns the authorization identifier specified by the `setUser` method.

If the `setUser` method is used to set a password, and if a DB connection method (`DataSource.getConnection method`, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method) that has an authorization identifier and a password set as arguments is also called, the authorization identifier setting specified by the DB connection method is returned.

(f) Exception that occurs

None.

16.11.9 setPassword

(a) Function

Sets a password.

(b) Format

```
public void setPassword (String password)
```

(c) Argument

String `password`

Specifies a password.

(d) Return value

None.

(e) Functional detail

Sets a password.

You can specify the password using an argument of the `DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method (which are referred to generically as the DB connection methods).

If this method is used to set a password, and if a DB connection method that has an authorization identifier and a password set as arguments is also called, the password setting specified by the DB connection method takes precedence.

(f) Exception that occurs

None.

16.11.10 getPassword

(a) Function

Acquires a password.

(b) Format

```
public String getPassword()
```

(c) Argument

None.

(d) Return value

String

This is a password. If none is set, `null` is returned.

(e) Functional detail

Returns the password specified by the `setPassword` method.

If the `setPassword` method is used to set a password, and if a DB connection method (`DataSource.getConnection` method, `ConnectionPoolDataSource.getPooledConnection` method, or `XADataSource.getXAConnection` method) that has an authorization identifier and a password set as arguments is also called, the password setting specified by the DB connection method is returned.

(f) Exception that occurs

None.

16.11.11 setXAOpenString

(a) Function

Sets an `XA` open character string.

(b) Format

```
public void setXAOpenString (String xa_string)
```

(c) Argument

String `xa_string`

Specifies an `XA` open character string.

(d) Return value

None.

(e) Functional detail

Sets an `XA` open character string.

This method is provided by the `JdbhDbpsvXADataSource` class only.

Specify the `XA` open character string in the format *HiRDB-environment-variable-group-identifier+HiRDB-environment-variable-group-name*. This HiRDB environment variable group identifier must be the one set in the `setDescription` method. The following shows examples.

Example 1

When setting the environment variable group name `HiRDB_ENV_GROUP` that has been registered by the tool for registering HiRDB client environment variables

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+HiRDB_ENV_GROUP");
```

Example 2

When the path of the HiRDB environment variable group name is `C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini` (∆: single-byte space)

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+C:\\Program∆Files\\HITACHI\\HiRDB
\\HiRDB.ini");
```

(f) Exception that occurs

None.

16.11.12 getXAOpenString

(a) Function

Acquires an `XA` open character string.

(b) Format

```
public String getXAOpenString()
```

(c) Argument

None.

(d) Return value

String

This is an `XA` open character string. If none is set, `null` is returned.

(e) Functional detail

Returns the `XA` open character string specified by the `setXAOpenString` method.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

None.

16.11.13 setXACloseString

(a) Function

Sets an `XA` close character string.

(b) Format

```
public void setXACloseString (String xa_string)
```

(c) Argument

String `xa_string`

Sets an `XA` close character string.

(d) Return value

None.

(e) Functional detail

Sets an `XA` close character string.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

None.

16.11.14 getXACloseString

(a) Function

Acquires an `XA` close character string.

(b) Format

```
public String getXACloseString()
```

(c) Argument

None.

(d) Return value

String

This is an `XA` close character string. If none is set, `null` is returned.

(e) Functional detail

Returns the `XA` close character string specified by the `setXACloseString` method.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

None.

16.11.15 setRMID

(a) Function

Sets an identifier for the Resource Manager.

(b) Format

```
public void setRMID (int rmid)
```

(c) Argument

`int rmid`

Specifies an identifier for the Resource Manager.

(d) Return value

None.

(e) Functional detail

Sets a positive numeric value of 1 or greater as the identifier for the Resource Manager.

If multiple Resource Managers are used, a unique identifier must be set for each Resource Manager.

If this method is not invoked, the default identifier of `1` is used.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

If the argument value is smaller than 1, the method throws an `SQLException`.

16.11.16 getRMID

(a) Function

Acquires an identifier for the Resource Manager.

(b) Format

```
public int getRMID()
```

(c) Argument

None.

(d) Return value

`int`

This is an identifier for the Resource Manager. If none is set, `1` is returned.

(e) Functional detail

Returns the Resource Manager identifier specified by the `setRMID` method.

This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

None.

16.11.17 setXAThreadMode

(a) Function

Sets a thread mode for using XA.

(b) Format

```
public void setXAThreadMode (boolean mode)
```

(c) Argument

`boolean mode`

Specifies a thread mode for using XA.

`true`: Multi-thread mode

`false`: Single-thread mode

(d) Return value

None.

(e) Functional detail

Sets a thread mode for using XA. If this method is not invoked, the default value is `false` (single-thread mode).

This method is provided by the `JdbhDbpsvXADataSource` class only.

If the XA library provided by the RM (Resource Manager) supports multi-thread and if the application operates in the multi-thread mode, this method must be invoked in the `true` setting (multi-thread mode).

(f) Exception that occurs

None.

16.11.18 getXAThreadMode

(a) Function

Acquires a thread mode for using XA.

(b) Format

```
public boolean getXAThreadMode()
```

(c) Argument

None.

(d) Return value

boolean

Specifies a thread mode for using XA.

`true`: Multi-thread mode

`false`: Single-thread mode

(e) Functional detail

Returns the thread mode for using XA, specified by the `setXAThreadMode` method. This method is provided by the `JdbhDbpsvXADataSource` class only.

(f) Exception that occurs

None.

16.11.19 setCommit_Behavior

(a) Function

Sets whether or not the following classes are to be valid after commit execution when HiRDB commits:

- `ResultSet` class
- `Statement` class, `PreparedStatement` class, and `CallableStatement` class

(b) Format

```
public void setCommit_Behavior (String type)
```

(c) Argument

String type

Sets whether or not the objects of the `Statement` class, `PreparedStatement` class, `CallableStatement` class, and `ResultSet` class remain valid even after a transaction terminates.

Specification value	ResultSet class	Statement class, PreparedStatement class, CallableStatement class
`DELETE` (default value)	Invalid[1]	Invalid[2]
`CLOSE`	Invalid[1]	Valid
`PRESERVE`	Valid[3]	Valid[3]

[1] The condition that invalidates objects of the `ResultSet` class after commit execution is that the `get`*XXX* method of the `ResultSet` class can be executed by executing the following methods of the `ResultSet` class:

- `next` method
- `first` method
- `last` method
- `absolute` method
- `relative` method

Correct execution of a method using objects of a `ResultSet` class that was invalidated is not guaranteed.

[2] Objects that are invalid after commit execution include the following:

- SQL statements precompiled by the `Connection.prepareStatement` method
- SQL statements precompiled by the `Connection.prepareCall` method

- ResultSet class objects acquired by the executeQuery method of the Statement class, PreparedStatement class. or CallableStatement class.

[3] If the version of the connected HiRDB is earlier than 07-01, using LOCK TABLE to lock the table is required.

(d) Return value

None.

(e) Functional detail

Sets whether or not the objects of the Statement class, PreparedStatement class, CallableStatement class, and ResultSet class remain valid even after the transaction terminates. If this method is not called, the default is DELETE.

Executing this method is equivalent to setting the COMMIT_BEHAVIOR property that is performed when a database is connected using DriverManager.

(f) Exception that occurs

When XADataSource is used for the connection, DELETE always results, regardless of the specified value. However, getCommit_Behavior returns the value specified in the type argument.

(g) Notes

For notes, see *Notes on COMMIT_BEHAVIOR* following Table 16-3.

16.11.20 getCommit_Behavior

(a) Function

Sets whether or not objects of the Statement class, PreparedStatement class, CallableStatement class, and ResultSet class are to be valid even after the transaction terminates.

(b) Format

```
public String getCommit_Behavior()
```

(c) Argument

None.

(d) Return value

String

Returns Delete if there is no setting of the type that determines whether or not objects of the Statement class, PreparedStatement class, `CallableStatement` class and `ResultSet` class remain valid even after the transaction ends.

(e) Functional detail

The information specified by the `setCommit_Behavior` method is returned.

(f) Exception that occurs

None.

16.11.21 setBlockUpdate

(a) Function

Sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, and `DELETE`).

(b) Format

```
public void setBlockUpdate(boolean Mode)
```

(c) Argument

`boolean Mode`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.

(d) Return value

None.

(e) Functional detail

This function sets whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

Whether or not the parameter sets are actually processed at one time depends on the method for using the facilities using arrays. For details about how to use the facilities using arrays, see *4.8 Facilities using arrays*.

(f) Exception that occurs

None.

(g) Notes

For details about how to process multiple lines of `?` parameters in batch mode, see *Table 16-3 Information to be specified for Properties info* and *16.3.2 Batch updating*.

This function can also be specified by the `HiRDB_for_Java_BLOCK_UPDATE` system property. If the `setBlockUpdate` method has been set, the `HiRDB_for_Java_BLOCK_UPDATE` system property setting is ignored.

16.11.22 getBlockUpdate

(a) Function

Acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter (`INSERT`, `UPDATE`, and `DELETE`).

(b) Format

```
public boolean getBlockUpdate()
```

(c) Argument

None.

(d) Return value

`boolean`

Specifies whether or not multiple parameter sets are to be processed at one time. When this information is omitted, `false` is assumed.

`true`

Processes multiple parameter sets at one time.

`false`

Processes one parameter set at a time.

(e) Functional detail

This function acquires a value indicating whether or not multiple parameter sets are to be processed at one time during database updating using the `?` parameter.

(f) Exception that occurs

None.

(g) Notes

None.

16.11.23 setLONGVARBINARY_Access

(a) Function

Specifies the database access method for LONGVARBINARY (column attribute is BLOB or BINARY).

(b) Format

```
public void setLONGVARBINARY_Access(String Mode)
```

(c) Argument

String Mode

Specifies the database access method for LONGVARBINARY (column attribute is BLOB or BINARY). When this argument is omitted, "REAL" is assumed.

"REAL"

Accesses real data.

"LOCATOR"

Uses HiRDB's locator facility to access data.

Other:

Assumes that "REAL" has been specified.

(d) Return value

None.

(e) Functional detail

Specifies the database access method for LONGVARBINARY (column attribute is BLOB or BINARY).

(f) Exception that occurs

None.

16.11.24 getLONGVARBINARY_Access

(a) Function

Acquires the database access method for LONGVARBINARY (column attribute is BLOB or BINARY).

(b) Format

```
public String getLONGVARBINARY_Access()
```

(c) Argument

None.

(d) Return value

`String`

Indicates the information set as the database access method for `LONGVARBINARY` (column attribute is `BLOB` or `BINARY`). When no information has been set, `"REAL"` is assumed.

`"REAL"`

Accesses real data.

`"LOCATOR"`

Uses HiRDB's locator facility to access data.

(e) Functional detail

Returns the information specified by the `setLONGVARBINARY_Access` method.

(f) Exception that occurs

None.

16.11.25 setSQLInNum

(a) Function

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLInNum(int inNum)
```

(c) Argument

`int inNum`:

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed. The permitted value range is from `1` to `30,000` (default is `64`).

(d) Return value

None.

(e) Functional detail

Specifies the maximum number of input or input/output `?` parameters to be acquired during SQL preprocessing.

If the actual number of `?` parameters is greater than this property value, this method acquires information about the input or input/output `?` parameters after SQL preprocessing.

The value specified in this method is used as the value of `HiRDB_for_Java_SQL_IN_NUM` property during database connection.

(f) Exception that occurs

If the specified argument value falls beyond the permitted range, the method throws an `SQLException`.

(g) Notes

- This function can also be specified by the `HiRDB_for_Java_SQL_IN_NUM` system property. If the `setSQLInNum` method has been set, the `HiRDB_for_Java_SQL_IN_NUM` system property setting is ignored.
- If you do not execute any SQL statement that uses input or input/output `?` parameters, we recommend that you specify a value of `1`.

16.11.26 getSQLInNum

(a) Function

Specifies the maximum number of input or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLInNum`.

(b) Format

```
public int getSQLInNum()
```

(c) Argument

None.

(d) Return value

`int`

This is the maximum number of input or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLInNum`. If no value has been set, the method returns the default value (`64`).

(e) Functional detail

Acquires the maximum number of input or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLInNum`.

(f) Exception that occurs

None.

16.11.27 setSQLOutNum

(a) Function

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLOutNum(int outNum)
```

(c) Argument

`int outNum`

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed. The permitted value range is from `1` to `30,000` (default is `64`).

(d) Return value

None.

(e) Functional detail

Specifies the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed.

This specification is used as the number of output items that are to be acquired during SQL preprocessing.

If the number of actual output items is greater than the value of this property, the method acquires information about the output items after SQL preprocessing.

The value specified in this method is used as the value of the `HiRDB_for_Java_SQL_OUT_NUM` property during database connection.

(f) Exception that occurs

If the specified argument value falls beyond the permitted range, the method throws an `SQLException`.

(g) Notes

- This function can also be specified by the `HiRDB_for_Java_SQL_OUT_NUM` system property. If the `setSQLOutNum` method has been set, the `HiRDB_for_Java_SQL_OUT_NUM` system property setting is ignored.
- If there is no search item, output `?` parameter, or input/output `?` parameter, we

recommend that you specify a value of `1`.

16.11.28 getSQLOutNum

(a) Function

Acquires the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLOutNum`.

(b) Format

```
public int getSQLOutNum()
```

(c) Argument

None.

(d) Return value

`int`

This is the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLOutNum`. If this value has not been set, the method returns the default value (`64`).

(e) Functional detail

Acquires the maximum number of search items, output `?` parameters, or input/output `?` parameters in the SQL statements to be executed that has been set by `setSQLOutNum`.

(f) Exception that occurs

None.

16.11.29 setSQLWarningLevel

(a) Function

Specifies the warning retention level that occurred during execution of SQL statements.

(b) Format

```
public void setSQLWarningLevel (String warningLevel)
```

(c) Argument

`String warningLevel`

Specifies the retention level of warning information that has been issued during execution of SQL statements. The permitted warning retention levels are listed below. For details about the relationship between the specified value and the retained warning, see *16.2.9 SQLWarning class*.

- `IGNORE`
- `SQLWARN` (default)
- `ALLWARN`

The value specified in the argument of this method is not case sensitive.

(d) Return value

None.

(e) Functional detail

Specifies the retention level of warning information that has been issued during execution of SQL statements.

The value specified in this method is used as the value of the `HiRDB_for_Java_SQLWARNING_LEVEL` property during database connection.

(f) Exception that occurs

If the specified argument value is invalid, the method throws an `SQLException`.

16.11.30 getSQLWarningLevel

(a) Function

Acquires the warning retention level specified in `setSQLWarningLevel`.

(b) Format

```
public String getSQLWarningLevel ()
```

(c) Argument

None.

(d) Return value

`String`

Returns the warning retention level set by `setSQLWarningLevel` (`IGNORE`, `SQLWARN`, or `ALLWARN`). For details about the relationship between the returned value and the retained warning, see *16.2.9 SQLWarning class*.

(e) Functional detail

Acquires the warning retention level specified in `setSQLWarningLevel`. If this information has not been set, the method returns the default value (SQLWARN).

(f) Exception that occurs

None.

16.11.31 setClear_Env

(a) Function

Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection.

(b) Format

```
public void setClear_Env(boolean Mode)
```

(c) Argument

`boolean Mode`

Specifies whether or not the HiRDB client environment definition is to be ignored.

`true`: Ignores.

`false`: Does not ignore.

(d) Return value

None.

(e) Functional detail

Specifies whether or not the HiRDB client environment definition set as OS environment variables is to be ignored during database connection.

The value specified in this method is equivalent to the `HiRDB_for_Java_CLEAR_ENV` property setting that is specified during database connection.

(f) Exception that occurs

None.

(g) Notes

For details, see *HiRDB_for_Java_CLEAR_ENV* in *Table 16-3 Information to be specified for Properties info*.

16.11.32 getClear_Env

(a) Function

Acquires the environment variable invalidation setting specified by `setClear_Env`.

(b) Format

```
public boolean getClear_Env()
```

(c) Argument

None.

(d) Return value

`String`

Returns the environment variable invalidation setting specified by `setClear_Env`.

`true`

Ignores the HiRDB client environment definition set as OS environment variables during database connection.

`false`

Does not ignore the HiRDB client environment definition set as OS environment variables during database connection.

(e) Functional detail

Acquires the environment variable invalidation setting specified by `setClear_Env`. If this setting has not been specified, the method returns the default value (`false`).

(f) Exception that occurs

None.

16.12 Data types and character codes

16.12.1 Data types

JDBC's SQL data types and the SQL data types connected via a HiRDB client library do not match perfectly. The JDBC driver maps JDBC's SQL data types and HiRDB's SQL data types. If an unmappable SQL data type is used for data access, the JDBC driver throws an `SQLException`.

The SQL data types are mapped with the `get`*XXX* and `set`*XXX* methods in the `ResultSet`, `PreparedStatement`, and `CallableStatement` classes. For the SQL data types and the `get`*XXX* and `set`*XXX* method mapping rules, see the documentation for the JDBC1.0 standard.

Table 16-19 shows the correspondence of SQL data types between HiRDB and JDBC.

Table 16-19: Correspondence of SQL data types between HiRDB and JDBC

HiRDB's SQL data type	JDBC's SQL data type
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL	DECIMAL
FLOAT, DOUBLE PRECISION	FLOAT
SMALLFLT, REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	CHAR
NVARCHAR	VARCHAR
MCHAR	CHAR
MVARCHAR	VARCHAR
DATE	DATE
TIME	TIME
BLOB	LONGVARBINARY
TIMESTAMP	TIMESTAMP
BINARY*	LONGVARBINARY

* Data is handled in the same way as BLOB.

16.12.2 Character code conversion facility

In a Java program, Unicode is used for the character codes. Therefore, the JDBC driver performs mutual character code conversion between the HiRDB character data and the Unicodes. For this character code conversion processing, the JDBC driver uses the encoder and decoder provided by the Java Virtual Machine. At this time, ENCODELANG of Properties info specifies the character set names specified by the JDBC driver for the encoder and decoder that are provided by the Java Virtual Machine.

Tables 16-20 and 16-21 show the correspondences between the HiRDB character codes and the Java character sets.

Table 16-20: Correspondence between HiRDB character codes and Java character sets (UNIX)

HiRDB character codes	Character set	Remarks
sjis (Shift JIS kanji)	"SJIS"	Double-byte characters include external characters.
ujis (EUC Japanese kanji)	"EUC_JP" (Japanese EUC)	Double-byte characters do not include external characters*
chinese (EUC Chinese kanji)	"EUC_CN" (Simplified Chinese)	Double-byte characters do not include external characters*
lang-c (8-bit codes)	"ISO-8859-1" (ISO Latin-1)	Can be used with US ASCII and 8-bit codes.
UTF-8	UTF-8	None

Note

If ENCODELANG of Properties info is set using the following methods, this setting takes precedence for encoding.

- Set using Properties info passed as the argument of the DriverManager.getConnection method
- Set using the JdbhDataSource.setEncodLang method, rce.setEncodLang method, or JdbhXADataSource method

For details about operation when ENCODELANG is not set using the above methods or when OFF is set, see *16.11.5 setEncodeLang*.

* You cannot use external character codes assigned to EUC code set 3 (character codes expressed by three bytes in the range of $(8F)^{16}$ to $(XXXX)^{16}$.

Table 16-21: Correspondence between HiRDB character codes and Java Character sets (Windows)

HiRDB character codes	Character set	Remarks
`sjis` (Shift JIS kanji)	MS932 when the Java Virtual Machine standard encoding is MS932; otherwise, it is SJIS.	Double-byte characters include external characters.
UTF-8	UTF-8	None

Note

If `ENCODELANG` of `Properties info` is set using the following methods, this setting takes precedence for encoding:

- Set using `Properties info` passed as the argument of the `DriverManager.getConnection` method
- Set using the `JdbhDataSource.setEncodLang` method, `JdbhDataSource.setEncodLang` method, or `JdbhXADataSource` method.

For details about operation when `ENCODELANG` is not set using the above methods or when `OFF` is set, see *16.11.5 setEncodeLang*.

16.13 Classes and methods with limitations

This section explains the classes defined in the JDBC1.0 standard.

The JDBC driver does not support the following classes that are defined in the JDBC2.0 basic standard:

- `Clob` class
- `Struct` class
- `Ref` class
- `SQLData` class
- `SQLInput` class
- `SQLOutput` class

16.13.1 Driver class

There is no limitation to this class.

16.13.2 Connection class

Table 16-22 lists limitations to the methods in the `Connection` class that are defined in the JDBC1.0 standard, while Table 16-23 lists limitations to the methods added in the JDBC2.0 basic standard.

Table 16-22: Limitations to the methods in the Connection class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
`setReadOnly`	Not usable.
`isReadOnly`	Unconditionally returns `false`.
`setCatalog`	Not usable.
`getCatalog`	Returns `null` unconditionally.
`setTransactionIsolation`	Not usable.
`getTransactionIsolation`	Returns `TRANSACTION_REPEATABLE_READ` unconditionally.

Table 16-23: Limitations to the methods in the Connection class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
createStatement	A result set reflecting updating results is not usable. Therefore, if TYPE_SCROLL_SENSITIVE is specified for the result set type, the method changes it to TYPE_SCROLL_INSENSITIVE and sets an SQLWarning.
prepareStatement	
prepareCall	
getTypeMap	Unconditionally throws SQLException because a user-defined type is not usable.
setTypeMap	

16.13.3 Statement class

Table 16-24 lists limitations to the methods in the `Connection` class that are defined in the JDBC1.0 standard, while Table 16-25 lists limitations to the methods added in the JDBC2.0 basic standard.

Table 16-24: Limitations to the methods in the Statement class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
setCursorName	Not usable (because positioned updating or deletion is not available).
getMaxFieldSize	Returns the value specified with setMaxFieldSize.
getMoreResults	Unconditionally returns false.
setMaxRows	Not usable.
setQueryTimeout	

Table 16-25: Limitations to the methods in the Statement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
setFetchDirection	Throws SQLException if anything other than FETCH_FORWARD is specified.
getFetchSize	Returns the value specified with the setFetchSize method.

16.13.4 PreparedStatement class

Table 16-26 lists limitations to the methods in the `PreparedStatement` class that are added in the JDBC2.0 basic standard.

Table 16-26: Limitations to the methods in the PreparedStatement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
`setBlob`	For the JDBC driver, the method treats the JDBC SQL type as `LONGVARBINARY`.
`setClob`	Unconditionally throws `SQLException` because the SQL `CLOB` type is not available.
`setRef`	Unconditionally throws `SQLException` because the SQL structured type is not available.
`setNull`	If the complete name of an SQL user-defined type is specified, the method unconditionally throws `SQLException` because the SQL structured type or the SQL array type is not available.
`setObject`	Ignores the specified `scale` and obtains the value of `scale` from the actual value specified.

16.13.5 CallableStatement class

Table 16-27 lists limitations to the methods in the `CallableStatement` class that are added in the JDBC2.0 basic standard.

Table 16-27: Limitations to the methods in the CallableStatement class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
`getObject`	If `Map` is specified, the method throws `SQLException` because the Map specification is not available.
`getBlob`	For the JDBC driver, the method treats the JDBC SQL type as `LONGVARBINARY`.
`getClob`	Unconditionally throws `SQLException` because the SQL `CLOB` type is not available.
`getRef`	Unconditionally throws `SQLException` because the SQL structured type is not available.

16.13.6 ResultSet class

Table 16-28 lists limitations to the methods in the `ResultSet` class that are added in the JDBC2.0 basic standard.

Table 16-28: Limitations to the methods in the ResultSet class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation
`setFetchDirection`	Throws `SQLException` if anything other than `FETCH_FORWARD` is specified.
`rowUpdated`	Unconditionally throws `SQLException` because an updatable result set is not available.
`rowInserted`	
`rowDeleted`	
`updateNull`	
`updateBoolean`	
`updateByte`	
`updateShort`	
`updateInt`	
`updateLong`	
`updateFloat`	
`updateDouble`	
`updateBigDecimal`	
`updateString`	
`updateBytes`	
`updateDate`	
`updateTime`	
`updateTimestamp`	
`updateAsciiStream`	
`updateBinaryStream`	
`updateCharacterStream`	

Method added in JDBC2.0 basic standard	Limitation
`updateObject`	
`insertRow`	
`updateRow`	
`deleteRow`	
`refreshRow`	
`cancelRowUpdates`	
`moveToInsertRow`	
`moveToCurrentRow`	
`getObject`	If `Map` is specified, the method throws `SQLException` because the Map specification is not available.
`getBlob`	For the JDBC driver, the method treats the JDBC SQL type as `LONGVARBINARY`.
`getClob`	Unconditionally throws `SQLException` because the SQL `CLOB` type is not available.
`getRef`	Unconditionally throws `SQLException` because the SQL structured type is not available.

16.13.7 ResultSetMetaData class

Table 16-29 lists limitations to the methods in the `ResutlSetMetaData` class that are defined in the JDBC1.0 standard. However, for details about the return value of each method of the `MetaData` class acquired from the result set generated by the `getResultSet` method of the `Array` class, see *Table 16-14*.

Table 16-29: Limitations to the methods in the ResultSetMetaData class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation
`isAutoIncrement`	Unconditionally returns `false`.
`isCaseSensitive`	Unconditionally returns `true`.
`isCurrency`	Unconditionally returns `false`.
`getColumnLabel`	Returns a column name because the column label (column header) is not available.

Method defined in JDBC1.0 standard	Limitation
getSchemaName	Unconditionally returns null.
getTableName	
getCatalogName	
isReadOnly	Unconditionally returns false.
isWritable	
isDefinitelyWritable	

16.13.8 DatabaseMetaData class

Table 16-30 lists limitations to the returned contents of methods in the DatabaseMetaData class that are defined in the JDBC1.0 standard, while Table 16-31 lists limitations to the returned contents of the methods added by the JDBC2.0 basic standard. Note that the value returned by each method is information related to the HiRDB server, whose version has to be the same as the JDBC driver being used.

Table 16-30: Limitations to the methods in the DatabaseMetaData class that are defined in the JDBC1.0 standard

Method defined in JDBC1.0 standard	Limitation or return value
allProceduresAreCallable	Returns false.
allTablesAreSelectable	Returns false.
getURL	Returns the JDBC URL of the connected database.
getUserName	Returns the authorization identifier used when connecting to the database.
isReadOnly	Unconditionally returns false because the access mode cannot be changed.
nullsAreSortedHigh	Returns true.
nullsAreSortedLow	Returns false.
nullsAreSortedAtStart	Returns false.
nullsAreSortedAtEnd	Unconditionally returns false.
getDatabaseProductName	Returns HiRDB.
getDatabaseProductVersion	Returns null.
getDriverName	Returns HiRDB_for_JDBC.

Method defined in JDBC1.0 standard	Limitation or return value
getDriverVersion	Returns 08.00.0000.
getDriverMajorVersion	8
getDriverMinorVersion	0
usesLocalFiles	Unconditionally returns false.
usesLocalFilePerTable	Unconditionally returns false.
supportsMixedCaseIdentifiers	Unconditionally returns false.
storesUpperCaseIdentifiers	Returns true.
storesLowerCaseIdentifiers	Unconditionally returns false.
storesMixedCaseIdentifiers	Returns false.
supportsMixedCaseQuotedIdentifiers	Returns true.
storesUpperCaseQuotedIdentifiers	Returns false.
storesLowerCaseQuotedIdentifiers	Unconditionally returns false.
storesMixedCaseQuotedIdentifiers	Returns true.
getIdentifierQuoteString	Unconditionally returns a quotation mark.
getSQLKeywords	Returns a HiRDB-specific SQL keyword.
getNumericFunctions	Returns a list of mathematical functions.
getStringFunctions	Returns a list of character string functions.
getSystemFunctions	Returns a list of system functions.
getTimeDateFunctions	Returns a list of time and date functions.
getSearchStringEscape	Returns \.
getExtraNameCharacters	Returns a special character that can be used as an SQL identification name.
supportsAlterTableWithAddColumn	Returns true.
supportsAlterTableWithDropColumn	
supportsColumnAliasing	
nullPlusNonNullIsNull	
supportsConvert (no argument)	Returns true.

Method defined in JDBC1.0 standard	Limitation or return value
`supportsConvert` (with arguments)	Returns either `true` or `false` depending on the combination of data types specified in arguments.
`supportsTableCorrelationNames`	Returns `true`.
`supportsDifferentTableCorrelationNames`	
`supportsExpressionsInOrderBy`	Returns `false`.
`supportsOrderByUnrelated`	Returns `true`.
`supportsGroupBy`	
`supportsGroupByUnrelated`	
`supportsGroupByBeyondSelect`	
`supportsLikeEscapeClause`	
`supportsMultipleResultSets`	Unconditionally returns `true`.
`supportsMultipleTransactions`	
`supportsNonNullableColumns`	Returns `true`.
`supportsMinimumSQLGrammar`	Unconditionally returns `true`.
`supportsCoreSQLGrammar`	
`supportsExtendedSQLGrammar`	Returns `false`.
`supportsANSI92EntryLevelSQL`	Unconditionally returns `true`.
`supportsANSI92IntermediateSQL`	Unconditionally returns `false`.
`supportsANSI92FullSQL`	
`supportsIntegrityEnhancementFacility`	Returns `false`.
`supportsOuterJoins`	Returns `true`.
`supportsFullOuterJoins`	Returns `false`.
`supportsLimitedOuterJoins`	Returns `true`.
`getSchemaTerm`	Returns `schema`.
`getProcedureTerm`	Returns `procedure`.
`getCatalogTerm`	Returns `null`.
`isCatalogAtStart`	Returns `false`.

Method defined in JDBC1.0 standard	Limitation or return value
`getCatalogSeparator`	Returns `null`.
`supportsSchemasInDataManipulation`	Unconditionally returns `true`.
`supportsSchemasInProcedureCalls`	Returns `true`.
`supportsSchemasInTableDefinitions`	
`supportsSchemasInIndexDefinitions`	
`supportsSchemasInPrivilegeDefinitions`	
`supportsCatalogsInDataManipulation`	Returns `false`.
`supportsCatalogsInProcedureCalls`	
`supportsCatalogsInTableDefinitions`	
`supportsCatalogsInIndexDefinitions`	Unconditionally returns `false`.
`supportsCatalogsInPrivilegeDefinitions`	
`supportsPositionedDelete`	
`supportsPositionedUpdate`	
`supportsSelectForUpdate`	
`supportsStoredProcedures`	Returns `true`.
`supportsSubqueriesInComparisons`	
`supportsSubqueriesInExists`	
`supportsSubqueriesInIns`	
`supportsSubqueriesInQuantifieds`	
`supportsCorrelatedSubqueries`	
`supportsUnion`	
`supportsUnionAll`	
`supportsOpenCursorsAcrossCommit`	Returns `true` if any of the following values is `PRESERVE`: • Setting of `COMMIT_BEHAVIOR` in URL • Setting of `COMMIT_BEHAVIOR` in `Properties info` • Argument when the `setCommit_Behavior` method is executed

Method defined in JDBC1.0 standard	Limitation or return value
`supportsOpenCursorsAcrossRollback`	Unconditionally returns `false`.
`supportsOpenStatementsAcrossCommit`	Returns `true` if any of the following values is `PRESERVE` or `CLOSE`: • Setting of `COMMIT_BEHAVIOR` in URL • Setting of `COMMIT_BEHAVIOR` in `Properties info` • Argument when the `setCommit_Behavior` method is executed
`supportsOpenStatementsAcrossRollback`	Unconditionally returns `false`.
`getMaxBinaryLiteralLength`	Returns a value of `64000`.
`getMaxCharLiteralLength`	Returns a value of `32000`.
`getMaxColumnNameLength`	Returns a value of `30`.
`getMaxColumnsInGroupBy`	Returns a value of `255`.
`getMaxColumnsInIndex`	Returns a value of `16`.
`getMaxColumnsInOrderBy`	Returns a value of `255`.
`getMaxColumnsInSelect`	Returns a value of `30000`.
`getMaxColumnsInTable`	
`getMaxConnections`	Returns a value of `0`.
`getMaxCursorNameLength`	Returns a value of `30`.
`getMaxIndexLength`	Returns a value of `4036`.
`getMaxSchemaNameLength`	Returns a value of `8`.
`getMaxProcedureNameLength`	Returns a value of `30`.
`getMaxCatalogNameLength`	Returns a value of `0`.
`getMaxRowSize`	
`doesMaxRowSizeIncludeBlobs`	Returns `false`.
`getMaxStatementLength`	Returns a value of `2000000`.
`getMaxStatements`	Returns a value of `64`.
`getMaxTableNameLength`	Returns a value of `30`.
`getMaxTablesInSelect`	Returns a value of `64`.

Method defined in JDBC1.0 standard	Limitation or return value
getMaxUserNameLength	Returns a value of 8.
getDefaultTransactionIsolation	Unconditionally returns TRANSACTION_REPEATABLE_READ.
supportsTransactions	Unconditionally returns true.
supportsTransactionIsolationLevel	Returns true when the given transaction isolation level is any of the following: • TRANSACTION_READ_COMMITTED • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_REPEATABLE_READ
SupportsDataDefinitionAndDataManipulationTransactions	Returns false.
supportsDataManipulationTransactionsOnly	Returns false.
dataDefinitionCausesTransactionCommit	Returns true.
dataDefinitionIgnoredInTransactions	Unconditionally returns false.
getProcedures	Returns information about the Java stored routines.
getProcedureColumns	Returns information about the parameters of the Java stored routines.
getTables	Returns information about tables. Only the table types returned by getTableTypes can be specified in the list of table types to be obtained (types).
getSchemas	Returns information about schemas.
getCatalogs	Always returns a 0 result.
getTableTypes	Returns information about table types. The following values are returned: "SYSTEM TABLE": System table "BASE TABLE": Base table "VIEW": View table "READ ONLY VIEW": Read-only view table "ALIAS": Another table
getColumns	Returns information about columns.
getColumnPrivileges	Returns information about column privileges.
getTablePrivileges	Returns information about table privileges.

Method defined in JDBC1.0 standard	Limitation or return value
getBestRowIdentifier	Always returns a 0 result.
getVersionColumns	
getPrimaryKeys	Returns information about primary key columns (always returns a 0 result).
getImportedKeys	Always returns a 0 result.
getExportedKeys	Returns information about external key columns that reference the primary key columns (always returns a 0 result).
getCrossReference	Returns information about the external key columns in the table with external keys that reference the primary key columns in the table with the primary key (always returns a 0 result).
getTypeInfo	Returns information about the standard SQL types supported for the database.
getIndexInfo	Returns information about indexes.

Table 16-31: Limitations to the methods in the DatabaseMetaData class that are added in the JDBC2.0 basic standard

Method added in JDBC2.0 basic standard	Limitation or return value
supportsResultSetType	Returns true if the result set type is TYPE_FORWARD_ONLY or TYPE_SCROLL_INSENSITIVE.
SupportsResultSet Concurrency	Returns true if the result set type is TYPE_FORWARD_ONLY or TYPE_SCROLL_INSENSITIVE and the parallel processing type is CONCUR_READ_ONLY.
ownUpdatesAreVisible	Unconditionally returns false.
ownDeletesAreVisible	
ownInsertsAreVisible	
othersUpdatesAreVisible	
othersDeletesAreVisible	
othersInsertsAreVisible	
updatesAreDetected	
deletesAreDetected	

Method added in JDBC2.0 basic standard	Limitation or return value
insertsAreDetected	
supportsBatchUpdates	Unconditionally returns `true`.
getUDTs	Always returns a `0` result.
getConnection	Returns the `Connection` instance that is the `DatabaseMetaData` instance generation source.

16.13.9 Blob class

Table 16-32 lists limitations to the methods in the `Blob` class that are added in the JDBC2.0 basic standard.

Table 16-32: Limitations to the methods added by JDBC2.0 basic standards for Blob class

Method added by JDBC2.0 basic standard	Limitation
setBinaryStream	Cannot be used for JDBC1.4 methods. If used, the method unconditionally throws an `SQLException`.
setBytes	
truncate	

16.13.10 Array class

Table 16-33 lists limitations to the methods in the `Array` class that are added by the JDBC2.0 basic standard.

Table 16-33: Restrictions on the methods added by the JDBC2.0 basic specification for the Array class

Methods added in the JDBC2.0 basic specification	Restrictions
getArray	Because `MAP` cannot be used, the method throws an `SQLException` if `MAP` is specified for the argument.
getResultSet	

Chapter

17. Type4 JDBC Driver

This chapter explains the Type4 JDBC driver installation, environment setup, and JDBC functions. Note that the Type4 JDBC driver cannot be used in the Linux for AP8000 version of a client.

Hereafter in this chapter, the Type4 JDBC driver is referred to as the *JDBC driver*.

17.1 Installation and environment setup

17.1.1 Installation

The JDBC driver can be installed when you install HiRDB. After the driver is installed, the file configuration is as follows:

For UNIX

```
HiRDB/client/lib/pdjdbc2.jar
```

For Windows (HiRDB server product)

```
HiRDB\client\utl\pdjdbc2.jar
```

For Windows (HiRDB/Run Time or HiRDB/Developer's Kit)

```
HiRDB\utl\pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

17.1.2 Environment setup

Before you use the JDBC driver to execute UAPs, you must specify the installed file in the OS's `CLASSPATH` environment variable. Also, before you compile a UAP you must set up the `CLASSPATH` environment variable in order to directly manipulate the classes provided by the JDBC driver, which is necessary for the methods provided by the JDBC driver that do not comply with the JDBC standards.

If you are using the JDBC driver from an application server, such as Cosminexus, the environment setup depends on the environment setup for the application server. Refer to the documentation for the particular application server, and check the specifications.

(1) UNIX environment

(a) Bourne shell

```
CLASSPATH=${CLASSPATH}:/HiRDB/client/lib/pdjdbc2.jar
export CLASSPATH
```

Note

The underlined portion indicates the HiRDB installation directory.

(b) C shell

```
setenv CLASSPATH ${CLASSPATH}:/HiRDB/client/lib/pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

(2) Windows environment (executing the program from the command prompt)

```
set CLASSPATH=%CLASSPATH%;C:\Program Files\HITACHI\HiRDB\client\utl\pdjdbc2.jar
```

Note

The underlined portion indicates the HiRDB installation directory.

17.1.3 Abbreviation of methods

- This manual uses the notation `get`*XXX method* to represent the following methods generically:

  `getArray`, `getAsciiStream`, `getBigDecimal`, `getBinaryStream`, `getBlob`, `getBoolean`, `getByte`, `getBytes`, `getCharacterStream`, `getClob`, `getDate`, `getDouble`, `getFloat`, `getInt`, `getLong`, `getObject`, `getRef`, `getShort`, `getString`, `getTime`, and `getTimestamp`

- This manual uses the notation `set`*XXX method* to represent the following methods generically:

  `setArray`, `setAsciiStream`, `setBigDecimal`, `setBinaryStream`, `setBlob`, `setBoolean`, `setByte`, `setBytes`, `setCharacterStream`, `setClob`, `setDate`, `setDouble`, `setFloat`, `setInt`, `setLong`, `setNull`, `setObject`, `setRef`, `setShort`, `setString`, `setTime`, and `setTimestamp`

- This manual uses the notation `execute`*XXX method* to represent the following methods generically:

  `execute`, `executeBatch`, `executeQuery`, and `executeUpdate`

- This manual uses the notation `DataSource`-*type interface* to represent the following interfaces generically:

  `DataSource`, `ConnectionPoolDataSource`, and `XADataSource`

17.2 Database connection using the DriverManager class

The procedure for connecting from the `DriverManager` class to HiRDB and generating an instance of the `Connection class` is as follows:

1. Register the `Driver` class into the Java Virtual Machine.
2. Set the connection information in the arguments, and use the `getConnection` method of the `DriverManager` class to connect to HiRDB.

17.2.1 Registering the Driver class

The procedure for registering the JDBC driver into the Java Virtual Machine is described below.

The driver name that must be used to register the `Driver` class into the Java Virtual Machine is *package-name*`.`*class-name*. The package and class names of the JDBC driver are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `HiRDBDriver`

(1) Registering using the forName method of the Class class

Call the `forName` method of the `Class` class from within the application as follows:

```
Class.forName("JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(2) Registering in the system properties

Set the following value in the `jdbc.drivers` system property of the Java Virtual Machine:

```
System.setProperty("jdbc.drivers", "JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver");
```

(3) Registering into the operation setup file of the Java Virtual machine (Applet)

Specify in the `[JAVA_HOME]\.hotjava\properties` file the information shown below (the value of `[JAVA_HOME]` depends on the Java execution environment). If you register multiple JDBC drivers, delimit them with colons (`:`).

```
jdbc.drivers="JP.co.Hitachi.soft.HiRDB.JDBC.HiRDBDriver"
```

17.2.2 Connecting to HiRDB with the getConnection method

The `getConnection` method of the `DriverManager` class is provided in the following three formats, each with its own set of arguments:

- `public static Connection getConnection(String url)`
- `public static Connection getConnection(String url, String user, String password)`
- `public static Connection getConnection(String url, Properties info)`

The arguments (`url`, `user`, `password`, and `info`) in these method formats specify connection information that is needed in order to connect to HiRDB.

When connection to HiRDB is established successfully, the JDBC driver returns a reference to a `Connection` class instance as the result of calling the method. However, the method throws an `SQLException` in the following cases:

- The required connection information is not specified in an argument.
- Specified connection information is invalid.
- Connection cannot be established (for example, because HiRDB has not been started at the connection destination).

Table 17-1 describes the details of specifying the `getConnection` method arguments.

Table 17-1: Specification details of the getConnection method arguments

Argument	Specification details
`String url`	Specifies the URL. For details, see *(1) URL syntax*.
`String user`	Specifies the authorization identifier. If the null value is specified, the JDBC driver assumes that no authorization identifier has been specified. If the character string has a length of `0`, the method throws an `SQLException` and `user` is set to *aa....aaa*, which are characters embedded in the KFPJ20212-E message. For details about the specification priorities, see *17.11 Connection information priorities*.
`String password`	Specifies the password. For details about the specification priorities, see *17.11 Connection information priorities*. If the null value is specified, or if a character string of length 0 is specified, the JDBC driver assumes that no password was specified.
`Properties info`	Specifies various connection information items. For details, see *(2) User properties*.

(1) URL syntax

This section explains the URL syntax supported by the JDBC driver.

You must not specify any spaces within an item or between items in a URL. Note that the item names are case sensitive.

(a) URL syntax

```
jdbc:hitachi:hirdb[://[DBID=additional-connection-information]
                   [,DBHOST=database-host-name]
                   [,ENCODELANG=conversion-character-set]
                   [,HIRDB_CURSOR=cursor-operation-mode]
                  [,STATEMENT_COMMIT_BEHAVIOR=Statement-object-status-after-commit-execution]]
```

(b) Explanation of URL items

`jdbc:hitachi:hirdb`

This item consists of the protocol name, subprotocol name, and subname. You must specify this item. This item is case sensitive.

`DBID=`*additional-connection-information*

Specifies the port number of the HiRDB server (corresponds to the `PDNAMEPORT` value in the client environment definitions). You can also specify a HiRDB environment variable group for this item.

If no port number is specified for the HiRDB server, one of the following values becomes effective:

- The `PDNAMEPORT` value in the HiRDB client environment variables specified by `HiRDB_for_Java_ENV_VARIABLES` in the `Properties` argument of the `getConnection` method
- The `PDNAMEPORT` value in the environment variable group specified by `DBID` in the URL

For details about the specification priorities, see *17.11 Connection information priorities*.

If neither value is specified, the `getConnection` method throws an `SQLException` when it executes.

Notes

You should note the following points about specifying an HiRDB environment variable group for the additional connection information:

- When you specify the name of the HiRDB environment variable group, specify `@HIRDBENVGRP=` followed by the absolute path name. If no value is specified after the equal sign, such as `@HIRDBENVGRP=,`, the JDBC driver assumes that no value is specified for this item.

- Note that an environment variable group name is case sensitive. Also, the environment variable group name depends on the OS.
- If the environment variable group name contains any single-byte space or single-byte `@` characters, you must enclose the name in single-byte double quotation marks (`"`). When an environment variable group name is enclosed in single-byte double quotation marks, the characters from the closing single-byte quotation mark to the next setting item or to the final character are ignored. Note that an environment variable group that includes a single-byte quotation mark or a single-byte comma cannot be specified.

  Below are examples of specifications that trigger an error:

```
@ΔHIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRPΔ=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=Δ/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.iniΔ
```

Note: Δ represents a single-byte space character.

`DBHOST=`*database-host-name*

Specifies the name of the HiRDB host.

When this specification is omitted, one of the following values becomes effective:

- The `PDHOST` value in the HiRDB client environment variables that were specified by `HiRDB_for_Java_ENV_VARIABLES` in the `Properties` argument of the `getConnection` method
- The `PDHOST` value in the HiRDB environment variable group that was specified by `DBID` in the URL

For details about the specification priorities, see *17.11 Connection information priorities*.

If neither value is specified, the `getConnection` method throws an `SQLException` when it executes.

`ENCODELANG=`*conversion-character-set*

Specifies the conversion character set for the HiRDB character codes of the connection destination when the JDBC driver uses the `String` class to exchange data with HiRDB. Select a specifiable conversion character set from the encoding list shown under *Internationalization* in the *Java™ 2 SDK, Standard Edition* documentation.

Table 17-2 lists the character codes of HiRDB and their corresponding conversion character sets.

Table 17-2: HiRDB character codes and corresponding conversion character sets

HiRDB character codes (character code set with pdntenv or pdsetup command)	Conversion character set to be specified
lang-c	ISO8859_1
sjis	SJIS or MS932#
ujis	EUC_JP
utf-8	UTF-8
chinese	EUC_CN

#

The specification of SJIS or MS932 depends on the handling of Windows special characters in the application.

When OFF is specified, the JDBC driver operates assuming that the conversion character set for the HiRDB character codes shown in Table 17-2 was specified. If the HiRDB character code set is sjis, the conversion character set determined by the OS running the JDBC driver is as follows:

For UNIX: SJIS

For Windows: MS932

Note that the specification is case sensitive (except for OFF).

If a conversion character set that is not supported by the Java Virtual Machine is specified, the JDBC driver throws an SQLException during connection with the HiRDB server.

If this specification is omitted, the JDBC driver converts characters using the appropriate conversion character set shown in Table 17-2. However, if one of the following is specified, the JDBC driver converts characters by using the default conversion character set of the Java Virtual Machine:

- Specification value for the UAP name (value specified by the UAPNAME property)
- Authorization identifier or password (value specified by the getConnection method)

- Specification value for the client environment definition specified by `EnvironmentVariables`
- Specification value for an environment variable specified by the environment variable group name of the HiRDB client

`HIRDB_CURSOR=`*cursor-operation-mode*

Specifies whether objects of the `ResultSet` class are to be validated or invalidated after HiRDB executes commit processing.

`TRUE`: Validate objects of the `ResultSet` class even after commit processing.

`FALSE`: Invalidate objects of the `ResultSet` class after commit processing.

If this specification is omitted, `FALSE` is assumed.

If a value other than `TRUE` or `FALSE` is specified, the JDBC driver throws an `SQLException`.

If an invalidated `ResultSet` object executes an operation other than calling the `close` method, the JDBC driver throws an `SQLException`.

Note

For notes about specifying `HIRDB_CURSOR`, see *(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

`STATEMENT_COMMIT_BEHAVIOR=`*Statement-object-status-after-commit-execution*

Specifies whether objects of the `Statement` and `PreparedStatement` classes (referred to collectively hereafter as `Statement`) are to be validated or invalidated after HiRDB executes commit processing.

`TRUE`: Validate `Statement` objects even after HiRDB executes commit processing.

`FALSE`: Invalidate `Statement` objects after HiRDB executes commit processing.

The entities that are invalidated after commit execution are SQL statements that were precompiled by the `prepareStatement` method of the `Connection` class, and `ResultSet class` objects obtained by the `executeQuery` method of `Statement`.

If this specification is omitted, `TRUE` is assumed.

Note

For notes about specifying `STATEMENT_COMMIT_BEHAVIOR`, see *(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR

The notes that follow apply to specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR.

When TRUE is specified in HIRDB_CURSOR or STATEMENT_COMMIT_BEHAVIOR

- If the value of the PDDDLDEAPRP client environment definition is NO and another user executes a definition SQL statement for a schema resource (table or index) to be accessed by a SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement, the definition SQL statement remains in lock-release wait status until the connection that was accessing the schema resource is disconnected.
- If the value of the PDDDLDEAPRP client environment definition is YES and another user executes a definition SQL statement for a schema resource (table or index) to be accessed by a SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement, the preprocessing result of the SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, or CALL statement becomes invalid. If an SQL statement with an invalid preprocessing result is executed, an SQLException occurs (the value obtained by the getErrorCode method is -1542).
- When TRUE is specified for HIRDB_CURSOR or STATEMENT_COMMIT_BEHAVIOR,#1 the only precompiled SQL statements#2 that are valid after execution of commit processing#3 are the SELECT, INSERT, DELETE, UPDATE, PURGE TABLE, and CALL statements.

#1

This also applies to either of the following specifications:

- TRUE is set for the following items in the properties specified by the getConnection method:
  - HIRDB_CURSOR
  - HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR
- true is specified by the following methods of the PrdbDataSource, PrdbConnectionPoolDataSource, or PrdbXADataSource class:
  - setHiRDBCursorMode
  - setStatementCommitBehavior

#2

SQL statements are precompiled by execution of the prepareStatement method of the `Connection` class.

#3

In addition to explicit commit processing by the `commit` method, the following cases also apply:

- Implicit commit processing by the `AUTO` commit mode
- Execution of a definition SQL statement
- Execution of the `PURGE TABLE` statement
- Implicit rollback processing by the `rollback` method
- Implicit rollback processing because of an SQL execution error

In the case of SQL statements other than `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `PURGE TABLE`, and `CALL`, precompiled SQL statements become invalid during commit processing.

If a `PreparedStatement` class object that stores an invalidated precompiled SQL statement is used in executing the SQL statement, an error occurs. Shown below is an example that triggers an error:

```
PreparedStatement pstmt1 = con.prepareStatement("lock table tb1");
PreparedStatement pstmt2 = con.prepareStatement("lock table tb2");
pstmt1.execute();  //Triggers an error.
con.commit();
pstmt2.execute();  //Does not trigger an error.
pstmt1.close();
pstmt2.close();
```

Because the SQL statements to be executed in this example are `LOCK` statements, after commit processing is executed, `PreparedStatement` becomes invalid and an error occurs, even if `TRUE` is specified for `STATEMENT_COMMIT_BEHAVIOR`.

- When `TRUE` is specified for `HIRDB_CURSOR`, the JDBC driver uses the holdable cursor facility of HiRDB.

Combinations of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR specification values

Table 17-3 shows whether `ResultSet` and `Statement` objects are validated or invalidated after commit execution for each combination of `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR` specification values.

Table 17-3: Status of ResultSet objects and Statement objects after commit execution

STATEMENT_COMMIT_BEHAVIOR specification value	HIRDB_CURSOR specification value	
	TRUE	FALSE
TRUE	`ResultSet` object: Valid `Statement` object: Valid	`ResultSet` object: Invalid `Statement` object: Valid
FALSE		`ResultSet` object: Invalid `Statement` object: Invalid

Table 17-4 shows the return value of the `DatabaseMetaData` method for each combination of `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR` specification values.

Table 17-4: Return values of the DatabaseMetaData method

STATEMENT_COMMIT_BEHAVIOR specification value	HIRDB_CURSOR specification value	
	TRUE	FALSE
TRUE	`supportsOpenStatementsAcrossCommit`: `true` `supportsOpenCursorsAcrossCommit`: `true`	`supportsOpenStatementsAcrossCommit`: `true` `supportsOpenCursorsAcrossCommit`: `false`
FALSE		`supportsOpenStatementsAcrossCommit`: `false` `supportsOpenCursorsAcrossCommit`: `false`

Examples of JDBC driver operation during COMMIT execution

The operation of the JDBC driver during `COMMIT` execution depends on the specification values of `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR`.

Specification examples

```
[A]
pstmt1=con.prepareStatement("select c1 from tb1");   [1]
[B]
rs1=pstmt1.executeQuery();                           [2]
[C]
rs1.next()                                           [3]
[D]
v1=rs1.getInt(1)                                     [4]
[E]
rs1.next()                                           [5]
[F]
v1=rs1.getInt(1)                                     [6]
[G]
rs1.close()                                          [7]
```

Driver operation at COMMIT execution

COMMIT timing	H=T and S=T#1	H=F and S=T#2	H=F and S=F#3
[A]	*[1]*-*[7]*: Operates normally.		
[B]	*[1]*-*[7]*: Operates normally.		*[1]*, *[2]*, and *[7]*: Operates normally. *[3]*-*[6]*: Throws an `SQLException`.
[C]	*[1]*-*[7]*: Operates normally.	*[1]*, *[2]*, and *[7]*: Operates normally. *[3]*-*[6]*: Throws an `SQLException`.	
[D]	*[1]*-*[7]*: Operates normally.	*[1]*-*[3]* and *[7]*: Operates normally. *[4]*-*[6]*: Throws an `SQLException`.	
[E]	*[1]*-*[7]*: Operates normally.	*[1]*-*[4]* and *[7]*: Operates normally. *[5]* and *[6]*: Throws an `SQLException`.	
[F]	*[1]*-*[7]*: Operates normally.	*[1]*-*[5]* and *[7]*: Operates normally. *[6]*: Throws an `SQLException`.	
[G]	*[1]*-*[7]*: Operates normally.		

#1: This represents the case when `TRUE` is specified for both `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR`.

#2: This represents the case when `FALSE` is specified for `HIRDB_CURSOR` and `TRUE` is specified for `STATEMENT_COMMIT_BEHAVIOR`.

#3: This represents the case when `FALSE` is specified for both `HIRDB_CURSOR` and `STATEMENT_COMMIT_BEHAVIOR`.

Other notes

For notes about the PDDDLDEAPRP client environment definition, see *6.6.4 Environment definition information*.

For details about the rules for the DECLARE CURSOR holdable cursor, see the manual *HiRDB Version 8 SQL Reference*.

(2) User properties

Table 17-5 shows the properties that you can specify in the getConnection method of the DriverManager class. If the null value is specified for a property, the JDBC driver assumes that specification was omitted.

Table 17-5: Properties that can be specified in the getConnection method

Item	Property	Specified information
(a)	user	Authorization identifier
(b)	password	Password
(c)	UAPNAME	UAP identifier
(d)	JDBC_IF	Whether or not a JDBC interface method trace is to be obtained
(e)	TRC_NO	Number of entries in the JDBC interface method trace
(f)	ENCODELANG	Conversion character set for the HiRDB character codes of the connection destination
(g)	HIRDB_CURSOR	Cursor operation mode
(h)	LONGVARBINARY_ACCESS	Method of accessing a JDBC SQL-type LONGVARBINARY (BLOB and BINARY types, which are HiRDB data types) database
(i)	HiRDB_for_Java_SQL_IN_NUM	Maximum number of input ? parameters in the SQL statements to be executed
(j)	HiRDB_for_Java_SQL_OUT_NUM	Maximum number of output items for the SQL statements to be executed
(k)	HiRDB_for_Java_SQLWARNING_LEVEL	Retention level for warning information that is issued during execution of SQL statements
(l)	HiRDB_for_Java_ENV_VARIABLES	HiRDB client environment variables
(m)	HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR	Statement object status after commit execution
(n)	HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE	Length of JDBC SQL-type LONGVARBINARY data to be requested at one time to the HiRDB server

Item	Property	Specified information
(o)	HiRDB_for_Java_MAXBINARYSIZE	Maximum data size during acquisition of JDBC SQL-type LONGVARBINARY data
(p)	HiRDB_for_Java_LONGVARBINARY_TRUNCERROR	Whether or not an exception is to be thrown if truncation occurs during acquisition of JDBC SQL-type LONGVARBINARY data

(a) user

Specifies the authorization identifier.

If the null value is specified, the JDBC driver assumes that no authorization identifier was specified. If the character string has a length of 0, the JDBC driver throws an SQLException.

If this specification is omitted, either the PDNAMEPORT specification value in the HiRDB client environment definitions specified by HiRDB_for_Java_ENV_VARIABLES in the Properties argument of the getConnection method, or the PDUSER specification value in the HiRDB environment variable group specified for DBID in the URL becomes effective. For details about the specification priorities, see *17.11 Connection information priorities*.

If neither value is specified, the JDBC driver throws an SQLException when the getConnection method is executed.

(b) password

Specifies the password.

If the specification value is the null or has a length of 0, the JDBC driver assumes that no password was specified.

For details about when this specification is omitted, see *17.11 Connection information priorities*.

(c) UAPNAME

Specifies UAP identification information (UAP identifier) for accessing the HiRDB server.

In the following cases, the JDBC driver assumes that no authorization identifier was specified:

- The null value is specified.
- A character string with a length of 0 or a character string of only single-byte space characters is specified.

For details about character strings that can be specified, see the description of the PDCLTAPNAME client environment definition in *6.6.4 Environment definition* *information*.

For details about when this specification is omitted, see *17.11 Connection information priorities*.

Note

The UAP specified by this property is encoded in the conversion character set specified by ENCODELANG, and the first 30 bytes of the encoded UAP identifier are transferred to the HiRDB server (if the UAP identifier consists of more than 30 bytes, it is truncated to the first 30 bytes). Therefore, the UAP identifier that the HiRDB server can obtain is up to the first 30 bytes after the identifier has been encoded.

(d) JDBC_IF

Specifies whether or not a JDBC interface method trace is to be obtained.

ON: Obtain a JDBC interface method trace.

OFF: Do not obtain a JDBC interface method trace.

If this specification is omitted, OFF is assumed.

If any other value is specified, the JDBC driver throws an SQLException.

If the setLogWriter method has not specified valid log data, the specification of this property is disabled.

For details about the JDBC interface method trace, see *17.12 JDBC interface method trace*.

(e) TRC_NO

~<unsigned integer>((10-1000))<<500>>

Specifies the number of entries in the JDBC interface method trace.

The specification of this property is enabled when both of the following conditions are satisfied:

- The setLogWriter method has set valid log data.
- ON is specified for JDBC_IF.

If the specification of this property is enabled but the specification value is invalid, the JDBC driver throws an SQLException.

For details about a JDBC interface method trace, see *17.12 JDBC interface method trace*.

(f) ENCODELANG

Specifies the conversion character set for the HiRDB character codes of the connection destination when the JDBC driver uses the String class to exchange data with

HiRDB.

Select a specifiable conversion character set from the encoding list shown under *Internationalization* in the *Java^{TM} 2 SDK, Standard Edition* documentation.

For details about the HiRDB character codes and their corresponding conversion character sets, see Table 17-2.

When `OFF` is specified, the JDBC driver operates assuming that the conversion character set that Table 17-2 shows for the HiRDB character codes was specified. If the HiRDB character codes are `sjis`, the conversion characters determined by the OS running the JDBC driver are as follows:

For UNIX: `SJIS`

For Windows: `MS932`

Note that the specification is case sensitive (except for `OFF`).

If a conversion character set that the Java Virtual Machine does not support is specified, the JDBC driver throws an `SQLException` during connection with the HiRDB server.

If this specification is omitted, the JDBC driver converts characters using the conversion character set specified by `ENCODELANG` in the URL.

(g) HIRDB_CURSOR

Specifies whether objects of the `ResultSet` class are to be validated or invalidated after HiRDB executes commit processing.

`TRUE`: Validate objects of the `ResultSet` class even after commit processing.

`FALSE`: Invalidate objects of the `ResultSet` class after commit processing.

If this specification is omitted, the value specified by `HIRDB_CURSOR` in the URL becomes valid. If a value other than `TRUE` or `FALSE` is specified, the JDBC driver throws an `SQLException`.

If an invalidated `ResultSet` object executes an operation other than calling the `close` method, the JDBC driver throws an `SQLException`.

Note

> For notes about specifying this property, see *(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(h) LONGVARBINARY_ACCESS

Specifies the method of accessing a JDBC SQL-type `LONGVARBINARY` (`BLOB` and `BINARY` types, which are HiRDB data types) database.

`REAL`: Access the database using real data.

`LOCATOR`: Access the database using the locator facility of HiRDB.

If this specification is omitted, `REAL` is assumed.

If any other value is specified, the JDBC driver throws an `SQLException`.

Note

For notes about specification of this property, see *(q) Notes about specification of LONGVARBINARY_ACCESS*.

(i) HiRDB_for_Java_SQL_IN_NUM

~<unsigned integer>((`1`-`30000`))<<`300`>>

Specifies the maximum number of input `?` parameters in the SQL statements to be executed.

This specification becomes the number of input `?` parameters that are obtained during SQL preprocessing. If the actual number of input `?` parameters is greater than the specification value of this property, the JDBC driver gets the input `?` parameter information from the HiRDB server after SQL preprocessing.

If the specification value is invalid, the JDBC driver throws an `SQLException`.

Note

If you will not be executing SQL statements that have input `?` parameters, you should specify `1`.

(j) HiRDB_for_Java_SQL_OUT_NUM

~<unsigned integer>((`1`-`30000`))<<`300`>>

Specifies the maximum number of output items for the SQL statements that are to be executed.

This specification becomes the number of output items obtained during SQL preprocessing. If the actual number of output items is greater than the specification value of this property, the JDBC driver gets output item information from the HiRDB server after SQL preprocessing.

If the specification value is invalid, the JDBC driver throws an `SQLException`.

Note

If you will not be executing SQL statements that have output items, you should specify `1`.

(k) HiRDB_for_Java_SQLWARNING_LEVEL

Specifies the retention level for warning information that is issued during execution of SQL statements. For details about the retention levels for warning information, see *17.4.10(2)(b) Issuing conditions for SQLWarning objects*.

IGNORE: Retain warning information at the IGNORE level.

SQLWARN: Retain warning information at the SQLWARN level.

ALLWARN: Retain warning information at the ALLWARN level.

If this specification is omitted, SQLWARN is assumed.

If the specification value is invalid, the JDBC driver throws an SQLException.

(l) HiRDB_for_Java_ENV_VARIABLES

Specifies environment variables of the HiRDB client, using the following format:

```
variable-name=value;variable-name=value;...;variable-name=value
```

For details about client environment definitions supported by the JDBC driver, see *17.10 Supported client environment definitions*. If a client environment definition that is not supported by the JDBC driver is specified in a variable name, the JDBC driver ignores the specification. Note that variable names are case sensitive.

For details about the priorities for connection information that can be specified in multiple ways, see *17.11 Connection information priorities*.

Specification example

```
java.util.Properties prop;
prop=new java.util.Properties();
prop.setProperty("HiRDB_for_Java_ENV_VARIABLES",
 "PDFESHOST=FES1;PDCWAITTIME=0");
```

(m) HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR

Specifies whether Statement objects are to be validated or invalidated after HiRDB executes commit processing.

TRUE: Validate Statement objects after commit processing.

FALSE: Invalidate Statement objects after commit processing.

The objects that are invalidated after commit execution are SQL statements that were precompiled by the prepareStatement method of the Connection class, and ResultSet class objects obtained by the executeQuery method.

If this specification is omitted, the value specified for STATEMENT_COMMIT_BEHAVIOR in the URL becomes effective.

Note

For notes about specification of this property, see *(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(n) HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

~<unsigned integer>((0-2097151))<<0>> (kilobytes)

Specifies the length of JDBC SQL-type `LONGVARBINARY` data to be requested at one time to the HiRDB server. If `LONGVARBINARY_ACCESS` specifies data other than `LOCATOR` data, this specification is invalid.

For example, suppose that `20` is specified for this property and the `getBytes` method of `ResultSet` attempts to get 100 kilobytes of JDBC SQL-type `LONGVARBINARY` data stored in the database. In such a case, the JDBC driver gets and returns the data by dividing the operation into five executions of 20 kilobytes each. If `0` is specified, the JDBC driver requests the data all at once.

If the specification value is invalid, the JDBC driver throws an `SQLException`.

Note

For notes about specification of this property, see *(q) Notes about specification of LONGVARBINARY_ACCESS*.

(o) HiRDB_for_Java_MAXBINARYSIZE

~<unsigned integer>((0-2147483647)) (bytes)

Specifies the maximum data size during acquisition of JDBC SQL-type `LONGVARBINARY` data.

When the JDBC driver is getting JDBC SQL-type `LONGVARBINARY` data, it allocates memory of the defined length because it cannot determine the actual data length until it actually gets the data. Therefore, to get the value of a string for which the specified size is large (for example, 2,147,483,647 bytes, which is the maximum length for HiRDB's `BINARY` and `BLOB` data types), the JDBC driver attempts to allocate 2,147,483,647 bytes of memory, because that is the defined length. Consequently, a memory shortage may occur, depending on the execution environment.

You should specify for this property the maximum length of the data that is actually stored. If the data length of the `BINARY` or `BLOB` data to be acquired is larger than the size specified by this property, the JDBC driver truncates the acquired data to the specified size. When the JDBC driver does truncate data, it receives a warning from the HiRDB server when it executes the `next` method of `ResultSet`. In response to the received warning, the JDBC driver throws an `SQLException` or generates (or ignores) an `SQLWarning`, as determined by the specification of `setLONGVARBINARY_TruncError`.

If no upper limit is set by this property, the defined length of the target acquisition data becomes the upper limit.

If the specification value is invalid, the JDBC driver throws an `SQLException`.

Note

When LOCATOR is specified for LONGVARBINARY_ACCESS, the specification value of this property becomes invalid. The JDBC driver allocates an area based on the actual data length and gets the entire data.

(p) HiRDB_for_Java_LONGVARBINARY_TRUNCERROR

Specifies whether an exception is or is not to be thrown if truncation occurs during acquisition of JDBC SQL-type LONGVARBINARY data.

TRUE: Throw an exception if truncation occurs.

FALSE: Do not throw an exception if truncation occurs.

If this specification is omitted, TRUE is assumed.

If IGNORE is specified for HiRDB_for_Java_SQLWARNING_LEVEL, the JDBC driver operates as if FALSE were specified for this property.

Any truncation that occurs during acquisition of JDBC SQL-type LONGVARBINARY data indicates that the following condition is satisfied:

Actual length of JDBC SQL-type LONGVARBINARY *data obtained during SQL execution > data length specified by* HiRDB_for_Java_MAXBINARYSIZE

(q) Notes about specification of LONGVARBINARY_ACCESS

These notes apply to specification of LONGVARBINARY_ACCESS.

When LONGVARBINARY_ACCESS is specified together with HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE

Table 17-6 describes the differences in how the JDBC driver gets BLOB and BINARY data (HiRDB data types) based on the HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE and LONGVARBINARY_ACCESS specifications.

Table 17-6: Differences in how the HiRDB driver gets BLOB and BINARY data (HiRDB data types)

Execution method	LONGVARBINARY_ACCESS specification value	
	REAL	LOCATOR
ResultSet.next	Gets all of the BLOB or BINARY data from the connected database.	Gets the locator that indicates the BLOB or BINARY data in the connected database, instead of all of the BLOB or BINARY data.

<table>
<tr><th colspan="2" rowspan="2">Execution method</th><th colspan="2">LONGVARBINARY_ACCESS specification value</th></tr>
<tr><th>REAL</th><th>LOCATOR</th></tr>
<tr><td colspan="2"><code>ResultSet.getBytes</code>
<code>ResultSet.getString</code>
<code>ResultSet.getObject</code></td><td>Uses the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>Divides the <code>BLOB</code> or <code>BINARY</code> data into <code>ACCESSSIZE</code> x 1024-byte units and gets all of the data from the connected database.</td></tr>
<tr><td colspan="2"><code>Blob.getBytes</code></td><td>Extracts and gets the data range specified by the argument from the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>Divides the <code>BLOB</code> or <code>BINARY</code> data range specified by the argument into <code>ACCESSSIZE</code> x 1024-byte units, and gets the data range from the connected database.</td></tr>
<tr><td colspan="2"><code>ResultSet.getBinaryStream</code>
<code>ResultSet.getAsciiStream</code>
<code>ResultSet.getUnicodeStream</code>
<code>Blob.getBinaryStream</code></td><td>When the <code>InputStream read</code> method obtained by the executed method is executed, the JDBC driver extracts and gets data from the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>When the <code>InputStream read</code> method obtained by the executed method is executed, the JDBC driver gets data from the connected database.</td></tr>
<tr><td colspan="2"><code>Blob.length</code></td><td>Gets the data length from the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>Gets the data length from the connected database.</td></tr>
<tr><td colspan="2"><code>Blob.position</code></td><td>Gets the position of the data matching the search pattern from the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>Gets the position of the data matching the search pattern from the connected database.</td></tr>
<tr><td rowspan="2"><code>InputStream</code> obtained by <code>ResultSet.getBinaryStream</code> or <code>Blob.getBinaryStream</code></td><td><code>InputStream.available</code></td><td>Returns a value equal to or less than the length of the actual data indicated by the locator.</td><td>Returns a value equal to or less than <code>ACCESSSIZE</code> x 1024 bytes.</td></tr>
<tr><td><code>InputStream.skip</code></td><td>Skips reading of the range up to the length of the actual data indicated by the locator.</td><td>Skips reading of the range up to the maximum <code>ACCESSSIZE</code> x 1024 bytes.</td></tr>
<tr><td colspan="2"><code>getCharcterStream</code>
<code>ResultSet.getCharacterStream</code></td><td>When the <code>Reader read</code> method obtained by <code>getCharcterStream</code> is executed, the JDBC driver extracts and gets data from the <code>BLOB</code> or <code>BINARY</code> data obtained by <code>ResultSet.next</code>.</td><td>When the <code>Reader read</code> method obtained by <code>getCharcterStream</code> is executed, the JDBC driver gets data from the connected database.</td></tr>
</table>

Legend:

`ACCESSSIZE`: Specification value of `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE`

`InputStream` and `Reader`: Classes of objects returned by getBinaryStream, getAsciiStream, or getCharacterStream of the JDBC driver

Notes about execution performance

When LOCATOR is specified for LONGVARBINARY_ACCESS, execution performance may drop compared to when REAL is specified.

When REAL is specified, the JDBC driver accesses the connected database once during ResultSet.next processing to get the locator. On the other hand, when LOCATOR is specified, in addition to the one access during ResultSet.next processing, the JDBC driver accesses the connected database once to get the data length and once to get the data during execution of a data acquisition method such as getBytes.

Notes about data operations after the transaction terminates

When LOCATOR is specified for LONGVARBINARY_ACCESS, data operations cannot be performed if the transaction terminates during the period between acquisition of the SQL execution results (ResultSet.next) and the data operation (such as Blob.getBytes or InputStream.read). Data operations cannot be executed after the transaction terminates even if the HIRDB_CURSOR specification is TRUE.

Thus, you must ensure that all data operations will execute before the transaction terminates.

17.3 Database connection using a DataSource object and JNDI

The JDBC2.0 Optional Package can now use database connections that use a `DataSource` object and JNDI.

Although use of JNDI is not required, the advantage of using it is that you only have to set up the connection information once. The standard JDK package does not include interface definitions for the `DataSource` class or JNDI, so you have to download these items from the JavaSoft Web site when you develop an AP.

To connect a database by using a `DataSource` object and JNDI:

1. Generate the `DataSource` object.
2. Set up the connection information.
3. Register the `DataSource` object into JNDI.
4. Get the `DataSource` object from JNDI.
5. Connect to the database.

If you are not using JNDI, steps 3 and 4 are not necessary.

If you are using JNDI, steps 1 to 3 need to be executed only once. Thereafter, you can connect to the database by performing only steps 4 and 5. Once you have performed step 4, you can change the connection information as necessary.

(1) Generating the DataSource object

Generate the `DataSource` class object to be provided by the JDBC driver.

The `DataSource` class name of the JDBC driver, which is necessary for generating the `DataSource` class object, is `PrdbDataSource`.

Below is an example of generating the `DataSource` class object:

```
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource ds = null ;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource() ;
```

(2) Setting up connection information

Call the method for setting up connection information for the `DataSource` object, and set up the connection information. Because there is also a method for acquiring connection information, you can use it to check the current connection information. For details about the connection information setup and acquisition methods, see *17.7 Connection information setup and acquisition interface*.

(3) Registering the DataSource object into JNDI

Register the `DataSource` object into JNDI.

JNDI can select from among several service providers, depending on the execution environment.

Shown below is an example of registering the `DataSource` object into JNDI (this example is for Windows). In the registration example, the File System service provider, which is one of the service providers, is used. For details about other service providers, see the JNDI documentation.

```
// Generate DataSource class object to be provided by JDBC driver.
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource ds;
ds = new JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource();

// Set connection information.
        :


// Get system properties.
Properties sys_prop = System.getProperties() ;

// Set properties of File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set directory to be used by File System service provider.
// (Register under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");

// Update system properties.
System.setProperties(sys_prop) ;


// Initialize JNDI.
Context ctx = new InitialContext();

// Register DataSource class object to be provided by HiRDB driver
// into JNDI. Use logical name jdbc/TestDataSource.
ctx.bind("jdbc" + "\\" + "TestDataSource", ds);
                       :
```

When you register the logical name to be registered into JNDI, the JDBC2.0 specifications recommend that you register the logical name under a subcontext called `jdbc` (`jdbc/TestDataSource` in the registration example).

(4) Getting the DataSource object from JNDI

Get the `DataSource` object from JNDI.

Shown below is a registration example for the `DataSource` object (this is an example for Windows). This registration example uses the File System service provider, which is one of the service providers. For details about other service providers, see the JNDI documentation.

```
// Get system properties.
Properties sys_prop = System.getProperties() ;

// Set properties of File System service provider.
sys_prop.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.sun.jndi.fscontext.RefFSContextFactory");


// Set directory to be used by File System service provider.
// (Register under c:\JNDI_DIR.)
sys_prop.put(Context.PROVIDER_URL, "file:c:\\" + "JNDI_DIR");


// Update system properties.
System.setProperties(sys_prop) ;

// Initialize JNDI.
Context ctx = new InitialContext();


// Get object of local name jdbc/TestDataSource from JNDI.
Object obj = ctx.lookup("jdbc" + "\\" + "TestDataSource") ;

// Cast retrieved object to DataSource class type.
DataSource ds = (DataSource)obj;
              :
```

(5) Connecting to the database

Call the `getConnection` method for the `DataSource` object.

Shown below is an example of calling the `getConnection` method.

```
DataSource ds

// Get DataSource object from JNDI.
                     :


// Issue getConnection method.
Connection con = ds.getConnection();
  or
Connection con = ds.getConnection("USERID", "PASSWORD");#
```

#

The method's arguments (authorization identifier and password) take priority over the connection information that was set for the `DataSource` object. If needed connection information has not been set for the `DataSource` object, or if the contents of the connection information are invalid, or if connection with the HiRDB server fails, the `getConnection` method throws an `SQLException`.

After getting the `DataSource` object from JNDI, set up the connection information again, as necessary. In this case, you must cast the `DataSource` object to the `DataSource` class type provided by the JDBC driver before you set up the information. An example is shown below:

```
DataSource ds
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource hirdb_ds;

// Get DataSource object from JNDI.
                     :


// Cast DataSource object to DataSource class type provided
// by JDBC driver.
dbp_ds = (JP.co.Hitachi.soft.HiRDB.JDBC.PrdbDataSource)ds;

// Set up connection information again.
          :
```

17.4 JDBC1.2 core API

17.4.1 Driver interface

(1) Overview

The `Driver` interface provides the following principal functions:

- Database checking
- Validity check on a specified URL
- Acquisition of connection properties specified with the `DriverManager.getConnection` method
- Return of the driver version

(2) Methods

Table 17-7 lists the methods of the `Driver` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-7: Driver interface methods

Method	Remarks
`acceptsURL(String url)`	--
`connect(String url,Properties info)`	For details about the specification values for the `String url` and `Properties info` arguments of this method, see *17.2.2(1) URL syntax* and *17.2.2(2) User properties*. This method uses the value returned by the `getLoginTimeout` method of the `DriverManager` class as the maximum wait time for communication during connection with the HiRDB server. If the `getLoginTimeout` method returns `0`, the value specified for the `PDCONNECTWAITTIME` client environment definition becomes the maximum wait time. The wait time can be specified in the `setLoginTimeout` method of the `DriverManager` class. If the `getLoginTimeout` method of the `DriverManager` class returns a value outside the range 0-300, this method throws an `SQLException`.
`getMajorVersion()`	--
`getMinorVersion()`	--
`getPropertyInfo(String url,Properties info)`	--
`jdbcCompliant()`	--

Legend:

--: None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `HiRDBDriver`

17.4.2 Connection interface

(1) Overview

The `Connection` interface provides the following principal functions:

- Creation of objects in the `Statement` and `PreparedStatement` classes
- Transaction settlement (`COMMIT` or `ROLLBACK`)
- Specification of the `AUTO` commit mode

(2) Methods

Table 17-8 lists the methods of the `Connection` interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-8: Connection interface methods

Method	Remarks
`clearWarnings()`	--
`close()`	During a normal connection, this method releases the connection with the database. When a connection pool is used or during an XA connection, this method does not physically disconnect the connection. If an error occurs during `close` method execution, the method does not throw an `SQLException`. If a fatal error occurs during `close` method execution when a connection pool or an XA connection is being used and use of the connection pool becomes disabled, a `connectionErrorOccurred` method of the `ConnectionEventListener` class does not occur.
`commit()`	Even if this method is called while the `AUTO` commit mode is still effective, the interface executes commit processing without throwing an exception.
`createStatement()`	--

Method	Remarks
createStatement(int resultSetType, int resultSetConcurrency)	If TYPE_SCROLL_SENSITIVE is specified as the result set type, the JDBC driver switches to TYPE_SCROLL_INSENSITIVE and sets an SQLWarning. The only concurrent processing type that the JDBC driver supports is CONCUR_READ_ONLY. If CONCUR_UPDATABLE is specified, the JDBC driver switches to CONCUR_READ_ONLY and sets an SQLWarning.
createStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)	Same as above for the resultSetType and resultSetConcurrency arguments.
getAutoCommit()	--
getCatalog()	Returns the null value.
getHoldability()	--
getMetaData()	--
getTransactionIsolation	Always returns TRANSACTION_REPEATABLE_READ.
getTypeMap()	Returns a free map.
getWarnings()	--
isClosed()	--
isReadOnly()	Always returns false.
prepareStatement(String sql)	--
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	If TYPE_SCROLL_SENSITIVE is specified as the result set type, the JDBC driver switches to TYPE_SCROLL_INSENSITIVE and sets an SQLWarning. The only concurrent processing type that the JDBC driver supports is CONCUR_READ_ONLY. If CONCUR_UPDATABLE is specified, the JDBC driver switches to CONCUR_READ_ONLY and sets an SQLWarning.
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Same as above for the resultSetType and resultSetConcurrency arguments.
rollback()	--
setAutoCommit(boolean autoCommit)	If this method is called in the middle of a transaction, that transaction is not committed.
setCatalog(String catalog)	This specification is ignored.

Method	Remarks
`setHoldability(int holdability)`	--
`setReadOnly(boolean readOnly)`	This specification is ignored.
`setTransactionIsolation(int level)`	This specification is ignored.
`checkSession(int waittime)`	This method is specific to the JDBC driver. For details, see *(a) checkSession*.

Legend:

--: None

(a) checkSession

Function

Checks the current connection status.

Format

```
public int checkSession (int waittime) throws SQLException
```

Argument

`int waittime`:

Specifies the wait time (in seconds). If `0` is specified, the JDBC driver waits until the time specified by the `PDCWAITTIME` client environment definition.

Return value

`PrdbConnection.SESSION_ALIVE`:

The method was able to confirm that a connection is currently established.

`PrdbConnection.SESSION_NOT_ALIVE`:

Because of a cause other than a timeout within the time specified in the argument, the method was unable to confirm that a connection is currently established.

`PrdbConnection.SESSION_CHECK_TIMEOUT`:

Because of a timeout within the time specified in the argument, the method was unable to confirm that a connection is currently established.

Functional detail

Checks the current connection status.

Exception that occurs

If the wait time specification value is `-1` or less, the JDBC driver throws a `java.sql.SQLException`.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbConnection`

(4) Notes

(a) Holdability specification

If holdability is specified with one of the methods shown below, the `HIRDB_CURSOR` specification value in either the URL syntax or the properties can be overwritten for each `Statement` object (`Statement` or `PreparedStatement` object) and `Connection` object:

- `resultSetHoldability` argument of the `createStatement` or `preparedStatement` method
- `holdability` argument of the `setHoldability` method
- Whether or not `UNTIL DISCONNECT` is specified in the SQL statement (`SELECT` statement) to be executed

For `ResultSet` and `DatabaseMetaData` objects (when the `setHoldability` method is used) generated by the applicable method, the holdability specifications that become effective change depending on the combinations of these specifications and the `HIRDB_CURSOR` specifications.

Table 17-9 shows the holdability specifications that become effective for `Statement` objects generated by the following methods:

- `createStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)`
- `prepareStatement(int resultSetType,int resultSetConcurrency, int resultSetHoldability)`

Table 17-9: Effective holdability specifications (1/2)

HIRDB_CURSOR or setHiRDBCursorMode specification		Specification value of resultSetHoldability argument		
		ResultSet. HOLD_CURSORS_OVER_COMMIT	ResultSet. CLOSE_CURSORS_AT_COMMIT	
			Execution of SELECT statement with UNTIL DISCONNECT specified	Execution of other SQL statement
`TRUE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F
`FALSE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F
`true` specified for `setHiRDBCursorMode`		T	T	F
`false` specified for `setHiRDBCursorMode`		T	T	F

Legend:

T: The JDBC driver operates as if `TRUE` were specified for `HIRDB_CURSOR`.

F: The JDBC driver operates as if `FALSE` were specified for `HIRDB_CURSOR`.

Table 17-10 shows the holdability specifications that become effective for `Statement` or `DatabaseMetaData` objects generated by methods other than the Table 17-9 methods.

Table 17-10: Effective holdability specifications (2/2)

HIRDB_CURSOR or setHiRDBCursorMode specification		Specification value of setHoldability method			No execution of setHoldability method	
		ResultSet. HOLD_CURSORS_OVER_COMMIT	ResultSet. CLOSE_CURSORS_AT_COMMIT			
			Execution of SELECT statement with UNTIL DISCONNECT specified	Execution of other SQL statement	Execution of SELECT statement with UNTIL DISCONNECT specified	Execution of other SQL statement
`TRUE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	T
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	F
`FALSE` specified for `HIRDB_CURSOR` in properties	`TRUE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	T
	`FALSE` specified for `HIRDB_CURSOR` in URL syntax	T	T	F	T	F
`true` specified for `setHiRDBCursorMode`		T	T	F	T	T
`false` specified for `setHiRDBCursorMode`		T	T	F	T	F

Legend:

T: The JDBC driver operates as if `TRUE` were specified for `HIRDB_CURSOR`.

F: The JDBC driver operates as if `FALSE` were specified for `HIRDB_CURSOR`.

For details about `HIRDB_CURSOR`, see *17.2.2 Connecting to HiRDB with the* *getConnection method*.

17.4.3 Statement interface

(1) Overview

The `Statement` interface provides the following principal functions:

- SQL execution
- Creation of a result set (`ResultSet` object) as a retrieval result
- Return of the number of updated rows as an updating result
- Specification of the maximum number of rows to be retrieved
- Specification of the maximum query wait time

(2) Methods

Table 17-11 lists the methods of the `Statement` interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-11: Statement interface methods

Method	Remarks
`addBatch(String sql)`	Up to 2,147,483,647 SQL statements can be registered for execution. If the maximum is exceeded, this method throws an `SQLException`.
`cancel()`	For notes about this method, see *(4)(b) Asynchronous cancellation by the cancel method*.
`clearBatch()`	--
`clearWarnings()`	--
`close()`	If an error occurs during `close` method execution when a connection pool or an XA connection is being used, the method does not throw an `SQLException`. If a fatal error occurs and the connection pool can no longer be used, the `connectionErrorOccurred` method of the `ConnectionEventListener` class does not occur.
`execute(String sql)`	--
`executeBatch()`	--
`executeQuery(String Sql)`	If the SQL statement (such as an `INSERT` statement) does not have retrieval results, this method throws an `SQLException`.
`executeUpdate(String Sql)`	If the SQL statement (`SELECT` statement) returns retrieval results, this method throws an `SQLException`.

Method	Remarks
`getConnection()`	--
`getFetchDirection()`	--
`getFetchSize()`	Returns the value set by `setFetchSize`. If no value was set by `setFetchSize`, this method returns `0`.
`getMaxFieldSize()`	Returns the value set by `setMaxFieldSize`.
`getMaxRows()`	Returns the value set by `setMaxRows`.
`getMoreResults()`	--
`getQueryTimeout()`	Returns the value set by `setQueryTimeout`. If no value was set by `setQueryTimeout`, this method returns `0`.
`getResultSet()`	--
`getResultSetConcurrency()`	--
`getResultSetHoldability()`	--
`getResultSetType()`	--
`getUpdateCount()`	The method returns `-1` if any of the following applies: • No `execute`*XXX* method was executed. • The `execute`*XXX* that was executed last was the `executeBatch` method. • The `execute`*XXX* method that was executed last was not the `executeBatch` method and it returned a result set (example: `SELECT` statement execution). • The `getMoreResults` method was executed after the `execute`*XXX* method that was executed last. • The `execute`*XXX* method that was executed last threw an `SQLException`.
`getWarnings()`	--
`setCursorName(String name)`	--
`setEscapeProcessing(Boolean enable)`	--
`setFetchDirection(int direction)`	--
`setFetchSize(int rows)`	If no value was specified with this method or if `0` was specified, the JDBC driver uses the value of the `PDBLKF` client environment variable as an indicator of the number of rows that must be extracted from the database when it retrieves data. For notes about this method, see *(4)(a) Using the block transfer facility by specifying the setFetchSize method*.

Method	Remarks
setMaxFieldSize(int max)	--
setMaxRows(int max)	--
setQueryTimeout(int seconds)	Specifies the maximum wait time (seconds) for communication with the HiRDB server during SQL execution. If this method is not called, the time specified by the PDCWAITTIME client environment variable becomes the maximum wait time. If 65,536 or greater is specified, this method ignores the specification value.

Legend:

--: None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbStatement`

(4) Notes

(a) Using the block transfer facility by specifying the setFetchSize method

If the value `1` or greater is specified for the `setFetchSize` method, the JDBC driver uses the block transfer facility and requests the HiRDB server to transfer all at once the retrieval results for the number of rows specified in the argument. For details about the block transfer facility, see *4.7 Block transfer facility*.

Although there is no maximum specification value for the `setFetchSize` method, the block transfer facility can transfer only up to 4,096 rows at a time. Therefore, when a value greater than 4,096 is specified, the number of rows actually transferred at once will not exceed 4,096.

Table 17-12 shows the priorities that determine the number of rows that the JDBC driver requests the HiRDB server to transfer in one transmission.

Table 17-12: Priorities for number of rows that the JDBC driver requests the HiRDB server to transfer in one transmission

Priority	Specification value
1	Value specified in the argument of the `setFetchSize` method of the `ResultSet` class
2	Value specified in the argument of the `setFetchSize` method of the `Statement` class

Priority	Specification value
3	Value specified in the PDBLKF client environment definition

For details about the number of rows that the JDBC driver actually receives from the HiRDB server in one communication when the driver requests the number of rows indicated in Table 17-12, see *4.7(4) Number of rows transferred in one transmission*. However, when reading this section, replace *PDBLKF* with *number of rows requested for transfer as determined by the priorities shown in Table 17-12*, and replace *FETCH statement* with *next method of the ResultSet class*.

If the retrieval result is larger than the number of transfer rows shown in Table 17-12, the JDBC driver requests transfer to the HiRDB server as many times as necessary until retrieval is completed (or until all retrieval requests from the UAP are processed).

If one of the following conditions is satisfied, the number of rows that the JDBC driver receives from the HiRDB server in one transmission is 1:

- A projection column of the result set contains HiRDB `BLOB` type data.
- A projection column of the result set contains HiRDB `BINARY` type data with a defined length greater than 32,000, and the specification of the `PDBINARYBLKF` client environment definition is `NO`.
- All of the following conditions are satisfied:
    - During connection setup, `LOCATOR` is specified for the `LONGVARBINARY_ACCESS` property or the `setLONGVARBINARY_Access` argument of the `DataSource` class.
    - One of the following is specified:

      - `UNTIL DISCONNECT` is specified in a `SELECT` statement.

      - `ResultSet.HOLD_CURSORS_OVER_COMMIT` is specified in the `resultSetHoldability` argument of the `createStatement` or `prepareStatement` method of the `Connection` class.

      - During connection setup, `TRUE` is specified for the `HIRDB_CURSOR` setup item in the properties or the URL.

      - `true` is specified for the `setHiRDBCursorMode` argument of the `DataSource` class.

(b) Asynchronous cancellation by the cancel method

You can use the cancel method to execute asynchronous cancellation of SQL statements being processed by the HiRDB server. Even if the target `Statement` object is not executing an SQL statement, asynchronous cancellation is executed if another object is executing an SQL statement for the same connected object.

When asynchronous cancellation is executed at the HiRDB server, all PreparedStatement and ResultSet objects that were created before the asynchronous cancellation become invalid, regardless of the specification for validating or invalidating Statement and ResultSet objects after commit execution.

The following methods specify whether or not objects are to remain valid after commit execution:

- HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR property in the Properties argument of the getConnection method of the DriverManager class
- STATEMENT_COMMIT_BEHAVIOR in the URL
- setStatementCommitBehavior of the DataSource system interface
- HIRDB_CURSOR property in the Properties argument of the getConnection method of the DriverManager class
- HIRDB_CURSOR in the URL
- setHiRDBCursorMode of a DataSource-type interface
- setHoldability method of the Connection interface
- resultSetHoldability argument of the createStatement or prepareStatement method of the Connection interface
- SQL statement (with UNTIL DISCONNECT specification)

Asynchronous cancellation is not executed for the HiRDB server if the target Statement object is not executing an SQL statement, and if no other object is executing an SQL statement for that same connection object.

If XADataSource was used for the connection, an asynchronous cancellation request is not valid.

17.4.4 PreparedStatement interface

(1) Overview

The PreparedStatement interface provides the following principal functions:

- Execution of SQL statements in which the ? parameter is specified
- Specification of the ? parameter
- Generation and return of a ResultSet object as a retrieval result
- Return of the number of updated rows as an updating result

Because the PreparedStatement interface is a subinterface of the Statement interface, it inherits all of the Statement interface functions.

(2) Methods

Table 17-13 lists the methods of the `PreparedStatement` interface. This interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-13: PreparedStatement interface methods

Method	Remarks
`addBatch()`	Registers up to 2,147,483,647 parameter sets for the SQL statements to be executed. If this upper limit is exceeded, this method throws an `SQLException`.
`clearParameters()`	--
`execute()`	--
`executeQuery()`	If the SQL statement (such as an `INSERT` statement) does not have retrieval results, this method throws an `SQLException`.
`executeUpdate()`	A retrieval SQL statement cannot be executed. If a retrieval SQL statement is specified, this method throws an `SQLException`.
`setAsciiStream(int parameterIndex, InputStream x,int length)`	After input from the `x` argument is completed, this method does not execute the `close` method for `x`.
`setBigDecimal(int arameterIndex,BigDecimal x)`	--
`setBinaryStream(int parameterIndex, InputStream x,int length)`	After input from the `x` argument is completed, this method does not execute the `close` method for `x`.
`setBlob(int i, Blob x)`	--
`setBoolean(int parameterIndex,boolean x)`	If the data type of the `?` parameter specified in the `parameterIndex` argument is HiRDB's `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, or `NVARHAR` type, the value that is set for the `?` parameter is `"true"` when the `x` argument is `true`, and `"false`**Δ**`"` (**Δ** is a single-byte space) when the `x` argument is `false`.
`setByte(int parameterIndex,byte x)`	--
`setBytes(int parameterIndex,byte b[])`	--
`setCharacterStream(int parameterIndex,Reader reader,int length)`	--

Method	Remarks
`setDate(int parameterIndex,Date x)`	--
`setDate(int parameterIndex,Date x,Calendar cal)`	--
`setDouble(int parameterIndex,double x)`	--
`setFloat(int parameterIndex,float x)`	--
`setInt(int parameterIndex,int x)`	--
`setLong(int parameterIndex,long x)`	--
`setNull(int parameterIndex,int sqlType)`	The JDBC driver ignores the `sqlType` argument.
`setObject(int parameterIndex,Object x)`	If the data type of the `?` parameter specified in the `parameterIndex` argument is HiRDB's `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, or `NVARHAR` type and if `x` is a `Boolean` object, the value that is set for the `?` parameter is `"true"` when the `x` argument value is `true`, and `"falseΔ"` (Δ is a single-byte space) when the `x` argument value is `false`.
`setObject(int parameterIndex,Object x,int targetSqlType)`	If the `targetSqlType` argument is `java.sql.Types.CHAR`, `java.sql.Types.VARCHAR`, or `java.sql.Types.LONGVARCHAR` and if the `x` argument is a `Boolean` object, the value that is set for the `?` parameter is `1` when the `x` argument value is `true`, and `0` when the `x` argument value is `false`. If the value `1` or `0` is set for a `?` parameter that has HiRDB's `NCHAR` or `NVARCHAR` type, this method throws an `SQLException`.
`setObject(int parameterIndex,Object x,int targetSqlType,int scale)`	
`setShort(int parameterIndex,short x)`	--
`setString(int parameterIndex,String x)`	--
`setTime(int parameterIndex,Time x)`	--
`setTime(int parameterIndex,Time x,Calendar cal)`	--

Method	Remarks
`setTimestamp(int parameterIndex, Timestamp x)`	--
`setTimestamp(int parameterIndex, Timestamp x,Calendar cal)`	--

Legend:

--: None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbPreparedStatement`

(4) Notes

Because the `PreparedStatement` interface is a subinterface of the `Statement` interface, all notes for the `Statement` interface also apply to the `PreparedStatement` interface.

This section describes additional notes that apply to the `PreparedStatement` interface.

(a) ? parameter setup

- For details about whether mapping is possible with a `set`*XXX* method, see *17.8.3 Mapping when a ? parameter is set*.
- If the column number or name specified in a `set`*XXX* method does not exist, the JDBC driver throws an `SQLException`.
- If a value specified in a `set`*XXX* method exceeds the value range that can be represented by the data type of the corresponding `?` parameter, an overflow occurs, resulting in an `SQLException`. For details about the combinations of `set`*XXX* methods for which overflow can occur and the HiRDB data types, see *17.8.5 Overflow handling*.
- The values specified by a `set`*XXX* method remain effective until one of the following operations is executed:
    - The `clearParameters` method is executed for the target `PreparedStatement` object.
    - A `set`*XXX* method is executed for the target `PreparedStatement` object, and the `?` parameters to be specified are the same.

- The `close` method is executed for the target `PreparedStatement` object.

(b) Retaining SQL preprocessing results beyond commit or rollback processing

For details about retaining SQL preprocessing results beyond commit or rollback processing, see *17.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

(c) Specification values for ? parameters of HiRDB's DECIMAL type

Described below are operations that are executed when a `set`*XXX* method is used to specify a value for a `?` parameter of HiRDB's `DECIMAL` type, and when the precision and decimal scaling position of the `?` parameter do not match those of the specification value.

When the precision of the specification value is greater than the actual precision: the HiRDB driver throws an `SQLException`.

When the precision of the specification value is smaller than the actual precision: the HiRDB driver expands the precision of the specification value.

When the decimal scaling position of the specification value is greater than the actual decimal scaling position: the HiRDB driver truncates the actual decimal scaling position.

When the decimal scaling position of the specification value is smaller than the actual decimal scaling position: the HiRDB driver expands the decimal scaling position by adding zeros.

(d) Specification values for ? parameters of HiRDB's TIMESTAMP type

When a `set`*XXX* method is used to specify a value for a `?` parameter of HiRDB's `TIMESTAMP` type, and the fraction-of-a-second precision of the value is greater than the fraction-of-a-second precision of the `?` parameter, the JDBC driver truncates the fraction-of-a-second precision to match that of the `?` parameter.

(e) Specification values for ? parameters of HiRDB's CHAR, VARCHAR, NCHAR, NVARCHAR, MCHAR, or MVARCHAR type

When a `set`*XXX* method is used to specify a value for a `?` parameter of HiRDB's `CHAR`, `VARCHAR`, `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR` type, and when the length of the value after conversion to a character string expression is greater that the defined length of the `?` parameter, the JDBC driver throws an `SQLException`.

(f) Objects that can be specified with setObject

The objects that can be specified for the `x` argument of `setObject` are objects of the following types:

- `byte[]`

- `java.lang.Byte`
- `java.lang.Double`
- `java.lang.Float`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Short`
- `java.lang.String`
- `java.math.BigDecimal`
- `java.sql.Blob`
- `java.sql.Boolean`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`

17.4.5 ResultSet interface

(1) Overview

The `ResultSet` interface provides the following principal functions:

- Movement of data within a result set in units of rows
- Return of result data
- Notification of whether the retrieval result data is the null value

(2) Methods

Table 17-14 lists the methods of the `ResultSet` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-14: ResultSet interface methods

Method	Remarks
`absolute(int row)`	--
`afterLast()`	--
`beforeFirst()`	--
`clearWarnings()`	--

Method	Remarks
`close()`	If an error caused by physical disconnection from the database occurs during `close` method execution when a connection pool or an XA connection is being used and use of the connection pool becomes disabled, a `connectionErrorOccurred` method of the `ConnectionEventListener` class does not occur.
`findColumn(String columnName)`	--
`first()`	--
`getAsciiStream(int columnIndex)`	--
`getAsciiStream(String columnName)`	--
`getBigDecimal(int columnIndex)`	--
`getBigDecimal(String columnName)`	--
`getBinaryStream(int columnIndex)`	--
`getBinaryStream(String columnName)`	--
`getBlob(int i)`	--
`getBlob(String colName)`	--

`getBoolean(int columnIndex)`
`getBoolean(String columnName)`
	If the data type of the projection column specified by the `columnIndex` index is HiRDB's `MVARCHAR`, `MCHAR`, `NVARCHAR`, `VARCHAR`, or `CHAR` type, one of the following values is returned (after leading and trailing single-byte space characters have been deleted), depending on the data obtained from the HiRDB server:
	`1` (not applicable for `NVARCHAR`): `true`
	`true` (not case sensitive): `true`
	`0` (not applicable for `NVARCHAR`): `false`
	Other value: `false`
	
	If the data type of the projection column specified by the `columnIndex` argument is HiRDB's `NCHAR` type, one of the following values is retuned (after leading and trailing single-byte space characters have been deleted), depending on the data obtained from the HiRDB server:
	First four characters are `true` (not case sensitive): `true`
	Other value: `false`

`getByte(int columnIndex)`	--

Method	Remarks
`getByte(String columnName)`	--
`getBytes(int columnIndex)`	--
`getBytes(String columnName)`	--
`getCharacterStream(int columnIndex)`	--
`getCharacterStream(String columnName)`	--
`getConcurrency()`	--
`getCursorName()`	Returns the null value.
`getDate(int columnIndex)`	--
`getDate(int columnIndex,Calendar cal)`	--
`getDate(String columnName)`	--
`getDate(String columnName,Calendar cal)`	--
`getDouble(int columnIndex)`	--
`getDouble(String columnName)`	--
`getFetchDirection()`	--
`getFetchSize()`	Returns the value that was set for `setFetchSize`. If no value was set for `setFetchSize`, this method returns `0`.
`getFloat(int columnIndex)`	--
`getFloat(String columnName)`	--
`getInt(int columnIndex)`	--
`getInt(String columnName)`	--
`getLong(int columnIndex)`	--
`getLong(String columnName)`	--
`getMetaData()`	--
`getObject(int columnIndex)`	--
`getObject(String columnName)`	--

Method	Remarks
`getRow()`	If the maximum number of retrieved rows exceeds 2,147,483,647, this method returns `2,147,483,647`.
`getShort(int columnIndex)`	--
`getShort(String columnName)`	--
`getStatement()`	--
`getString(int columnIndex)`	--
`getString(String columnName)`	--
`getTime(int columnIndex)`	--
`getTime(int columnIndex,Calendar cal)`	--
`getTime(String columnName)`	--
`getTime(String columnName,Calendar cal)`	--
`getTimestamp(int columnIndex)`	--
`getTimestamp(int columnIndex, Calendar cal)`	--
`getTimestamp(String columnName)`	--
`getTimestamp(String columnName, Calendar cal)`	--
`getType()`	--
`getWarnings()`	--
`isAfterLast()`	--
`isBeforeFirst()`	--
`isFirst()`	--
`isLast()`	--
`last()`	--
`next()`	The cursor opens the first time the `next` method is called.
`previous()`	--

Method	Remarks
`relative(int rows)`	--
`setFetchDirection(int direction)`	--
`setFetchSize(int rows)`	If no value was specified with this method, the JDBC driver uses the number of rows specified for the `Statement` object as an indicator when it retrieves data. If no number of rows value was specified in the `Statement` object or if no `ResultSet` object was generated from the `Statement` object, the JDBC driver uses the value of the `PDBLKF` client environment variable as an indicator when it retrieves data. If `0` is specified for this method, the JDBC driver uses the value of the `PDBLKF` client environment variable as an indicator when it retrieves data. For notes about this method, see *17.4.3(4)(a) Using the block transfer facility by specifying the setFetchSize method*.
`wasNull()`	Returns `false` before a value is returned by a `get`*XXX* method.

Legend:

--: None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbResultSet`

(4) Fields

Table 17-15 lists the fields supported by the `ResultSet` interface.

Table 17-15: Fields supported by the ResultSet interface

Field	Remarks
`public static final int FETCH_FORWARD`	--
`public static final int FETCH_REVERSE`	--
`public static final int FETCH_UNKNOWN`	--
`public static final int TYPE_FORWARD_ONLY`	--
`public static final int TYPE_SCROLL_INSENSITIVE`	--

Field	Remarks
`public static final int TYPE_SCROLL_SENSITIVE`	When this value is specified, the JDBC driver assumes that `TYPE_SCROLL_INSENSITIVE` was specified.
`public static final int CONCUR_READ_ONLY`	--
`public static final int CONCUR_UPDATABLE`	When this value is specified, the JDBC driver assumes that `CONCUR_READ_ONLY` was specified.
`public static final int HOLD_CURSORS_OVER_COMMIT`	--
`public static final int CLOSE_CURSORS_AT_COMMIT`	--

Legend:

--: None

(5) Notes

(a) Value acquisition using a getXXX method

- For details about whether mapping is possible with a `get`*XXX* method, see *17.8.2 Mapping during retrieval data acquisition*.
- If the column number or name specified in a `set`*XXX* method does not exist, the JDBC driver throws an `SQLException`.
- If a value specified in a `set`*XXX* method exceeds the value range that can be represented by the data type of the corresponding `?` parameter (for example, if `getShort` is used to get an `INTEGER`-type value of `40,000`), an overflow occurs and results in an `SQLException`. For details about the combinations of `set`*XXX* methods for which overflow can occur and the HiRDB data types, see *17.8.5 Overflow handling*.

(b) Mapping (conversion)

For details about whether mapping is possible with a `get`*XXX* method to be used in getting retrieval data, see *17.8.2 Mapping during retrieval data acquisition*. If a `get`*XXX* method is called for a JDBC SQL type that cannot be mapped, the JDBC driver throws an `SQLException`.

(c) Using the block transfer facility by specifying the setFetchSize method

For details, see *17.4.3(4)(a) Using the block transfer facility by specifying the setFetchSize method*.

(d) Memory size used when the result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE

When the result set type is `ResultSet.TYPE_SCROLL_INSENSITIVE` or `ResultSet.TYPE_SCROLL_SENSITIVE`, the JDBC driver allocates memory for accumulating the retrieval results when the following methods of the `ResultSet` interface are executed:

- `ResultSet.next` method
- `ResultSet.last` method
- `ResultSet.absolute` method
- `ResultSet.relative` method
- `ResultSet.afterLast` method

The JDBC driver assigns and accumulates memory objects to all values in the retrieval results. If a value has a variable length, the memory object is set to the actual size of the retrieved data.

(e) next, absolute, relative, last, and afterLast methods

When the `next` method is executed, the JDBC driver retrieves and accumulates data from the database as described in Table 17-16.

Table 17-16: Data retrieved and accumulated from the database during execution of the next method

Condition	Result set type	
	TYPE_FORWARD_ONLY	TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE
The data of the current row, which was moved by the `next` method, has not been read into the JDBC driver.	The JDBC driver gets the moved current row from the connected database.	The JDBC driver gets the moved current row from the connected database, then reads and stores the row in its memory.
The data of the current row, which was moved by the `next` method, has been read into the JDBC driver.		The JDBC driver does not retrieve data from the connected database.

When the `absolute`, `relative`, `last`, or `afterLast` method is executed, the JDBC driver retrieves and accumulates data from the database as described in Table 17-17.

Table 17-17: Data retrieved and accumulated from the database during execution of the absolute, relative, last, or afterLast method

Condition	Result set type is TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE
The first row to the specified row# of the retrieval results contain data that the JDBC driver has not read.	The JDBC driver retrieves the rows that were not read from the connected database and stores them in its memory.
The first row to the specified row# of the retrieval results do not contain data that the JDBC driver has not read.	The JDBC driver does not retrieve data from the connected database.

Note

If the data type of the result set is `TYPE_FORWARD_ONLY`, the JDBC driver throws an `SQLException`.

#

If the `last` or `afterLast` method is used, the range is from the first row to the last row.

(f) getAsciiStream, getBinaryStream, getCharacterStream, and getUnicodeStream methods

The JDBC driver does not implicitly close objects returned by the `getAsciiStream`, `getBinaryStream`, `getCharacterStream`, and `getUnicodeStream` methods. You must make provision for the method-calling side to execute the `close` method.

(g) Number of retrieved rows

Table 17-18 shows the number of retrieved rows that `ResultSet` objects can obtain from the HiRDB server. The JDBC driver discards retrieval results that exceed the applicable number of rows shown in Table 17-18.

Table 17-18: Number of retrieved rows that ResultSet objects can obtain from the HiRDB server

ResultSet object	Result set type	
	TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE	Other type
`ResultSet` object generated by `Statement` object that executed `setMaxRows` method	The number of retrieved rows is the number of rows specified by `setMaxRows` method.	

ResultSet object	Result set type	
	TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE	Other type
Other `ResultSet` object	The number of retrieved rows is the upper limit for `setMaxRows` (2,147,483,647).	No upper limit

17.4.6 DatabaseMetaData interface

(1) Overview

The `DatabaseMetaData` interface provides the following principal functions:

- Return of various information related to the connected database
- Return of listing information, such as a list of tables or columns (the information is stored in a result set)

(2) Methods

Table 17-19 lists the methods of the `DatabaseMetaData` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-19: DatabaseMetaData interface methods

Method	Remarks
`allProceduresAreCallable()`	--
`allTablesAreSelectable()`	--
`dataDefinitionCausesTransactionCommit()`	--
`dataDefinitionIgnoredInTransactions()`	--
`deletesAreDetected(int type)`	--
`doesMaxRowSizeIncludeBlobs()`	--
`getBestRowIdentifier (String catalog,String schema,String table,int scope,boolean nullable)`	The JDBC driver ignores the `catalog` argument. The JDBC driver returns a result set that has 0 retrieved rows.
`getCatalogs()`	Returns a result set that has 0 retrieved rows.
`getCatalogSeparator()`	Returns the null value.
`getCatalogTerm()`	Returns the null value.

Method	Remarks
getColumns (String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)	The JDBC driver ignores the catalog argument.
getConnection	--
getDatabaseMajorVersion()	--
getDatabaseMinorVersion()	--
getDatabaseProductName()	Returns HiRDB.
getDatabaseProductVersion()	--
getDefaultTransactionIsolation()	--
getDriverMajorVersion()	--
getDriverMinorVersion()	--
getDriverName()	--
getDriverVersion()	--
getExtraNameCharacters()	--
getIdentifierQuoteString()	--
getIndexInfo (String catalog,String schema,String table,boolean unique,boolean approximate)	The JDBC driver ignores the catalog argument.
getMaxBinaryLiteralLength()	--
getMaxCatalogNameLength()	--
getMaxCharLiteralLength()	--
getMaxColumnNameLength()	--
getMaxColumnsInGroupBy()	--
getMaxColumnsInIndex()	--
getMaxColumnsInOrderBy()	--
getMaxColumnsInSelect()	--
getMaxColumnsInTable()	--
getMaxConnections()	--
getMaxCursorNameLength()	--

Method	Remarks
`getMaxIndexLength()`	--
`getMaxProcedureNameLength()`	--
`getMaxRowSize()`	--
`getMaxSchemaNameLength()`	--
`getMaxStatementLength()`	--
`getMaxStatements()`	--
`getMaxTableNameLength()`	--
`getMaxTablesInSelect()`	--
`getMaxUserNameLength()`	--
`getNumericFunctions()`	--
`getPrimaryKeys()`	--
`getProcedureTerm()`	--
`getSchemas()`	--
`getSchemaTerm()`	Returns `schema`.
`getSearchStringEscape()`	Returns `\`.
`getSQLKeywords()`	--
`getStringFunctions()`	--
`getSystemFunctions()`	--
`getTables(String catalog,String schemaPattern,String tableNamePattern,String[] types)`	The JDBC driver ignores the `catalog` argument.
`getTimeDateFunctions()`	--
`getTypeInfo()`	--
`insertsAreDetected(int type)`	--
`isCatalogAtStart()`	--
`isReadOnly()`	--
`nullPlusNonNullIsNull()`	--
`nullsAreSortedAtEnd()`	--

Method	Remarks
nullsAreSortedAtStart()	--
nullsAreSortedHigh()	--
nullsAreSortedLow()	--
othersDeletesAreVisible()	--
othersInsertsAreVisible()	--
othersUpdatesAreVisible()	--
ownDeletesAreVisible()	--
ownInsertsAreVisible()	--
ownUpdatesAreVisible()	--
storesLowerCaseIdentifiers()	--
storesLowerCaseQuotedIdentifiers()	--
storesMixedCaseIdentifiers()	--
storesMixedCaseQuotedIdentifiers()	--
storesUpperCaseIdentifiers()	--
storesUpperCaseQuotedIdentifiers()	--
supportsAlterTableWithAddColumn()	--
supportsAlterTableWithDropColumn()	--
supportsANSI92EntryLevelSQL()	--
supportsANSI92FullSQL()	--
supportsANSI92IntermediateSQL()	--
supportsBatchUpdates()	--
supportsCatalogsInDataManipulation()	--
supportsCatalogsInIndexDefinitions()	--
supportsCatalogsInPrivilegeDefinitions()	--
supportsCatalogsInProcedureCalls()	--
supportsCatalogsInTableDefinitions()	--
supportsColumnAliasing()	--

Method	Remarks
supportsConvert()	--
supportsConvert(int fromType,int toType)	--
supportsCoreSQLGrammar()	--
supportsCorrelatedSubqueries()	--
supportsDataDefinitionAndDataManipulationTransactions()	--
supportsDataManipulationTransactionsOnly()	--
supportsDifferentTableCorrelationNames()	--
supportsExpressionsInOrderBy()	--
supportsExtendedSQLGrammar()	--
supportsFullOuterJoins()	--
supportsGroupBy()	--
supportsGroupByBeyondSelect()	--
supportsGroupByUnrelated()	--
supportsIntegrityEnhancementFacility()	--
supportsLikeEscapeClause()	--
supportsLimitedOuterJoins()	--
supportsMinimumSQLGrammar()	--
supportsMixedCaseIdentifiers()	--
supportsMixedCaseQuotedIdentifiers()	--
supportsMultipleResultSets()	--
supportsMultipleTransactions()	--
supportsNonNullableColumns()	--
supportsOpenCursorsAcrossCommit()	--
supportsOpenCursorsAcrossRollback()	--
supportsOpenStatementsAcrossCommit()	--
supportsOpenStatementsAcrossRollback()	--

Method	Remarks
`supportsOrderByUnrelated()`	--
`supportsOuterJoins()`	--
`supportsPositionedDelete()`	--
`supportsPositionedUpdate()`	--
`supportsResultSetConcurrency(int type, int concurrency)`	--
`supportsResultSetType(int type)`	--
`supportsSchemasInDataManipulation()`	--
`supportsSchemasInIndexDefinitions()`	--
`supportsSchemasInPrivilegeDefinitions()`	--
`supportsSchemasInProcedureCalls()`	--
`supportsSchemasInTableDefinitions()`	--
`supportsSelectForUpdate()`	--
`supportsStoredProcedures()`	--
`supportsSubqueriesInComparisons()`	--
`supportsSubqueriesInExists()`	--
`supportsSubqueriesInIns()`	--
`supportsSubqueriesInQuantifieds()`	--
`supportsTableCorrelationNames()`	--
`supportsTransactionIsolationLevel(int level)`	Returns `true` if the provided transaction level is `TRANSACTION_REPEATABLE_READ`.
`supportsTransactions()`	--
`supportsUnion()`	--
`supportsUnionAll()`	--
`updatesAreDetected()`	--
`usesLocalFilePerTable()`	--
`usesLocalFiles()`	--

Legend:

--: None

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbDatabaseMetaData`

17.4.7 ResultSetMetaData interface

(1) Overview

The `ResultSetMetaData` interface provides the following principal function:

- Return of meta information, such as the data type and the data length, for each column in the result set.

(2) Methods

Table 17-20 lists the methods of the `ResultSetMetaData` interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an `SQLException`.

Table 17-20: ResultSetMetaData interface methods

Method	Remarks
`getCatalogName(int column)`	--
`getColumnClassName(int column)`	--
`getColumnCount()`	--
`getColumnDisplaySize(int column)`	--
`getColumnLabel(int column)`	--
`getColumnName(int column)`	--
`getColumnType(int column)`	--
`getColumnTypeName(int column)`	--
`getPrecision(int column)`	--
`getScale(int column)`	--
`getSchemaName(int column)`	--
`getTableName(int column)`	--
`isAutoIncrement(int column)`	--

Method	Remarks
`isCaseSensitive(int column)`	--
`isCurrency(int column)`	--
`isDefinitelyWritable(int column)`	--
`isNullable(int column)`	--
`isReadOnly(int column)`	--
`isSearchable(int column)`	--
`isSigned(int column)`	--
`isWritable(int column)`	--

Legend:

--: None

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbResultSetMetaData`

(4) Notes

(a) getColumnName and getColumnLabel methods

The `getColumnName` and `getColumnLabel` methods get retrieval item names from `SQLNAME` in the Column Name Descriptor Area (`SQLCNDA`) that the HiRDB driver sends to the JDBC driver. The methods then convert the names to Java internal codes and return them. For a description of the return values of these methods for specified columns, see *C.1 Organization and contents of the Column Name Descriptor Area*.

17.4.8 Blob interface

(1) Overview

The `Blob` interface provides the following principal functions:

- Acquisition of binary data
- Acquisition of the length of binary data
- Acquisition of the pattern-matching position

The JDBC driver uses the `PrdbBlob` class to install the `Blob` interface.

The JDBC driver generates `PrdbBlob` class objects as return values of the `getBlock` method of ResultSet.

(2) Methods

Table 17-21 lists the methods of the Blob interface. The interface does not support methods that are not listed in the table. If an unsupported method is specified, the interface throws an SQLException.

Table 17-21: Blob interface methods

Method	Remarks
getBinaryStream()	--
getBytes(long pos, int length)	--
length()	--
position(Blob pattern, long start)	--
position(byte[] pattern, long start)	--

Legend:

--: None.

(3) Package and class names

The names of the package and class for installing this interface are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbBlob

17.4.9 SQLException interface

The SQLException interface uses the SQLException class of the java.sql package directly. For details and usage information about each method provided by the SQLException interface, see the JDBC documentation provided by JavaSoft.

17.4.10 SQLWarning interface

(1) Overview

The SQLWarning interface provides the following principal function:

- Provision of information related to database access warnings

If a method object triggers a warning report, an SQLWarning object is accumulated without an exception notice to that method object.

(2) Notes

(a) Releasing accumulated SQLWarning objects

`SQLWarning` objects are accumulated as a chain linked to the method object (`Connection`, `Statement`, `PreparedStatement`, or `ResultSet`) that triggers the warning reports.

To release accumulated `SQLWarning` objects explicitly, execute the `clearWarnings` method for the method object that triggered the warnings.

(b) Issuing conditions for SQLWarning objects

If the specified warning retention level indicates that warnings that occur during SQL execution are to be retained in the JDBC driver, the JDBC driver generates `SQLWarning` objects and retains warning information. In addition, a property can be used to specify warning retention for `Connection` objects.

Table 17-22 describes the conditions under which `SQLWarning` objects are generated.

Table 17-22: Conditions for generation of SQLWarning objects

SQL execution result		Warning retention level		
		IGNORE	SQLWARN	ALLWARN
`SQLCODE` is a value greater than `0` other than `100`, `110`, or `120`	Generated by an object other than a `Connection` object	No	No	Yes
	Generated by a `Connection` object	No	No	Yes#
`SQLWARN0` of the SQL Communication Area is `W` (except when `SQLWARN6` is `W`)	Generated by an object other than a `Connection` object	No	Yes	Yes
	Generated by a `Connection` object	No	Yes#	Yes#
Warning occurs in the JDBC driver	Generated by an object other than a `Connection` object	No	Yes	Yes
	Generated by a `Connection` object	No	Yes#	Yes#

Legend:

Yes: An `SQLWarning` object is generated.

No: An `SQLWarning` object is not generated.

Note

You use the HiRDB_for_Java_SQLWARNING_LEVEL property or the setSQLWarningLevel method to specify a warning retention level. The default level is SQLWARN.

#

If the specification for not retaining warnings has been set for Connection objects, an SQLWarning object is not generated.

17.4.11 Unsupported interfaces

The JDBC1.2 core API does not support the following interfaces:

- Array
- CallableStatement
- Clob
- ParameterMetaData
- Savepoint
- SQLData
- SQLInput
- SQLOutput

17.5 JDBC2.1 Core API

17.5.1 Expansion of the result set

Scrolling and parallel processing have been added to the JDBC2.1 Core API as expansion facilities for result sets (`ResultSet` class).

(1) Scrolling types

There are three types of scrolling for result sets:

- Forward-only scrolling
- Scroll-insensitive scrolling
- Scroll-sensitive scrolling

The JDBC2.1 Core API supports only forward-only scrolling and scroll-insensitive scrolling.

(2) Parallel processing types

There are two types of parallel processing for result sets:

- Read-only parallel processing
- Updatable parallel processing

The JDBC2.1 Core API supports only read-only parallel processing.

(3) Notes

(a) Notes about specifying an unsupported result set or type of parallel processing

No error results when an unsupported result set or an unsupported type of parallel processing is specified. The JDBC2.1 Core API assumes the result set that is closest to the specified type of result set or type of parallel processing, and generates an instance of the `Statement` class or that subclass. At this time, the API generates a warning (`SQLWarning` object) and associates it with an instance of the `Connection` class.

(b) Notes on using a scrolling-type result set

In the case of a scrolling-type result set, all retrieved data is cached in the JDBC driver. This means that a large data size increases the possibility of a memory shortage or a drop in performance. When you use a scrolling-type result set, you should take steps in advance to minimize the amount of retrieved data. For example, you can add appropriate conditions to the SQL statements.

17.5.2 Batch update

In the JDBC 2.1 Core API, a batch update facility has been added to the `Statement` and `PreparedStatement` classes. This facility enables you to register multiple SQL statements or parameter values and execute them all at once.

When you execute a batch update, you can use facilities that use HiRDB arrays.

Facilities that use arrays are effective when you need to update quickly a large volume of data for HiRDB. For details about facilities that use arrays, see *4.8 Facilities using arrays*.

(1) Batch update with the Statement class

The following notes apply to batch update with the `Statement` class.

- Use the `addBatch` method to register multiple update SQL statements.
- Use the `executeBatch` method to execute registered update SQL statements collectively.
- An array of the numbers of rows updated by the individual update SQL statements is returned as the batch execution results.
- If an error occurs during batch update, the batch update facility throws a `BatchUpdateException`.
- If the registered SQL statements include a retrieval SQL statement, the batch update facility throws a `BatchUpdateException` when the `executeBatch` method is called.

Because the JDBC driver cannot execute multiple SQL statements simultaneously, it executes the registered SQL statements consecutively.

(2) Batch update with the PreparedStatement class

The following notes apply to batch update with the `PreparedStatement` class.

- Use the normal procedure (`set`*XXX* method) to specify the `?` parameters for the update SQL statements specified during `PreparedStatement` instance generation.
- Use the `addBatch` method to register the `?` parameter sets.
- Use the `executeBatch` method to execute the registered `?` parameter sets collectively.
- An array of the number of rows updated by the individual `?` parameter sets is returned as the batch execution results.
- If an error occurs during batch execution, the batch update facility throws a `BatchUpdateException`.
- If an SQL statement specified during `PreparedStatement` instance generation is a retrieval SQL statement, the batch update facility throws a `BatchUpdateException` when the `executeBatch` method is called.

The JDBC driver executes processing by using facilities that use HiRDB arrays.

Notes

You must pay close attention to subsequent executions of `addBatch`, because the values that were set for the previous execution are inherited when the number of parameters specified by the `set`*XXX* method is insufficient.

The following example has two `INTEGER`-type arrays (array 1 and array 2):

Specification example

```
prepstmt.setInt(1,100);
prepstmt.setInt(2,100);
prepstmt.addBatch();
prepstmt.setInt(1,200);
prepstmt.addBatch();
prepstmt.executeBatch();
```

Explanation

- The values that are set by the first `addBatch` are array 1=`100` and array 2=`100`.

  If the number of parameters specified by `addBatch` is insufficient, an error occurs.

- The values that are set by the second `addBatch` are array 1=`200` and array 2=`100`.

  The second `addBatch` does not update the information for array 2, so the array 2 information is inherited from the first `addBatch`.

(3) Notes

(a) Implicit commit by the HiRDB server

If the SQL statements registered with `addBatch` contain one of the following SQL statements, you must use the batch update facility for SQL statements carefully, because the HiRDB server commits that SQL statement implicitly when the statement is executed:

- `PURGE TABLE` statement
- Any definition SQL statement in which `YES` is specified for the `PDCMMTBFDDL` client environment variable

(b) Processing by the batch update facility when addBatch specifications for parameters and SQL statements are combined

When `addBatch` specifications for parameters and `addBatch` specifications for SQL statements are combined, the batch update facility executes the `addBatch` specifications sequentially instead of by batch update. An example is shown below:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.addBatch();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.addBatch();
pstmt.addBatch("INSERT INTO T2 VALUES(1,2,3)");
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.addBatch();
pstmt.executeBatch();
```

When this UAP is executed, each `addBatch` unit becomes an SQL execution, because there are both `addBatch` specifications for parameters and `addBatch` specifications for SQL statements. Therefore, executing this UAP produces the same results as executing the following UAP:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE T1 SET C1=? WHERE C2=?");
pstmt.setInt(1, 1);
pstmt.setInt(2, 1);
pstmt.executeUpdate();
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.executeUpdate("INSERT INTO T2 VALUES(1,2,3)");
pstmt.setInt(1, 3);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
pstmt.setInt(1, 4);
pstmt.setInt(2, 4);
pstmt.executeUpdate();
```

When you use the batch update facility on a combination of `addBatch` for parameters and `addBatch` for SQL statements, it is recommended that you disable the auto-commit mode for the `Connection` class.

(c) Batch update with SQL statements that contain a ? parameter for HiRDB's BINARY type

When batch update is executed with SQL statements that contain a `?` parameter for HiRDB's `BINARY` type, sequential execution is executed instead of batch update when the following condition applies:

- The length of the data to be set with the `set`*XXX* method for the `?` parameter exceeds 32,000 bytes (if character data is specified with the `setString` method, the data length after the data has been encoded into data to be passed to HiRDB exceeds 32,000 bytes).

(d) Batch update for SQL statements that contain a ? parameter for HiRDB's BLOB type

When batch update is executed with SQL statements that include a `?` parameter for HiRDB's `BLOB` type, the statements are executed sequentially instead of by batch update.

(e) Registering multiple parameters with the addBatch method

The JDBC driver accumulates in the driver all parameters registered with the `addBatch` method until the `executeBatch` method is executed. You should make note of the amount of memory being used when you are registering multiple parameters.

When batch update is executed with a facility that uses HiRDB arrays, the maximum number of executions that the JDBC driver can request to the HiRDB server is 30,000. To register more than 30,000 parameters, you must divide them into groups of no more than 30,000 and request SQL execution to the HiRDB server for each group. Note also that because of the amount of memory in the JDBC driver that is used in this case, the performance enhancement expected for batch updating may not be realized. When more than 30,000 SQL executions are necessary, it is recommended that you execute the `executeBatch` method in units of 30,000 or fewer SQL executions.

17.5.3 Added data types

Several new JDBC SQL types have been added to the JDBC2.1 Core API. Although the following JDBC SQL types have been added, the JDBC driver cannot use them:

- `BLOB`
- `CLOB`
- `ARRAY`
- `REF`
- `DISTINCT`
- `STRUCT`
- `JAVA OBJECT`

17.5.4 Unsupported interfaces

The JDBC2.1 Core API does not support the following interfaces:

- `Array`
- `Clob`

- `Ref`
- `SQLData`
- `SQLInput`
- `SQLOutput`
- `Struct`

17.6 JDBC2.0 Optional Package

The following functions were added to the JDBC2.0 Optional Package:

- JNDI support
- Connection pool
- Distributed transactions
- RowSets

Note, however, that the JDBC driver cannot use RowSets.

17.6.1 JNDI support

(1) DataSource interface

For details and usage information about the methods provided by the `DataSource` interface, see the JDBC documentation. This section shows the `DataSource` interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-23 lists the methods of the `DataSource` interface.

Table 17-23: DataSource interface methods

Method	Remarks
`getConnection()`	For details about the priorities among the setting methods for authorization identifiers and passwords, see *17.11 Connection information priorities*.
`getConnection(String username,String password)`	If the `user` or `password` argument is the null value, this method indicates that no authorization identifier or password was specified by that argument. If the `password` argument is a character string whose length is 0, this method indicates that no password was specified. For details about the setting value used when a password is not specified, see *17.11 Connection information priorities*. If the `user` argument is a character string whose length is 0, this method throws an `SQLException`.
`getLoginTimeout()`	This method returns the value specified by the `setLoginTimeout` method. If no value was specified by the `setLoginTimeout` method, this method returns `0`.
`getLogWriter()`	--

Method	Remarks
`setLoginTimeout(int seconds)`	Use this method to set the physical connection time with the HiRDB server when a `Connection` object is retrieved with the `getConnection` method. If the `setLoginTimeout` method has not been executed, the time specified in `PDCONNECTWAITTIME` in the client environment definition becomes the maximum wait time for the HiRDB server. If a value outside the range 0-300 is specified, this method throws an `SQLException`.
`setLogWriter(PrintWriter out)`	--

Legend:

--: None.

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbDataSource`

17.6.2 Connection pool

(1) ConnectionPoolDataSource interface

For details and usage information about the methods provided by the `ConnectionPoolDataSource` interface, see the JDBC documentation. This section shows the `ConnectionPoolDataSource` interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-24 lists the methods of the `ConnectionPoolDataSource` interface.

Table 17-24: ConnectionPoolDataSource interface methods

Method	Remarks
`getLoginTimeout()`	This method returns the value specified by the `setLoginTimeout` method. If no value was specified by the `setLoginTimeout` method, this method returns `0`.
`getLogWriter()`	--
`getPooledConnection()`	For details about the priorities among the setting methods for authorization identifiers and passwords, see *17.11 Connection information priorities*.

Method	Remarks
getPooledConnection(String user,String password)	If the user or password argument is the null value, this method indicates that no authorization identifier or password was specified by this argument. If the password argument is a character string whose length is 0, this method indicates that no password was specified. For details about the setting value used when a password is not specified, see *17.11 Connection information priorities*. If the user argument is a character string whose length is 0, this method throws an SQLException.
setLoginTimeout(int seconds)	This specification is used only for the physical connection time with the HiRDB server. When 0 is specified or when the setLoginTimeout method is not executed, the time that was specified in PDCONNECTWAITTIME in the client environment definition becomes the maximum wait time for the HiRDB server. If a value outside the range 0-300 is specified, this method throws an SQLException exception.
setLogWriter(PrintWriter out)	--

Legend:

--: None.

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: JP.co.Hitachi.soft.HiRDB.JDBC

Class name: PrdbConnectionPoolDataSource

(2) PooledConnection interface

For details and usage information about the methods provided by the PooledConnection interface, see the JDBC documentation. This section shows the PooledConnection interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-25 lists the methods of the PooledConnection interface.

Table 17-25: PooledConnection interface methods

Method	Remarks
getConnection()	The returned Connection object has a 1-to-1 relationship with the physical connection with the HiRDB server, and a physical connection is established as necessary. Once a physical connection is established, it is not disconnected until this class object is closed. Even if the close method is executed for the Connection object, the class object retains the physical connection without closing it. The retained physical connection is reused the next time the application calls this method to request a connection. (The wait time that was specified with the setLoginTimeout method or in PDCONNECTWAITTIME in the client environment definition does not occur.)
addConnectionEventListener(ConnectionEventListener listener)	Methods of the JDBC driver are not called from the event listener registered with this method. If the event listener tries to call a method, the JDBC driver may not respond.
close()	This method closes a physical connection. Even if a Connection object is obtained and the database is being accessed, this method tries to physically close the connection when it is executed.
removeConnectionEventListener(ConnectionEventListener listener)	--

Legend:

--: None

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbPooledConnection`

17.6.3 Distributed transactions

(1) XAConnection interface

For details and usage information about the methods provided by the `XAConnection` interface, see the JDBC documentation. This section shows the `XAConnection` interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-26 lists the methods of the `XAConnection` interface.

Table 17-26: XAConnection interface methods

Method	Remarks
`getXAResource()`	--

Legend:

--: None.

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbXAConnection`

(2) XADataSource interface

For details and usage information about the methods provided by the `XADataSource` interface, see the JDBC documentation. This section shows the `XADataSource` interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-27 lists the methods of the `XADataSource` interface.

Table 17-27: XADataSource interface methods

Method	Remarks
`getLoginTimeout()`	Returns the value specified by the `setLoginTimeout` method. If no value was set by the `setLoginTimeout` method, this method returns `0`.
`getLogWriter()`	--
`getXAConnection()`	For details about the priorities among the setting methods for authorization identifiers and passwords, see *17.11 Connection information priorities*.
`getXAConnection(String username,String password)`	If the `user` or `password` argument is the null value, this method indicates that no authorization identifier or password was specified by this argument. If the `password` argument is a character string whose length is 0, this method indicates that no password was specified. For details about the setting value used when a password is not specified, see *17.11 Connection information priorities*. If the `user` argument is a character string whose length is 0, this method throws an `SQLException`.

Method	Remarks
`setLoginTimeout(int seconds)`	This specification is used only for the physical connection time with the HiRDB server. When `0` is specified or when the `setLoginTimeout` method is not executed, the time that was specified in `PDCONNECTWAITTIME` in the client environment definition becomes the maximum wait time for the HiRDB server. If a value outside the range 0-300 is specified, this method throws an `SQLException`.
`setLogWriter(PrintWriter out)`	--

Legend:

--: None

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbXADataSource`

(3) XAResource interface

For details and usage information about the methods provided by the `XAResource` interface, see the JDBC documentation. This section shows the `XAResource` interface methods that are supported by the JDBC driver.

(a) Methods

Table 17-28 lists the methods of the `XAResource` interface.

Table 17-28: XAResource interface methods

Method	Remarks
`commit(Xid xid, boolean onePhase)`	--
`end(Xid xid, int flags)`	--
`getTransactionTimeout()`	This method returns `0` unconditionally.
`prepare(Xid xid)`	--
`recover(int flag)`	--
`rollback(Xid xid)`	--

Method	Remarks
`setTransactionTimeout(int seconds)`	This method does not set the transaction timeout value. Instead, it returns `false` to indicate that the transaction timeout time was not set properly.
`start(Xid xid, int flags)`	--

Legend:

--: None

(b) Package and class names

The names of the package and class for using this interface directly are as follows:

Package name: `JP.co.Hitachi.soft.HiRDB.JDBC`

Class name: `PrdbXAResource`

(4) XAException interface

The `XAException` interface directly uses the `XAException` class of the `javax.transaction.xa` package. For details and usage instructions about the methods provided by the `XAException` interface, see the related JDBC documentation.

17.6.4 Unsupported interfaces

The JDBC2.0 Optional Package does not support the following interfaces:

- `RowSet`
- `RowSetInternal`
- `RowSetListner`
- `RowSetMetaData`
- `RowSetReader`

17.7 Connection information setup and acquisition interface

The `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` classes provided by the JDBC2.0 Optional Package provide methods for setting and getting connection information necessary for connection to the database, in addition to the methods prescribed by the JDBC2.0 Optional Package specifications.

Table 17-29 lists the methods for setting and getting connection information.

Table 17-29: Methods for setting and getting connection information

Method	Function
setDescription	Sets the additional connection information needed for connection to the database.
getDescription	Gets the additional connection information needed for connection to the database.
setDBHostName	Sets the host name of the HiRDB to be connected.
getDBHostName	Gets the host name of the HiRDB to be connected.
setJDBC_IF_TRC	Sets whether or not a JDBC interface method trace is to be acquired.
getJDBC_IF_TRC	Gets the setting information about whether or not a JDBC interface method trace is to be acquired.
setTRC_NO	Sets the number of entries in the JDBC interface method trace.
getTRC_NO	Gets the number of entries in the JDBC interface method trace.
setUapName	Sets a UAP name.
getUapName	Gets the UAP name.
setUser	Sets an authorization identifier for database connection.
getUser	Gets the authorization identifier for database connection.
setPassword	Sets a password for database connection.
getPassword	Gets the password for database connection.
setXAOpenString	Sets an XA open character string.
getXAOpenString	Gets the XA open character string.
setXACloseString	Sets an XA close character string.
getXACloseString	Gets the XA close character string.

Method	Function
setLONGVARBINARY_Access	Sets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).
getLONGVARBINARY_Access	Gets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).
setSQLInNum	Sets the maximum number of input ? parameters in the SQL statements to be executed.
getSQLInNum	Gets the maximum number of input ? parameters in the SQL statements to be executed.
setSQLOutNum	Sets the maximum number of retrieval items for the SQL statements to be executed.
getSQLOutNum	Gets the maximum number of retrieval items for the SQL statements to be executed.
setSQLWarningLevel	Sets the warning retention level for warnings that occur during SQL execution.
getSQLWarningLevel	Gets the warning retention level for warnings that occur during SQL execution.
setXALocalCommitMode	Sets whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.
getXALocalCommitMode	Gets the setting information about whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.
setSQLWarningIgnore	Sets whether or not warnings returned from the database are to be discarded by the Connection class.
getSQLWarningIgnore	Gets the setting information about whether or not warnings returned from the database are to be discarded by the Connection class.
setHiRDBCursorMode	Sets whether or not objects of the ResultSet class are to be validated when HiRDB executes commit processing.
getHiRDBCursorMode	Gets the setting information about whether or not objects of the ResultSet class are to be validated when HiRDB executes commit processing.
setNotErrorOccurred	Sets whether or not the calling of ConnectionEventListener.connectionErrorOccurred is to be suppressed.
getNotErrorOccurred	Gets the setting information about whether or not the calling of ConnectionEventListener.connectionErrorOccurred has been suppressed.

Method	Function
setEnvironmentVariables	Sets client environment definitions for HiRDB.
getEnvironmentVariables	Gets the client environment definitions for HiRDB that were set.
setEncodeLang	Sets the name of the conversion character set for data conversion.
getEncodeLang	Gets the name of the conversion character set for data conversion that was set.
setMaxBinarySize	Sets the maximum data size for retrieval of data of the LONGVARBINARY type (a JDBC SQL type).
getMaxBinarySize	Gets the maximum data size for retrieval of data of the LONGVARBINARY type (a JDBC SQL type).
setStatementCommitBehavior	Sets whether or not statement objects are to remain valid after a transaction is committed.
getStatementCommitBehavior	Gets the setting information about whether or not statement objects are to remain valid after a transaction is committed.
setLONGVARBINARY_AccessSize	Sets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.
getLONGVARBINARY_AccessSize	Gets the LONGVARBINARY (a JDBC SQL type) data length for one access request to the HiRDB server.
setLONGVARBINARY_TruncError	Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the LONGVARBINARY type (a JDBC SQL type).
getLONGVARBINARY_TruncError	Gets the setting information about whether or not an exception is to be thrown if truncation occurs during acquisition of data of the LONGVARBINARY type (a JDBC SQL type).

17.7.1 setDescription

(a) Function

Sets the additional connection information needed for connection to the database.

(b) Format

```
public void setDescription ( String description ) throws
SQLException
```

(c) Arguments

String description

Specifies additional connection information. If the null value is specified, the current additional connection information that had been set by this method is invalidated and the settings are returned to their initial status.

(d) Return value

None.

(e) Functional detail

The following table shows the additional connection information that can be set with this method.

Setting	Setting details	Setting required?
HiRDB port number	Sets the HiRDB port number, as a character string. For details about the priorities among the setting methods for the HiRDB port number, see *17.11 Connection information priorities*.	Optional
Environment variable group name of HiRDB client	Sets the environment variable group name of the HiRDB client. The name is expressed as an absolute path name that follows `@HIRDBENVGRP=`. Note the following points: • If no value is set following the equal sign, as in `@HIRDBENVGRP=,`, the JDBC driver assumes that there is no specification for this item. • The environment variable group name is case sensitive. Also, the environment variable group name depends on the OS. • If the environment variable group name contains a single-byte space or a single-byte at mark (`@`), you must enclose the name in single-byte double quotation marks (`"`). When an environment variable group name is enclosed in single-byte double quotation marks, any characters following the concluding single-byte double quotation mark through the end of the character string are ignored. An environment variable group name containing a single-byte double quotation mark or a single-byte comma cannot be specified. • For Windows, an environment variable group name that was specified with the HiRDB client environment variable registration tool cannot be specified.	Optional
HiRDB environment variable group identifier	Sets the HiRDB environment variable group identifier, as four alphanumeric characters.	Required during XA connection

Note 1:

Specification examples are shown below. In these examples `ds` represents the name of a variable that has reference to the `PrdbDataSource` class's instance. **Δ** represents a single-byte space character.

Example 1: When specifying the HiRDB port number

```
ds.setDescription ("22200");
```

Example 2: When the path of the environment variable group name is `C:\HiRDB_P\Client\HiRDB.ini`

```
ds.setDescription
("@HIRDBENVGRP=C:\\HiRDB_P\\Client\\HiRDB.ini");
```

Example 3: When the path of the environment variable group name is `C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini`

```
ds.setDescription
("@HIRDBENVGRP=\"C:\\Program∆Files\\HITACHI\\HiRDB\HiR
DB.ini\"");
```

Example 4: When the path of the environment variable group name is `/HiRDB_P/Client/HiRDB.ini`

```
ds.setDescription ("@HIRDBENVGRP=/HiRDB_P/Client/
HiRDB.ini");
```

Example 5: When a HiRDB environment variable group identifier is specified during an XA connection

```
ds.setDescription ("HDB1");ds.setXAOpenString
("HDB1+C:\\Program∆Files\\HITACHI\\HiRDB\\HiRDB.ini");
```

Note 2:

Do not include single-byte spaces in an environment variable group name. Examples of specification errors are shown below:

```
@∆HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP∆=/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=∆/HiRDB_P/Client/HiRDB.ini
@HIRDBENVGRP=/HiRDB_P/Client/HiRDB.ini∆
```

Note: ∆ represents a single-byte space character.

(f) Exceptions

When an environment variable group name begins with a single-byte at mark (`@`) and the information specified following the at mark includes a single-byte space, this method throws an `SQLException`.

17.7.2 getDescription

(a) Function

Gets the additional connection information needed for connection to the database.

(b) Format

```
public String getDescription() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the additional connection information. If no information has been set, the null value is returned.

(e) Functional detail

Returns the additional connection information needed for connection to the database, as was set by the `setDescription` method.

(f) Exceptions

None.

17.7.3 setDBHostName

(a) Function

Sets the name of the HiRDB host to be connected.

(b) Format

```
public void setDBHostName ( String db_host_name ) throws
SQLException
```

(c) Arguments

`String db_host_name`

Sets a HiRDB host name.

If the null value is specified, the current host name that had been set with this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the host name of the HiRDB to be connected.

For details about the priorities among the settings methods for the HiRDB host name, see *17.11 Connection information priorities*.

(f) Exceptions

None.

17.7.4 getDBHostName

(a) Function

Gets the name of the HiRDB host to be connected.

(b) Format

```
public String getDBHostName() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the HiRDB host name. If no value has been set, the null value is returned.

(e) Functional detail

Returns the host name of the HiRDB to be connected, as was set with the `setDBHostName` method.

(f) Exceptions

None.

17.7.5 setJDBC_IF_TRC

(a) Function

Sets whether or not a JDBC interface method trace is to be acquired.

(b) Format

```
public void setJDBC_IF_TRC ( boolean flag ) throws SQLException
```

(c) Arguments

`boolean flag`

Specifies whether or not a trace is to be acquired:

`true`: Acquire a trace.

`false`: Do not acquire a trace.

(d) Return value

None.

(e) Functional detail

Sets whether or not a JDBC interface method trace is to be acquired.

The default value when this method is not called is `false` (trace is not acquired). You can use the `setLogWriter` method in a separate operation to set the effective output destination. For details about the JDBC interface method trace, see *17.12 JDBC interface method trace*.

(f) Exceptions

None.

(g) Note

Whether or not a JDBC interface method trace is to be acquired cannot be set separately for each instance. The setting that is set by this method affects all `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` instances in existence, both when the setting is set and after the setting has been set.

17.7.6 getJDBC_IF_TRC

(a) Function

Gets setting information about whether or not a JDBC interface method trace is to be acquired.

(b) Format

```
public boolean getJDBC_IF_TRC() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not a trace is to be acquired:

`true`: A trace is acquired.

`false`: A trace is not acquired.

(e) Functional detail

Returns the setting information about whether or not a trace is to be acquired, as was set by the `setJDBC_IF_TRC` method.

For details about the JDBC interface method trace, see *17.12 JDBC interface method trace*.

(f) Exceptions

None.

17.7.7 setTRC_NO

(a) Function

Sets the number of entries in the JDBC interface method trace.

(b) Format

```
public void setTRC_NO ( int trc_no ) throws SQLException
```

(c) Arguments

`int trc_no`

Specifies the number of entries in the JDBC interface method trace.

(d) Return value

None.

(e) Functional detail

Sets the number of entries in the JDBC interface method trace, as a value in the range from 10 to 1,000.

When this method is not called, the default number of entries in the JDBC interface method trace is `500`.

For details about the JDBC interface method trace, see *17.12 JDBC interface method trace*.

(f) Exceptions

If a value outside the range from 10 to 1,000 is set, this method throws an `SQLException`.

17.7.8 getTRC_NO

(a) Function

Gets the number of entries in the JDBC interface method trace.

(b) Format

```
public int getTRC_NO() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

This is the number of entries in the JDBC interface method trace. If no value has been set, the default value `500` is returned.

(e) Functional detail

Returns the number of entries in the JDBC interface method trace, as was set by the `setTRC_NO` method.

For details about the JDBC interface method trace, see *17.12 JDBC interface method trace*.

(f) Exceptions

None.

17.7.9 setUapName

(a) Function

Sets a UAP name.

(b) Format

```
public void setUapName ( String uap_name ) throws SQLException
```

(c) Argument

`String uap_name`

Specifies a UAP name.

If the null value is specified, the current UAP name that had been set with this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets a UAP name.

The specified UAP name is used for the following purposes:

- In the output information to each type of trace information
- In the UAP identification information that is output when the `-d prc` option is specified in the `pdls` command

In the following cases, the JDBC driver assumes that no UAP name has been set by this method (for details about how the JDBC driver handles the situation when there is no setting, see *17.11 Connection information priorities*):

- When the null value is specified in the `uap_name` argument
- When a character string whose length is 0 or a character string consisting of only single-byte spaces is specified in the `uap_name` argument

(f) Exceptions

None.

(g) Notes

The UAP specified by this method is encoded using the conversion character set specified by the `setEncodeLang` method, and the first 30 bytes of the encoded UAP name are transferred to the HiRDB server (the name is truncated after 30 bytes even if the 30th byte is only part of a character). The UAP name that can be obtained by the HiRDB server is only the first 30 bytes after encoding.

17.7.10 getUapName

(a) Function

Gets the UAP name.

(b) Format

```
public String getUapName() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the UAP name.

(e) Functional detail

Returns the UAP name that was set with the `setUapName` method. If a UAP name has not been set, `HiRDB_Type4_JDBC_Driver` is returned.

(f) Exceptions

None.

17.7.11 setUser

(a) Function

Sets an authorization identifier for database connection.

(b) Format

```
public void setUser ( String user ) throws SQLException
```

(c) Arguments

`String user`

Specifies an authorization identifier.

If the null value is specified, the current authorization identifier that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an authorization identifier.

When one of the following methods is executed, the authorization identifier and password that were specified with the `setUser` and `setPassword` methods are used in establishing a physical connection to the database:

- `getConnection` method (no arguments) of the `DataSource` interface
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface
- `getXAConnection` method of the `XADataSource` interface

If the `user` argument is the null value, the JDBC driver assumes that no authorization identifier has been set by this method.

For details about how the JDBC driver handles the situation when there is no setting, see *17.11 Connection information priorities*.

(f) Exceptions

If the length of the character string specified by the `user` argument is `0`, this method throws an `SQLException`.

17.7.12 getUser

(a) Function

Gets the authorization identifier for database connection.

(b) Format

```
String void getUser() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the authorization identifier.

(e) Functional detail

Returns the authorization identifier that was set by the `setUser` method. If an authorization identifier has not been set, the null value is returned.

(f) Exceptions

None.

17.7.13 setPassword

(a) Function

Sets a password for database connection.

(b) Format

```
public void setPassword ( String password ) throws SQLException
```

(c) Arguments

`String password`

Specifies a password.

If the null value is specified, the current password that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets a password.

When one of the following methods is executed, the authorization identifier and password that were specified with the `setUser` and `setPassword` methods are used in establishing a physical connection to the database:

- `getConnection` method (no arguments) of the `DataSource` interface
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface
- `getXAConnection` method of the `XADataSource` interface

If the `password` argument is the null value or a character string whose length is 0, the JDBC driver assumes that no password has been set by this method.

For details about how the JDBC driver handles the situation when there is no setting, see *17.11 Connection information priorities*.

(f) Exceptions

None.

17.7.14 getPassword

(a) Function

Gets the password for database connection.

(b) Format

```
public String getPassword() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the password.

(e) Functional detail

Returns the password that was set by the `setPassword` method.

(f) Exceptions

None.

17.7.15 setXAOpenString

(a) Function

Sets an XA open character string.

(b) Format

```
public void setXAOpenString ( String xa_string ) throws
SQLException
```

(c) Arguments

`String xa_string`

Specifies an XA open character string.

If the null value is specified, the current XA open character string that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an XA open character string. This method is provided by the `XADataSource` interface only. Specify the XA open character string in the following format:

Format

HiRDB-environment-variable-group-identifier +
environment-variable-group-name-of-HiRDB-client

Specify the HiRDB environment variable group identifier that was set by the `setDescription` method. Unlike when the environment variable group name of the HiRDB client is specified by the `setDescription` method, in this case the environment variable group name of the HiRDB client doe not need to be enclosed in quotation marks even if the name includes a single-byte at mark (`@`) or a single-byte space.

Setting example 1

When the path of the environment variable group name of the HiRDB client is `/HiRDB/HiRDB.ini`

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+/HiRDB/HiRDB.ini");
```

Setting example 2

When the path of the environment variable group name of the HiRDB client is `C:\Program∆Files\HITACHI\HiRDB\HiRDB.ini` (∆ is a single-byte space)

```
ds.setDescription("HDB1");
ds.setXAOpenString("HDB1+C:\\Program∆Files\\HITACHI\\HiRDB
\\HiRDB.ini");
```

(f) Exceptions

None.

17.7.16 getXAOpenString

(a) Function

Gets the XA open character string.

(b) Format

```
public String getXAOpenString() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the XA open character string. If no value has been set, the null value is returned.

(e) Functional detail

Returns the XA open character string that was set by the `setXAOpenString` method. This method is provided by the `XADataSource` interface only.

(f) Exceptions

None.

17.7.17 setXACloseString

(a) Function

Sets an XA close character string.

(b) Format

```
public void setXACloseString ( String xa_string ) throws
SQLException
```

(c) Arguments

`String xa_string`

Specifies an XA close character string.

If the null value is specified, the current XA close character string that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets an XA close character string. This method is provided by the `XADataSource` interface only.

(f) Exceptions

None.

17.7.18 getXACloseString

(a) Function

Gets the XA close character string.

(b) Format

```
public String getXACloseString() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the XA close character string. If no value has been set, the null value is returned.

(e) Functional detail

Returns the XA close character string that was set by the `setXACloseString` method. This method is provided by the `XADataSource` interface only.

(f) Exceptions

None.

17.7.19 setLONGVARBINARY_Access

(a) Function

Sets the method of accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types).

(b) Format

```
public void setLONGVARBINARY_Access ( String mode ) throws
SQLException
```

(c) Arguments

`String mode`

Specifies the method of accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types).

For this method, the value specified in the argument is not case sensitive.

`REAL`

Access the data with real data.

`LOCATOR`

Access the data using HiRDB's locator facility.

If the null value is specified, the current data access method that had been set is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the method for accessing data of the `LONGVARBINARY` type (a JDBC SQL type corresponding to HiRDB's `BLOB` and `BINARY` data types). The default value when this method is not called is `REAL`.

Setting a value with this method is equivalent to setting the `LONGVARBINARY_ACCESS` property, which is shown in *17.2.2(2) User properties*.

(f) Exceptions

If a value other than REAL or LOCATOR is specified in the mode argument, this method throws a java.sql.SQLException.

(g) Notes

See *17.2.2(2)(q) Notes about specification of LONGVARBINARY_ACCESS*.

17.7.20 getLONGVARBINARY_Access

(a) Function

Gets the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types).

(b) Format

```
public String getLONGVARBINARY_Access()
```

(c) Arguments

None.

(d) Return value

String

This is the setting information for the method of accessing data of the LONGVARBINARY type (a JDBC SQL type corresponding to HiRDB's BLOB and BINARY data types):

REAL

The data is accessed using real data.

LOCATOR

The data is accessed using HiRDB's locator facility.

(e) Functional detail

Returns the information that was set by the setLONGVARBINARY_Access method.

(f) Exceptions

None.

17.7.21 setSQLInNum

(a) Function

Sets the maximum number of input ? parameters in the SQL statements to be executed.

(b) Format

```
public void setSQLInNum ( int inNum ) throws SQLException
```

(c) Arguments

`int inNum`

Specifies the maximum number of input `?` parameters in the SQL statements to be executed. The specification value range is from 1 to 30,000.

(d) Return value

None.

(e) Functional detail

Sets the number of input `?` parameter information items to be retrieved during SQL preprocessing.

If the actual number of `?` parameters is greater than the specification value of this method, the input `?` parameter information is retrieved after SQL preprocessing. The default value when this method is not called is `300`.

At the time of database connection, the value specified by this method becomes the value of the `HiRDB_for_Java_SQL_IN_NUM` property, which is shown in *17.2.2(2) User properties*.

(f) Exceptions

If a value outside the range from 1 to 30,000 is specified in the argument, this method throws an `SQLException`.

(g) Notes

If the application does not execute SQL statements that have input `?` parameters, you should specify `1` as the argument value.

17.7.22 getSQLInNum

(a) Function

Gets the maximum number of input `?` parameters in the SQL statements to be executed.

(b) Format

```
public int getSQLInNum() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

This is the maximum number of input `?` parameters in SQL statements to be executed, as set by the `setSQLInNum` method. If a value has not been set, the default value `300` is returned.

(e) Functional detail

Gets the maximum number of input `?` parameters in the SQL statements to be executed, as set by the `setSQLInNum` method.

(f) Exceptions

None.

17.7.23 setSQLOutNum

(a) Function

Sets the maximum number of retrieval items for the SQL statements to be executed.

(b) Format

```
public void setSQLOutNum ( int outNum ) throws SQLException
```

(c) Arguments

`int outNum`

Specifies the maximum number of retrieval items for the SQL statements to be executed. The specification value range is from 1 to 30,000.

(d) Return value

None.

(e) Functional detail

Sets the maximum number of retrieval items for the SQL statements to be executed.

This specification becomes the number of output items to be acquired during SQL preprocessing. The default value when this method is not called is `300`.

If the actual number of output items is greater than the specification value of this method, the output item information is acquired after SQL preprocessing.

The value specified by this method becomes the value of the `HiRDB_for_Java_SQL_OUT_NUM` property, which is shown in *17.2.2(2) User* *properties*.

(f) Exceptions

If the specified value is outside the range from 1 to 30,000, this method throws an `SQLException`.

(g) Notes

When there are no retrieval items, you should specify `1` as the argument value.

17.7.24 getSQLOutNum

(a) Function

Gets the maximum number of retrieval items for the SQL statements to be executed.

(b) Format

```
public int getSQLOutNum() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

This is the maximum number of retrieval items for the SQL statements to be executed, as set by the `setSQLOutNum` method. If a value has not been set, the default value `300` is returned.

(e) Functional detail

Gets the maximum number of retrieval items for the SQL statements to be executed, as set by the `setSQLOutNum` method.

(f) Exceptions

None.

17.7.25 setSQLWarningLevel

(a) Function

Sets the warning retention level for warnings that occur during SQL execution.

(b) Format

```
public void setSQLWarningLevel ( String warningLevel ) throws
SQLException
```

(c) Arguments

`String warningLevel`

Specifies the retention level for the warning information that occurs during SQL execution.

The following values can be specified (for details about the relationships between the specification values and the warnings to be retained, see *17.4.10(2)(b) Issuing conditions for SQLWarning objects*):

- `IGNORE`
- `SQLWARN`
- `ALLWARN`

For this method, the value specified in the argument is not case sensitive. If the null value is specified, the current warning retention level that had been set by this method is invalidated, and the setting is returned to its initial status.

(d) Return value

None.

(e) Functional detail

Sets the retention level for the warning information that occurs during SQL execution. The default value when this method is not called is `SQLWARN`.

The value specified by this method becomes the value of the `HiRDB_for_Java_SQLWARNING_LEVEL` property, which is shown in *17.2.2(2) User properties*.

(f) Exceptions

If the argument is a value other than the specification values shown above, this method throws an `SQLException`.

17.7.26 getSQLWarningLevel

(a) Function

Gets the warning retention level that was set by the `setSQLWarningLevel` method.

(b) Format

```
public String getSQLWarningLevel() throws SQLException
```

(c) Arguments

None.

(d) Return value

`String`

This is the warning retention level that was set by the `setSQLWarningLevel` method. For details about the return value and the warnings that are retained, see *17.4.10(2)(b) Issuing conditions for SQLWarning objects*.

(e) Functional detail

Returns the information that was set by the `setSQLWarningLevel` method. If no information has been set, the default value `SQLWARN` is returned.

(f) Exceptions

None.

17.7.27 setXALocalCommitMode

(a) Function

Sets whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.

(b) Format

```
public void setXALocalCommitMode ( boolean autoCommitMode )
throws SQLException
```

(c) Arguments

`boolean autoCommitMode`

Specifies the auto-commit facility:

`true`: Enable the auto-commit facility.

`false`: Disable the auto-commit facility.

(d) Return value

None.

(e) Functional detail

Sets the auto-commit facility during an XA connection. The default value is `false` (the auto-commit facility is disabled). The table below shows the relationships between this method's specification values and the JDBC driver operations.

Specification value	Condition	JDBC driver operation
`true`	Auto-commit default during `Connection` object generation	Enables auto-commit.
	Transaction termination by the `con.commit` or `con.rollback` method	Accepts normally.
	`setAutoCommit(true)` execution	Enables auto-commit.
	`setAutoCommit(false)` execution	Disables auto-commit.
`false` (default)	Auto-commit default during `Connection` object generation	Disables auto-commit.
	Transaction termination by the `con.commit` or `con.rollback` method	`SQLException`
	`setAutoCommit(true)` execution	`SQLException`
	`setAutoCommit(false)` execution	Normal termination (the driver does nothing because auto-commit cannot be enabled)

(f) Exceptions

None.

17.7.28 getXALocalCommitMode

(a) Function

Gets the setting information about whether or not the auto-commit facility is to be enabled if a transaction during an XA connection is not a distributed transaction.

(b) Format

```
public boolean getXALocalCommitMode() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting for the auto-commit facility:

`true`: The auto-commit facility is enabled.

`false`: The auto-commit facility is disabled.

(e) Functional detail

Gets the setting for the auto-commit facility.

(f) Exceptions

None.

17.7.29 setSQLWarningIgnore

(a) Function

Sets whether or not warnings returned from the database are to be discarded by the `Connection` class.

(b) Format

```
public void setSQLWarningIgnore ( boolean mode )
```

(c) Arguments

`boolean mode`

Specifies whether or not warnings are to be discarded:

`true`: Discard warnings.

`false`: Retain warnings.

(d) Return value

None.

(e) Functional detail

Sets whether or not warnings returned from the database are to be discarded by the `Connection` class. The default value is `false` (warnings are retained).

(f) Exceptions

None.

17.7.30 getSQLWarningIgnore

(a) Function

Gets the setting information about whether or not warnings returned from the database are to be discarded by the `Connection` class.

(b) Format

```
public boolean getSQLWarningIgnore()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not warnings are to be discarded:

`true`: Discards warnings.

`false`: Retains warnings.

(e) Functional detail

Gets the setting information about whether or not warnings returned from the database are to be discarded by the `Connection` class.

This method returns the information that was set by the `setSQLWarningIgnore` method. If the `setSQLWarningIgnore` method has not been executed, the default value `false` is returned.

(f) Exceptions

None.

17.7.31 setHiRDBCursorMode

(a) Function

Set whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(b) Format

```
public void setHiRDBCursorMode ( boolean mode )
```

(c) Arguments

`boolean mode`

Specifies one of the following values:

`true`: Validate objects of the `ResultSet` class after commit processing. When true is specified, objects of the following classes also become valid after commit processing:

- `Statement` class
- `PreparedStatement` class

`false`: Invalidate objects of the `ResultSet` class after commit processing.

(d) Return value

None.

(e) Functional detail

Sets whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing. The default value if this method cannot be called is `false`.

If an invalidated `ResultSet` object executes an operation other than `close` method calling, this method throws an `SQLException`.

Executing this method is the same as setting the `HIRDB_CURSOR` item, which is shown in *17.2.2(1) URL syntax*.

(f) Exceptions

None.

(g) Notes

See *17.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

17.7.32 getHiRDBCursorMode

(a) Function

Gets the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(b) Format

```
public boolean getHiRDBCursorMode()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing:

`true`: Objects of the `ResultSet` class are valid after commit processing.

`false`: Objects of the `ResultSet` class become invalid after commit processing.

(e) Functional detail

Gets the setting information about whether or not objects of the `ResultSet` class are to be validated when HiRDB executes commit processing.

(f) Exceptions

None.

(g) Notes

None.

17.7.33 setNotErrorOccurred

(a) Function

Sets whether or not the calling of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.

(b) Format

```
public void setNotErrorOccurred ( boolean mode )
```

(c) Arguments

`boolean mode`

Specifies whether or not occurrences of `connectionErrorOccurred` are to be suppressed:

`true`: Suppress the calling of `connectionErrorOccurred`.

`false`: Do not suppress the calling of `connectionErrorOccurred` (default).

(d) Return value

None.

(e) Functional detail

Specifies the setting for suppressing the calling of `ConnectionEventListener.connectionErrorOccurred`, which is called when an error occurs while `ConnectionPoolDataSource` or `XADataSource` is being used.

If this method is not set, `connectionErrorOccurred` is called. Normally, do not set this method or set `false`.

(f) Exceptions

None.

17.7.34 getNotErrorOccurred

(a) Function

Gets the setting information about whether or not the calling of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.

(b) Format

```
public boolean getNotErrorOccurred()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether or not `ConnectionEventListener.connectionErrorOccurred` is called:

`true`: `connectionErrorOccurred` is not called.

`false`: `connectionErrorOccurred` is called (default).

(e) Functional detail

Gets the setting information about whether or not `ConnectionEventListener.connectionErrorOccurred` is to be called when a fatal connection error occurs while `ConnectionPoolDataSource` or `XADataSource` is being used. If no setting information has been set, this method returns `false`.

(f) Exceptions

None.

17.7.35 setEnvironmentVariables

(a) Function

Sets client environment definitions for HiRDB.

(b) Format

```
public void setEnvironmentVariables ( String variables ) throws
SQLException
```

(c) Arguments

`String variables`

Specifies HiRDB client environment definitions in the format shown below:

Format

"*variable-name*=*value*;*variable-name*=*value*;...;*variable-name*=*value*"

A specification example is shown below:

Specification example

```
setEnvironmentVariables
("PDFESHOST=FES1;PDCWAITTIME=0");
```

If the null value is specified, the current client environment definitions that had been set by this method are invalidated, and the settings are returned to their initial status.

(d) Return value

None.

(e) Functional detail

Sets HiRDB client environment definitions.

For details about the client environment definitions that can be specified by the JDBC driver, see *17.10 Supported client environment definitions*. If a client environment definition that cannot be specified by the JDBC driver is specified for a variable, the specification is ignored. Note that the variable names are case sensitive.

For details about the priorities among connection information items that have multiple setting methods, see *17.11 Connection information priorities*.

This method does not check each specification of the client environment definitions. The specification values are checked during connection to the database, and an `SQLException` is thrown if an error is detected.

(f) Exceptions

None.

17.7.36 getEnvironmentVariables

(a) Function

Gets the client environment definitions for HiRDB.

(b) Format

```
String void getEnvironmentVariables()
```

(c) Arguments

None.

(d) Return value

`String`

This shows the client environment definitions of HiRDB. If no definitions have been specified, the null value is returned.

(e) Functional detail

Gets the client environment definitions of HiRDB.

(f) Exceptions

None.

17.7.37 setEncodeLang

(a) Function

Sets the name of the conversion character set for data conversion.

(b) Format

```
public void setEncodeLang ( String encode_lang ) throws
SQLException
```

(c) Arguments

`String encode_lang`

Specifies the name of the conversion character set. You must select a name from the list of encodings shown under *Internationalization* in the *Java*[TM] *2 SDK, Standard Edition* documentation.

The table below shows the HiRDB character encodings and the corresponding conversion character sets.

HiRDB character encoding (character encoding set with pdntenv or pdsetup command)	Conversion character set to be specified
`lang-c`	`ISO8859_1`
`sjis`	`SJIS` or `MS932`#
`ujis`	`EUC_JP`
`utf-8`	`UTF-8`

HiRDB character encoding (character encoding set with pdntenv or pdsetup command)	Conversion character set to be specified
chinese	EUC_CN

Note:

If the specified conversion character set name is not in compliance with the applicable name shown in this table, the operation of the JDBC driver is not guaranteed.

#

The specification of SJIS or MS932 depends on the handling of Windows special characters in the application.

When OFF is specified, the JDBC driver assumes that the applicable conversion character set name shown in this table was specified. If the HiRDB character encoding is sjis, the conversion character set determined by the OS running the JDBC driver is as follows.

In UNIX: SJIS

In Windows: MS932

If the null value is specified, the current conversion character set name that had been set by this method is invalidated, and the setting is returned to its initial status.

Note that the specification values are case sensitive (except for OFF).

(d) Return value

None.

(e) Functional detail

The conversion character set that was specified by this method is used for carrying out the following data conversions:

- Conversion to character data (Unicode) when the application uses String to get data that was retrieved from HiRDB
- Conversion to binary data when the application uses String to set a value in HiRDB

If this method is not specified, the JDBC driver converts characters using the applicable conversion character set shown in the table above. However, the JDBC driver uses the default character conversion set of the Java Virtual Machine to convert the following items:

- Specification value of `setUapName`
- Authorization identifier and password (values specified by `setUser`, `setPassword`, and `getConnection`)
- Specification values of client environment definitions specified by `setEnvironmentVariables`
- Specification values of environment variables specified by the environment variable group name of the HiRDB client

(f) Exceptions

If the specified conversion character set is not supported by the Java Virtual Machine, this method throws an `SQLException`.

17.7.38 getEncodeLang

(a) Function

Gets the conversion character set for data conversion name that was set.

(b) Format

```
public String getEncodeLang()
```

(c) Arguments

None.

(d) Return value

`String`

This is the conversion character set name. If a conversion character set name was not specified by the `setEncodeLang` method, the null value is returned.

(e) Functional detail

Returns the conversion character set name that was set by the `setEncodeLang` method.

(f) Exceptions

None.

17.7.39 setMaxBinarySize

(a) Function

Sets the maximum data size for retrieval of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public void setMaxBinarySize ( int size ) throws SQLException
```

(c) Arguments

`int size`

Specifies the maximum data size, in the range from 0 to 2,147,483,647.

If `0` is specified, the defined length of the data to be retrieved is set as the maximum size.

(d) Return value

None.

(e) Functional detail

Sets the maximum data size (bytes) when data of the `LONGVARBINARY` JDBC SQL type is retrieved.

When the JDBC driver retrieves `LONGVARBINARY` data, it allocates memory of the defined length because it cannot recognize the actual data length until it retrieves the data. Consequently, if the JDBC driver retrieves values from a column that is very large, such as 2,147,483,647 bytes (the maximum size for HiRDB's `BINARY` and `BLOB` data types), it attempts to allocate memory space of the defined length (2,147,483,647 bytes) as the defined length. Depending on the execution environment, this may cause a memory shortage.

You should use this method to specify the maximum length of the data that is actually stored. If the defined length of the `BINARY` and `BLOB` data to be retrieved is larger than the size specified by this method, the JDBC driver truncates the retrieved data to the specified size. When data has been truncated, the JDBC driver receives a warning from the HiRDB server when the next method of `ResultSet` is executed. The JDBC driver responds to the received warning by throwing an `SQLException` or issuing (or ignoring) an `SQLWarning`, as specified by the `setLONGVARBINARY_TruncError` value.

If a maximum data size has not been set by this method, the defined length of the data to be retrieved is used as the maximum data size.

(f) Exceptions

If a negative value is specified, this method throws an `SQLException`.

(g) Notes

Any value specified for this method is not effective when `LOCATOR` is specified in the `mode` argument of the `setLONGVARBINARY_Access` method. In such a case, the JDBC driver allocates an area based on the actual data length and retrieves all of the data.

17.7.40 getMaxBinarySize

(a) Function

Gets the maximum data size for retrieval of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public int getMaxBinarySize()
```

(c) Arguments

None.

(d) Return value

`int`

This is the value that was set as the maximum data size.

(e) Functional detail

Returns the maximum data size for retrieving data of the `LONGVARBINARY` type (a JDBC SQL type), as set by the `setMaxBinarySize` method.

If a maximum data size has not been set by the `setMaxBinarySize` method, `0` is returned.

(f) Exceptions

None.

17.7.41 setStatementCommitBehavior

(a) Function

Sets whether or not statement objects are to remain valid after a transaction is committed. Here, *statement objects* refer to the following classes:

- `Statement` class
- `PreparedStatement` class

(b) Format

```
public void setStatementCommitBehavior ( boolean mode ) throws
SQLException
```

(c) Arguments

`boolean mode`

Specifies whether statement objects are to be valid both before and after a transaction is terminated by commit processing:

`true`: Validate statement objects after a transaction is completed.

`false`: Invalidate statement objects after a transaction is completed.

(d) Functional detail

Sets whether or not statement objects are to remain valid after a transaction is committed. The default when this method is not called is `true`.

Executing this method is the same as setting the `STATEMENT_COMMIT_BEHAVIOR` item, which is shown in *17.2.2(1) URL syntax*.

(e) Exceptions

None.

(f) Notes

See *17.2.2(1)(c) Notes about specification of HIRDB_CURSOR and STATEMENT_COMMIT_BEHAVIOR*.

17.7.42 getStatementCommitBehavior

(a) Function

Gets setting information about whether or not statement objects are to remain valid even after a transaction is committed. Here, *statement objects* refer to the following classes:

- `Statement` class
- `PreparedStatement` class

(b) Format

```
public boolean getStatementCommitBehavior() throws SQLException
```

(c) Arguments

None.

(d) Return value

`boolean`

Indicates whether statements objects are to remain valid after a transaction is terminated by commit processing:

`true`: The statement objects are to remain valid.

`false`: The statement objects are not to remain valid.

(e) Functional detail

Gets the setting information about whether objects of the following classes are to remain valid after commit execution:

- `Statement` class
- `PreparedStatement` class

This method returns the setting value of the `setStatementCommitBehavior` method. If no value has been set, `true` is returned.

(f) Exceptions

None.

(g) Notes

None.

17.7.43 setLONGVARBINARY_AccessSize

(a) Function

Sets the `LONGVARBINARY` (a JDBD SQL type) data length for one access request to the HiRDB server.

(b) Format

```
public void setLONGVARBINARY_AccessSize ( int access_size )
throws SQLException
```

(c) Arguments

`int access_size`

Specifies the data length (kilobytes) to be requested. The specification value range is from 0 to 2,097,151 (the default is `0`). If `0` is specified, the entire data is requested at once.

(d) Return value

None.

(e) Functional detail

Sets the `LONGVARBINARY` (a JDBC SQL type) data length for one access request to the HiRDB server.

For example, if `20` is specified for the `access_size` argument and the application uses the `getBytes` method of `ResultSet` to retrieve 100 kilobytes of `LONGVARBINARY` data stored in the database, the JDBC driver retrieves the data by dividing the operation into five executions of 20 kilobytes each.

This specification value becomes invalid if a value other than `LOCATOR` is specified in the `mode` argument of the `setLONGVARBINARY_Access` method.

Specifying a value for this method is equivalent to setting the `HiRDB_for_Java_LONGVARBINARY_ACCESS_SIZE` property, which is shown in *17.2.2(2) User properties*.

(f) Exceptions

If a value outside the range from 0 to 2,097,151 is specified in the `access_size` argument, this method throws a `java.sql.SQLException`.

(g) Notes

See *17.2.2(2)(q) Notes about specification of LONGVARBINARY_ACCESS*.

17.7.44 getLONGVARBINARY_AccessSize

(a) Function

Gets the `LONGVARBINARY` (a JDBC SQL type) data length for one access request to the HiRDB server.

(b) Format

```
public int getLONGVARBINARY_AccessSize() throws SQLException
```

(c) Arguments

None.

(d) Return value

`int`

This is the data length (kilobytes) for one access request. If a value has not been set, `0` is returned.

(e) Functional detail

Gets the `LONGVARBINARY` (JDBC SQL type) data length for one access request to the HiRDB server. This method returns the setting value of the `setLONGVARBINARY_AccessSize` method.

(f) Exceptions

None.

17.7.45 setLONGVARBINARY_TruncError

(a) Function

Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public void setLONGVARBINARY_TruncError ( boolean mode ) throws
SQLException
```

(c) Arguments

`boolean mode`

Specifies whether or not an exception is to be thrown when truncation occurs:

`true`

Throw an exception.

`false`

Do not throw an exception.

(d) Return value

None.

(e) Functional detail

Sets whether or not an exception is to be thrown if truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type). If this method is not set, the JDBC driver assumes that `true` was specified.

The specification value of this method becomes invalid if `IGNORE` is specified in the `warningLevel` argument of the `setSQLWarningLevel` method. In such a case, the JDBC driver operates as if `false` were specified.

A truncation that occurs when `LONGVARBINARY` data is retrieved refers to the action that occurs when the flowing conditional expression is satisfied:

actual-length-of-LONGVARBINARY-data-retrieved-by-SQL-execution >
data-length-specified-by-setMaxBinarySize

(f) Exceptions

None.

17.7.46 getLONGVARBINARY_TruncError

(a) Function

Gets the setting information about whether or not an exception is to be thrown if truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type).

(b) Format

```
public boolean getLONGVARBINARY_TruncError()
```

(c) Arguments

None.

(d) Return value

`boolean`

This is the setting information about whether not an exception is to be thrown when truncation occurs:

`true`

An exception is thrown.

`false`

An exception is not thrown.

(e) Functional detail

Gets the setting information about whether or not an exception is to be thrown when truncation occurs during acquisition of data of the `LONGVARBINARY` type (a JDBC SQL type).

(f) Exceptions

None.

17.8 Data types

17.8.1 Mapping SQL data types

There is not an exact match between HiRDB's SQL data types and JDBC's SQL data types. For this reason, the JDBC driver performs mapping (conversion) between JDBC's SQL data types and the SQL data types of the HiRDB to be connected. If an unmappable SQL data type is used for data access, the JDBC driver throws an `SQLException`. If an SQL statement that uses HiRDB's `ROW` type, which cannot be mapped to any of JDBC's SQL data types, is executed for an HiRDB server that uses little endian, the JDBC driver throws an `SQLException` that includes the `KFPA11104-E` message indicating a syntax error.

The SQL data types are mapped with `get`*XXX* and `set`*XXX* methods of the `ResultSet` and `PreparedStatement` classes. For details about the mapping rules for the SQL data types and the `get`*XXX* and `set`*XXX* methods, see the documentation for the JDBC1.0 standard and JDBC2.0 basic standard.

Table 17-30 shows the correspondences between the SQL data types of HiRDB and JDBC.

Table 17-30: SQL data type correspondences between HiRDB and JDBC (Type4 JDBC driver)

HiRDB's SQL data type	JDBC's SQL data type
INTEGER	INTEGER
SMALLINT	SMALLINT
DECIMAL, NUMERIC	DECIMAL (NUMERIC)#1
FLOAT, DOUBLE PRECISION	FLOAT (DOUBLE)#1
SMALLFLT, REAL	REAL
CHAR	CHAR
VARCHAR	VARCHAR (LONGVARCHAR)#1
NCHAR	CHAR
NVARCHAR	VARCHAR (LONGVARCHAR)#1
MCHAR	CHAR
MVARCHAR	VARCHAR (LONGVARCHAR)#1
DATE	DATE

HiRDB's SQL data type	JDBC's SQL data type
TIME	TIME
BLOB	LONGVARBINARY (BINARY, VARBINARY, BLOB)#1
BINARY	LONGVARBINARY (BINARY, VARBINARY, BLOB)#1
TIMESTAMP	TIMESTAMP
BOOLEAN#2	BIT

#1

The data types shown in parentheses are supported only when JDBC's SQL data types are specified in the arguments of the `setNull` or `setObject` method. They are not supported during mapping from HiRDB's SQL data types to JDBC's SQL data types.

#2

This refers to a `BOOLEAN` column in a `ResultSet` object that is generated by the `getTypeInfo` method of `DatabaseMetaData`.

17.8.2 Mapping during retrieval data acquisition

Tables 17-31 and 17-32 show the mapping between `get`*XXX* methods of the `ResultSet` class and JDBC's SQL data types. If a `get`*XXX* method is called for one of JDBC's unmappable SQL data types, the JDBC driver throws an `SQLException`.

Table 17-31: Mapping between getXXX methods of the ResultSet class and JDBC's SQL data types (1/2)

getXXX method	JDBC's SQL data type				
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL
getByte	Y	Y	Y	Y	Y
getShort	Rec.	Y	Y	Y	Y
getInt	Y	Rec.	Y	Y	Y
getLong	Y	Y	Y	Y	Y
getFloat	Y	Y	Y	Rec.	Y
getDouble	Y	Y	Rec.	Y	Y
getBigDecimal	Y	Y	Y	Y	Rec.
getBoolean	Y	Y	Y	Y	Y

getXXX method	JDBC's SQL data type				
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL
getString	Y	Y	Y	Y	Y
getBytes	--	--	--	--	--
getDate	--	--	--	--	--
getTime	--	--	--	--	--
getTimestamp	--	--	--	--	--
getAsciiStream	--	--	--	--	--
getBinaryStream	--	--	--	--	--
getObject	Y	Y	Y	Y	Y
getCharacterStream	--	--	--	--	--
getBlob	--	--	--	--	--

Legend:

Rec.: Mapping is recommended.

Y: Can be mapped. Note, however, that data loss or an error may occur depending on the format of the mapping-source data.

--: Cannot be mapped.

Table 17-32: Mapping between getXXX methods of the ResultSet class and JDBC's SQL data types (2/2)

getXXX method	JDBC's SQL data type				
	CHAR	VARCHAR	DATE	TIMESTAMP	LONGVARBINARY
getByte	Y#1	Y#1	--	--	--
getShort	Y#1	Y#1	--	--	--
getInt	Y#1	Y#1	--	--	--
getLong	Y#1	Y#1	--	--	--
getFloat	Y#1	Y#1	--	--	--
getDouble	Y#1	Y#1	--	--	--

getXXX method	JDBC's SQL data type				
	CHAR	VARCHAR	DATE	TIMESTAMP	LONGVARBINARY
`getBigDecimal`	Y#1	Y#1	--	--	--
`getBoolean`	Y	Y	--	--	--
`getString`	Rec.	Rec.	Y	Y	Y
`getBytes`	--	--	--	--	Y
`getDate`	Y#1	Y#1	Rec.#2	Y	--
`getTime`	Y#1	Y#1	--	Y	--
`getTimestamp`	Y#1	Y#1	Y	Rec.	--
`getAsciiStream`	Y	Y	--	--	Y
`getBinaryStream`	--	--	--	--	Rec.
`getObject`	Y	Y	Y	Y	Y
`getCharacterStream`	Y	Y	--	--	Y
`getBlob`	--	--	--	--	--

Legend:

Rec.: Mapping is recommended

Y: Can be mapped. Note, however, that data loss or a conversion error may occur depending on the format of the conversion-source data.

--: Cannot be mapped.

#1

If there are any single-byte spaces preceding or following the character string data retrieved from the database during conversion by this method, the JDBC driver removes them. After removing the single-byte spaces, the JDBC driver converts the data to the Java data type returned by the `get`*XXX* method.

Note the following items when data is converted to a Java data type:

- If the character string data contains a fractional part and the `getByte`, `getInt`, `getShort`, or `getLong` method is executed, the JDBC driver discards the fractional part and then converts and returns only the integer.
- If the character string data contains double-byte characters, the JDBC driver throws an `SQLException` without converting the data. Double-byte characters

include double-byte spaces used for padding when a character string shorter than the defined column length is stored in a column of HiRDB's `NCHAR` data type.

- If overflow occurs after character string data is converted to a Java data type, the JDBC driver throws an `SQLException`.

#2

When the JDBC SQL type is the `DATE` type and a `java.util.Calender` object is specified in a `setDate` method that is then executed, the JDBC driver uses the specified `java.util.Calendar` object to convert the data, discards the time data, and stores only the date data in the database. Because the time data is discarded, if an application specifies a `java.util.Calendar` object in the `getDate` method and executes the method to retrieve the data that was stored with the `setDate` method, the retrieved date may differ from the one that was specified in the `setDate` method.

Example

In this example, the UAP uses Japan Standard Time as the default time zone and specifies a `java.util.Calendar` object that uses Greenwich Mean Time in the `setDate` and `getDate` methods.

When the UAP specifies a `java.sql.Date` object representing `2005-10-03` in the `setDate` method and executes the method, the JDBC driver supplements `00:00:00` to the time portion, and then subtracts 9 hours because of the time zone difference. The result is `2005-10-02 15:00:00`, and the JDBC driver stores the date portion `2005-10-02` of the result in the database. When the UAP uses the `getDate` method to retrieve this data, the JDBC driver gets the date portion `2005-10-02` from the database, supplements `00:00:00` to the time portion, and adds 9 hours because of the time difference to produce `2005-10-02 09:00:00`. Consequently, in the `java.sql.Date` object used as the return value of the `getDate` method, the JDBC driver sets `2005-10-02`, which differs from `2005-10-03`, as was specified in the `SetDate` method.

17.8.3 Mapping when a ? parameter is set

Table 17-33 lists the `set`*XXX* methods of the `PreparedStatement` class and shows the corresponding JDBC SQL types that are mapped. If a JDBC SQL type cannot be used, the `set`*XXX* method throws an `SQLException`.

The `setCharacterStream` method has been added as a replacement for the `setUnicodeStream` method, because the JDBC2.0 basic standard does not recommend the latter method.

Table 17-33: JDBC SQL types mapped by the setXXX methods of the PreparedStatement class

setXXX method of PreparedStatement class	Mapped JDBC SQL type
`setCharacterStream`	`CHAR` or `VARCHAR`
`setRef`#	`REF`
`setBlob`	`LONGVARBINARY`
`setClob`#	`CLOB`
`setArray`	`ARRAY`

#

The JDBC driver cannot use this method.

Tables 17-34 and 17-35 show the mapping between the `set`*XXX* methods of the `PreparedStatement` class and JDBC's various SQL types.

Table 17-34: Mapping between the setXXX methods of the PreparedStatement class and JDBC's SQL data types (1/2)

setXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL#3	CHAR
`setByte`	Y	Y	Y	Y	Y	Y
`setShort`	Rec.	Y	Y	Y	Y	Y
`setInt`	Y	Rec.	Y	Y	Y	Y
`setLong`	Y	Y	Y	Y	Y	Y
`setFloat`	Y	Y	Y	Rec.	Y	Y
`setDouble`	Y	Y	Rec.	Y	Y	Y
`setBigDecimal`	Y	Y	Y	Y	Rec.	Y
`setBoolean`	Y	Y	Y	Y	Y	Y
`setString`	Y	Y	Y	Y	Y	Rec.
`setBytes`	--	--	--	--	--	--
`setDate`	--	--	--	--	--	Y
`setTime`	--	--	--	--	--	Y

setXXX method	JDBC's SQL data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL#3	CHAR
setTimestamp#1	--	--	--	--	--	Y
setAsciiStream	--	--	--	--	--	Y
setBinaryStream	--	--	--	--	--	--
setObject#2	Y	Y	Y	Y	Y	Y
setCharacterStream	--	--	--	--	--	Y#3
setBlob	--	--	--	--	--	--

Legend:

Rec.: Mapping is recommended

Y: Can be mapped. Note, however, that data loss or a conversion error may occur depending on the format of the conversion-source data.

--: Cannot be mapped.

#1

If a `set`*XXX* method specifies a value for a `?` parameter of HiRDB's `TIMESTAMP` data type, and the `?` parameter and the value have different precisions for the fractional seconds part, the JDBC driver performs one of the following operations:

- When the value has a larger fractional seconds precision than the `?` parameter: truncates the fractional seconds part of the value.
- When the value has a smaller fractional seconds precision than the `?` parameter: expands the fractional seconds part of the value.

#2

Objects of the `InputStream` class and the `Reader` class (including subclasses) cannot be specified in the `setObject` method.

#3

If a `set`*XXX* method specifies a value for a `?` parameter of HiRDB's `DECIMAL` data type, and the `?` parameter and the value have different precisions and decimal scaling positions, the JDBC driver performs one of the following operations:

- When the value has a larger precision than the `?` parameter: throws an `SQLException`.

- When the value has a smaller precision than the `?` parameter: expands the precision.
- When the value has a larger decimal scaling position than the `?` parameter: truncates the value according to the actual scaling position.
- When the value has a smaller decimal scaling position than the `?` parameter: adds zeros to expand the decimal scaling position.

Table 17-35: Mapping between the setXXX methods of the PreparedStatement class and JDBC's SQL data types (2/2)

setXXX method	JDBC's SQL data type					
	VARCHAR	DATE	TIME	TIMESTAMP	LONGVAR BINARY	BLOB
`setByte`	Y	--	--	--	--	--
`setShort`	Y	--	--	--	--	--
`setInt`	Y	--	--	--	--	--
`setLong`	Y	--	--	--	--	--
`setFloat`	Y	--	--	--	--	--
`setDouble`	Y	--	--	--	--	--
`setBigDecimal`	Y	--	--	--	--	--
`setBoolean`	Y	--	--	-	--	--
`setString`	Rec.	Y	Y	Y	Y	--
`setBytes`	--	--	--	-	Y	--
`setDate`	Y	Rec.#4	--	Y	--	--
`setTime`	Y	--	Rec.	Y	--	--
`setTimestamp`#1	Y	Y	--	Rec.	--	--
`setAsciiStream`	Y	--	--	--	Y	--
`setBinaryStream`	--	--	N	--	Y	--
`setObject`#2	Y	Y	Y	Y	Y	Y
`setCharacterStream`	Y#3	--	--	--	Y#3	--
`setBlob`	--	--	--	-	Y	--

Legend:

Rec.: Mapping is recommended

Y: Can be mapped. Note, however, that data loss or a conversion error may occur depending on the format of the conversion-source data.

--: Cannot be mapped.

#1

If a `set`*XXX* method specifies a value for a `?` parameter of HiRDB's `TIMESTAMP` data type, and the `?` parameter and the value have different precisions for the fractional seconds part, the JDBC driver performs one of the following operations:

- If the value has a larger fractional seconds precision than the `?` parameter: truncates the fractional seconds part of the value.
- If the value has a smaller fractional seconds precision than the `?` parameter: expands the fractional seconds part of the value.

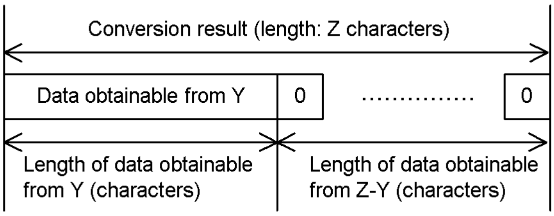
#2

Objects of the `InputStream` class and the `Reader` class (including subclasses) cannot be specified in the `setObject` method.

#3

If the length of the data that can be retrieved from a `java.io.Reader` object is shorter than the length specified in the arguments, the JDBC driver adds zeros as shown below until the length specified in the arguments is reached:

```
int X,Z;
java.io.Reader Y;
setCharacterStream(X,Y,Z)
```



#4

When the JDBC SQL type is the `DATE` type and a `java.util.Calender` object is specified in a `setDate` method that is then executed, the JDBC driver uses the specified `java.util.Calendar` object to convert the data, discards the time data, and stores only the date data in the database. Because the time data is discarded, if an application specifies a `java.util.Calendar` object in the `getDate` method and executes the method to retrieve the data that was stored with the `setDate` method, the retrieved date may differ from the one that was specified in the `setDate` method.

Example

In this example, the UAP uses Japan Standard Time as the default time zone and specifies a `java.util.Calendar` object that uses Greenwich Mean Time in the `setDate` and `getDate` methods.

When the UAP specifies a `java.sql.Date` object representing `2005-10-03` in the `setDate` method and executes the method, the JDBC driver supplements `00:00:00` to the time portion, and then subtracts 9 hours because of the time zone difference. The result is `2005-10-02 15:00:00`, and the JDBC driver stores the date portion `2005-10-02` of the result in the database. When the UAP uses the `getDate` method to retrieve this data, the JDBC driver gets the date portion `2005-10-02` from the database, supplements `00:00:00` to the time portion, and adds 9 hours because of the time difference to produce `2005-10-02 09:00:00`. Consequently, in the `java.sql.Date` object used as the return value of the `getDate` method, the JDBC driver sets `2005-10-02`, which differs from `2005-10-03`, which was specified in the `SetDate` method.

17.8.4 Data conversion of TIME, DATE, and TIMESTAMP columns

(1) setTime, setDate, setTimestamp, and setString methods

This item explains the conversion process when data of HiRDB's `TIME`, `DATE`, or `TIMESTAMP` data type is set in the `setTime`, `setDate`, `setTimestamp`, or `setString` method.

When the `setTime`, `setDate`, `setTimestamp`, or `setString` method is used to set data in a column of HiRDB's `TIME`, `DATE`, or `TIMESTAMP` data type, data conversion takes place according to the HiRDB data type.

Table 17-36 shows the conversion processing for combinations of the different column data types and methods.

Table 17-36: Conversion processing for combinations of the TIME, DATE, and TIMESTAMP types and the setXXX methods

setXXX method	HiRDB data type		
	TIME type	DATE type	TIMESTAMP type
`setTime(Time Obj)`#1	Stores the UAP setting value in the database without any conversion.	Throws an `SQLException`.	Stores in the database data that has `1970-01-01` added before the UAP setting value *hh*:*mm*:*ss*`[.000000]`.
`setDate(Date Obj)`#2	Throws an `SQLException`.	Stores the UAP setting value in the database without any conversion.	Stores data in the database that has `00:00:00[.000000]` added after the UAP setting value *yyyy-MM-DD*.
`setTimestamp(Timestamp Obj)`#3	Throws an `SQLException`.	Stores in the database the data formed when *yyyy-MM-DD* is removed from the UAP setting value.	Stores the UAP setting value in the database without any conversion.
`setString(`character string in *hh*:*mm*:*ss* format`)`	Converts the specified time with `java.sql.Time.valueOf()` and stores the result in the database.#5	Throws an `SQLException`.	Throws an `SQLException`.
`setString(`character string in *yyyy-MM-DD* format`)`	Throws an `SQLException`.	Converts the specified date with `java.sql.Date.valueOf()` and stores the result in the database.#5	Throws an `SQLException`.
`setString(`character string in *yyyy-MM-DD* **Δ** *hh:mm:ss*`[`*.ffffff*`]` format`)` #4	Throws an `SQLException`.	Throws an `SQLException`.	Converts the specified date/time with `java.sql.Timestamp.valueOf()` and stores the result in the database.#5

Note:

If a non-existent date or time is specified, the specified value is returned by the Java Virtual Machine.

#1

`Time Obj` is an object that has the value of a `java.sql.Time` object with the format *hour*:*minute*:*second*.

#2

`Date Obj` is an object that has the value of the `java.sql.Date` object with the format *year*-*month*-*day*.

#3

`Timestamp Obj` is an object that has the value of the `java.sql.Timestamp` object with the format *year*-*month*-*day* *hour*:*minute*:*second*:*nanosecond*.

#4

For `[.`*ffffff*`]`, the number of digits after the decimal point depends on the precision of HiRDB's `TIMESTAMP` type.

Δ represents a single-byte space character.

#5

The result when a non-existent date or time is specified depends on `java.sql.Time.valueOf()`, `java.sql.Date.valueOf()`, or `java.sql.Timestamp.valueOf()`:

Example 1: `25:00:00` becomes `01:00:00`.

Example 2: `2000-01-32` becomes `2000-02-01`.

Example 3: `1582-10-05` becomes `1582-10-15` (switching from the Julian to the Gregorian calendar).

(2) getTime, getDate, and getTimestamp methods

This item explains the conversion process when data of HiRDB's `TIME`, `DATE`, `TIMESTAMP` or character string (`CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, or `NVARCHAR`) data type is set in the `getTime`, `getDate`, or `getTimestamp` method.

When the `getTime`, `getDate`, or `getTimestamp` method is used to set data in a column of HiRDB's `TIME`, `DATE`, `TIMESTAMP`, or character string data type, data conversion takes place according to the HiRDB data type.

Table 17-37 shows the conversion processing for combinations of the different column data types and methods.

Table 17-37: Conversion processing for combinations of the TIME, DATE, TIMESTAMP, and character string types and the getXXX methods

getXXX method	HiRDB data type			
	TIME type	**DATE type**	**TIMESTAMP type**	**Character string type**
`getTime()`#2	Gets the value stored in the database and sets it as the `java.sql.Time` object without any conversion.#1	Throws an `SQLException`.	Removes the *hour*:*minute*:*second* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Time` object.#1	Gets only an *hh*:*mm*:*ss* character string expression of the `TIME` type as the `java.sql.Time` object. For other expressions, the method throws an exception.
`getDate()`#2	Throws an `SQLException`.	Gets the value stored in the database and sets it as the `java.sql.Date` object without any conversion.#1	Removes the *year-month-day* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Date` object.#1	Gets only a *yyyy-MM-DD* character string expression of the `DATE` type as the `java.sql.Date` object. For other expressions, the method throws an exception.
`getTimestamp()`#2	Throws an `SQLException`.	Appends `00:00:00.000000` to the `DATE` data retrieved from the database and sets the result as the `java.sql.Timestamp` object.	Gets the value stored in the database and sets it as the `java.sql.Timestamp` object without any conversion.	Gets only a *yyyy-MM-DD* Δ *hh:mm:ss* [*.ffffff*] character string expression of the `TIMESTAMP` type as the `java.sql.Timestamp` object (Δ is a single-byte space character). For other expressions, the method throws an `SQLException`.

Legend:

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

#1

The setting value of an unspecified date item (*year-month-day*) is `1970-01-01`, and the setting value of an unspecified time item (*hour*:*minute*:*second*.*millisecond*) is `00:00:00.000000`.

#2

The date and time stored in the database may be different from the date and time obtained from `java.sql.Time`, `java.sql.Date`, and `java.sql.Timestamp`:

Example 1: `25:00:00` becomes `01:00:00`.

Example 2: `2000-01-32` becomes `2000-02-01`.

Example 3: Both `1582-10-05` and `1582-10-15` become `1582-10-15` (the calendar switches from the Julian to the Gregorian calendar).

17.8.5 Overflow handling

This section explains when overflow is set when a program uses a `set`*XXX* method to set a value, or uses a `get`*XXX* method to get a value.

(1) setXXX methods (except for the setObject method)

Tables 17-38 and 17-39 show for each HiRDB data type whether or not overflow occurs when a `set`*XXX* method is used.

Table 17-38: Possibility of overflow when the setXXX method is used (1/2)

setXXX method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`setByte`	--	--	--	--	Y	-
`setShort`	--	--	--	--	Y	--
`setInt`	Y	--	--	--	Y	--
`setLong`	Y	Y	--	--	Y	--
`setFloat`	Y	Y	--	--	Y	--
`setDouble`	Y	Y	--	--	Y	--
`setBigDecimal`	Y	Y	--	--	Y	--
`setBoolean`	--	--	--	--	Y	--
`setString`	Y	Y	--	--	Y	--
`setBytes`	N/A	N/A	N/A	N/A	N/A	N/A
`setDate`	N/A	N/A	N/A	N/A	N/A	--

setXXX method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`setTime`	N/A	N/A	N/A	N/A	N/A	--
`setTimestamp`	N/A	N/A	N/A	N/A	N/A	--
`setBlob`	N/A	N/A	N/A	N/A	N/A	N/A
`setBinaryStream`	N/A	N/A	N/A	N/A	N/A	N/A
`setAsciiStream`	N/A	N/A	N/A	N/A	N/A	--
`setCharacterStream`	N/A	N/A	N/A	N/A	N/A	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 17-39: Possibility of overflow when the setXXX method is used (2/2)

setXXX method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`setByte`	N/A	N/A	N/A	N/A	N/A
`setShort`	N/A	N/A	N/A	N/A	N/A
`setInt`	N/A	N/A	N/A	N/A	N/A
`setLong`	N/A	N/A	N/A	N/A	N/A
`setFloat`	N/A	N/A	N/A	N/A	N/A
`setDouble`	N/A	N/A	N/A	N/A	N/A
`setBigDecimal`	N/A	N/A	N/A	N/A	N/A
`setBoolean`	N/A	N/A	N/A	N/A	N/A
`setString`	Y	--	Y	N/A	N/A
`setBytes`	N/A	N/A	N/A	--	--

setXXX method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`setDate`	Y	N/A	Y	N/A	N/A
`setTime`	N/A	Y	Y	N/A	N/A
`setTimestamp`	Y	N/A	Y	N/A	N/A
`setBlob`	N/A	N/A	N/A	--	--
`setBinaryStream`	N/A	N/A	N/A	--	--
`setAsciiStream`	N/A	N/A	N/A	--	--
`setCharacterStream`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

#

Overflow occurs when the value obtained by the `getTime` method of the `java.sql.Date`, `java.sql.Time`, or `java.sql.Timestamp` class is an object larger than 253,402,268,399,999 or smaller than -62,135,802,000,000. The `getTime` method returns the number of milliseconds since 1970-01-01 00:00:00 (Greenwich Mean Time).

The methods shown below can be used to obtain 253,402,268,399,999 from the maximum value that can be stored in HiRDB's `TIMESTAMP` type, and -62,135,802,000,000 from the minimum value that can be represented by the `java.sql.Timestamp` class.

253,402,268,399,999:

```
Timestamp.valueOf("9999-12-31
23:59:59.999999").getTime()
```

-62,135,802,000,000:

```
Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()
```

(2) setObject method

Tables 17-40 and 17-41 show whether or not overflow occurs for each HiRDB data type when the `setObject` method is used.

Table 17-40: Possibility of overflow when the setObject method is used (1/2)

setObject method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`Byte`	--	--	--	--	Y	--
`Short`	--	--	--	--	Y	--
`Integer`	Y	--	--	--	Y	--
`Long`	Y	Y	--	--	Y	--
`Decimal`	Y	Y	--	--	Y	--
`Float`	Y	Y	--	--	Y	--
`Double`	Y	Y	--	Y	Y	--
`Boolean`	--	--	--	--	Y	--
`String`	Y	Y	---	--	Y	--
`Date`	N/A	N/A	N/A	N/A	N/A	--
`Time`	N/A	N/A	N/A	N/A	N/A	--
`Timestamp`	N/A	N/A	N/A	N/A	N/A	--
`byte[]`	N/A	N/A	N/A	N/A	N/A	--
`Blob`	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 17-41: Possibility of overflow when the setObject method is used (2/2)

setObject method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
`Byte`	N/A	N/A	N/A	N/A	N/A

setObject method	HiRDB data type				
	DATE#	TIME#	TIMESTAMP#	BINARY	BLOB
Short	N/A	N/A	N/A	N/A	N/A
Integer	N/A	N/A	N/A	N/A	N/A
Long	N/A	N/A	N/A	N/A	N/A
Decimal	N/A	N/A	N/A	N/A	N/A
Float	N/A	N/A	N/A	N/A	N/A
Double	N/A	N/A	N/A	N/A	N/A
Boolean	N/A	N/A	N/A	N/A	N/A
String	Y	--	Y	N/A	N/A
Date	Y	N/A	Y	N/A	N/A
Time	N/A	Y	N/A	N/A	N/A
Timestamp	Y	N/A	Y	N/A	N/A
byte[]	N/A	N/A	N/A	--	--
Blob	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

#

Overflow occurs if the value obtained by the `getTime` method of the `java.sql.Date`, `java.sql.Time`, or `java.sql.Timestamp` class is an object larger than 253,402,268,399,999 or smaller than -62,135,802,000,000. The `getTime` method returns the number of milliseconds since 1970-01-01 00:00:00 (Greenwich Mean Time).

The methods shown below can be used to obtain 253,402,268,399,999 from the maximum value that can be stored in HiRDB's `TIMESTAMP` type, and -62,135,802,000,000 from the minimum value that can be represented by the `java.sql.Timestamp` class.

253,402,268,399,999:

```
Timestamp.valueOf("9999-12-31
23:59:59.999999").getTime()
```

-62,135,802,000,000:

```
Timestamp.valueOf("0001-01-01 00:00:00.0").getTime()
```

(3) getXXX methods (except the getObject method)

Tables 17-42 and 17-43 show whether or not overflow occurs for each HiRDB data type when a `get`*XXX* method is used.

Table 17-42: Possibility of overflow when the getXXX method is used (1/2)

getXXX method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string types
`getByte`	Y	Y	Y	Y	Y	Y
`getShort`	--	Y	Y	Y	Y	Y
`getInt`	--	--	Y	Y	Y	Y
`getLong`	--	--	Y	Y	Y	Y
`getFloat`	--	--	--	--	--	--
`getDouble`	--	--	--	--	--	--
`getBigDecimal`	--	--	--	--	--	--
`getBoolean`	--	--	--	--	--	--
`getString`	--	--	--	--	--	--
`getBytes`	N/A	N/A	N/A	N/A	N/A	N/A
`getDate`	N/A	N/A	N/A	N/A	N/A	--
`getTime`	N/A	N/A	N/A	N/A	N/A	--
`getTimestamp`	N/A	N/A	N/A	N/A	N/A	--
`getAsciiStream`	N/A	N/A	N/A	N/A	N/A	--
`getBinaryStream`	N/A	N/A	N/A	N/A	N/A	N/A
`getCharacterStream`	N/A	N/A	N/A	N/A	N/A	--
`getBlob`	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 17-43: Possibility of overflow when the getXXX method is used (2/2)

getXXX method	HiRDB data type				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
`getByte`	N/A	N/A	N/A	N/A	N/A
`getShort`	N/A	N/A	N/A	N/A	N/A
`getInt`	N/A	N/A	N/A	N/A	N/A
`getLong`	N/A	N/A	N/A	N/A	N/A
`getFloat`	N/A	N/A	N/A	N/A	N/A
`getDouble`	N/A	N/A	N/A	N/A	N/A
`getBigDecimal`	N/A	N/A	N/A	N/A	N/A
`getBoolean`	N/A	N/A	N/A	N/A	N/A
`getString`	--	--	--	--	--
`getBytes`	N/A	N/A	N/A	--	--
`getDate`	--	N/A	--	N/A	N/A
`getTime`	N/A	--	--	N/A	N/A
`getTimestamp`	--	N/A	--	N/A	N/A
`getAsciiStream`	N/A	N/A	N/A	--	--
`getBinaryStream`	N/A	N/A	N/A	--	--
`getCharacterStream`	N/A	N/A	N/A	--	--
`getBlob`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

(4) getObject method

Tables 17-44 and 17-45 show whether or not overflow occurs for each HiRDB data type when the `getObject` method is used.

Table 17-44: Possibility of overflow when the getObject method is used (1/2)

getObject method	HiRDB data type					
	SMALLINT	INTEGER	FLOAT	REAL	DECIMAL	Character string type
`Byte`	Y	Y	Y	Y	Y	Y
`Short`	--	Y	Y	Y	Y	Y
`Int`	--	--	Y	Y	Y	Y
`Long`	--	--	Y	Y	Y	Y
`Float`	--	--	Y	--	Y	Y
`Double`	--	--	--	Y	Y	Y
`BigDecimal`	--	--	--	Y	Y	Y
`Boolean`	--	--	--	--	--	--
`String`	--	--	--	--	--	--
`Bytes`	N/A	N/A	N/A	N/A	N/A	N/A
`Date`	N/A	N/A	N/A	N/A	N/A	--
`Time`	N/A	N/A	N/A	N/A	N/A	--
`Timestamp`	N/A	N/A	N/A	N/A	N/A	--
`AsciiStream`	N/A	N/A	N/A	N/A	N/A	--
`BinaryStream`	N/A	N/A	N/A	N/A	N/A	N/A
`Object`	--	--	--	--	--	--
`CharacterStream`	N/A	N/A	N/A	N/A	N/A	--
`Blob`	N/A	N/A	N/A	N/A	N/A	N/A

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

N/A: This combination is not allowed.

Character string types: `CHAR`, `MCHAR`, `NCHAR`, `VARCHAR`, `MVARCHAR`, and `NVARCHAR`

Table 17-45: Possibility of overflow when the getObject method is used (2/2)

getObject method	HiRDB data type				
	DATE	TIME	TIMESTAMP	BINARY	BLOB
`Byte`	N/A	N/A	N/A	N/A	N/A
`Short`	N/A	N/A	N/A	N/A	N/A
`Int`	N/A	N/A	N/A	N/A	N/A
`Long`	N/A	N/A	N/A	N/A	N/A
`Float`	N/A	N/A	N/A	N/A	N/A
`Double`	N/A	N/A	N/A	N/A	N/A
`BigDecimal`	N/A	N/A	N/A	N/A	N/A
`Boolean`	N/A	N/A	N/A	N/A	N/A
`String`	--	--	--	N/A	N/A
`Bytes`	N/A	N/A	N/A	--	--
`Date`	--	N/A	--	N/A	N/A
`Time`	N/A	--	--	N/A	N/A
`Timestamp`	--	N/A	--	N/A	N/A
`AsciiStream`	N/A	N/A	N/A	--	--
`BinaryStream`	N/A	N/A	N/A	--	--
`Object`	--	--	--	--	--
`CharacterStream`	N/A	N/A	N/A	--	--
`Blob`	N/A	N/A	N/A	--	--

Legend:

--: Overflow does not occur regardless of the value.

Y: Overflow may occur depending on the value.

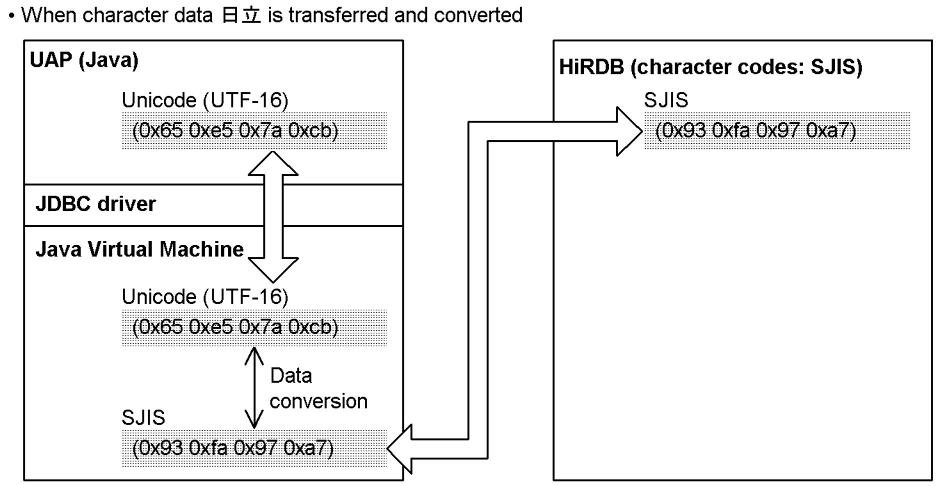
N/A: This combination is not allowed.

17.9 Character conversion facility

Because character codes in Java programs are handled as Unicode, the JDBC driver performs mutual character code conversion between HiRDB character data and Unicode. In this character code conversion process, the JDBC driver uses the encoder provided by the Java Virtual Machine.

Figure 17-1 shows the flow of mutual character code conversion between HiRDB character data and Unicode.

Figure 17-1: Flow of mutual character code conversion between HiRDB character data and Unicode



When the JDBC driver exchanges character data with HiRDB, it specifies the character set name to the encoder of the Java Virtual Machine. At this time, the JDBC driver gets the character encoding of the HiRDB server and specifies the character set name that corresponds to that encoding. If a character set name was specified by the `ENCODELANG` property or by the `setEncodeLang` method when the connection was established, the specified character set name is specified with priority to the encoder of the Java Virtual Machine. Therefore, if a character set name that does not correspond to the character encoding of the HiRDB server is specified in the `ENCODELANG` property or by the `setEncodeLang` method, an error occurs during character code conversion.

17.10 Supported client environment definitions

Table 17-46 lists the client environment definitions that can be specified with the JDBC driver. The numbers in the list correspond to the numbers of the individual environment variables in *6.6.4 Environment definition information*.

Table 17-46: Client environment variables that can be specified with the JDBC driver

No.	Environment variable name	Function	Environment variable type
1	PDHOST	Specifies the host name of the HiRDB server to be connected.	System configuration
2	PDNAMEPORT	Specifies the port number of the HiRDB server.	
3	PDFESHOST	Specifies the host name of the front-end server.	
4	PDSERVICEGRP	Specifies the server name of the single server or front-end server.	
5	PDSRVTYPE	Specifies the HiRDB server type.	
6	PDSERVICEPORT	Specifies the port number for high-speed connection.	
8	PDCLTRCVPORT	Specifies the client receive port number.	
9	PDCLTRCVADDR	Specifies the IP address or host name of the client.	
19	PDUSER	Specifies the authorization identifier and password. In UNIX, this environment variable can be omitted.	User execution environment
20	PDCLTAPNAME	Specifies UAP identification information (UAP identifier) of the UAP that accesses the HiRDB server.	
23	PDDBLOG	Specifies whether or not the database update log is to be retrieved when the 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 representing one character.	

No.	Environment variable name	Function	Environment variable type
29	PDCLTGRP	Specifies a client group name when the connection frame guarantee facility for client groups is used.	
31	PDAUTORECONNECT	Specifies whether or not the automatic reconnect facility is to be used.	
32	PDRCCOUNT	Specifies the number of times the CONNECT statement is retried by the automatic reconnect facility.	
33	PDRCINTERVAL	Specifies the CONNECT retry interval at which the automatic reconnect facility executes reconnect processing.	
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 to be applied to processing when a page accessed by the UAP is cached to the global buffer.	
36	PDHATRNQUEUING	Specifies that the client does not use the transaction queuing facility.	
46	PDCWAITTIME	Specifies the maximum time that the HiRDB client waits for a response from the HiRDB server after sending a request to the HiRDB server.	System monitoring
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 server to arrive after returning a response to the previous request from the HiRDB client. This function monitors the time other than the transaction processing time.	

No.	Environment variable name	Function	Environment variable type
49	PDCWAITTIMEWRNPNT	Specifies the output timing of the SQL runtime warning information file when the SQL runtime warning output facility is used. The output timing is specified as a percentage of the maximum time that the HiRDB client waits, or as an amount of time.	
54	PDCONNECTWAITTIME	Specifies the maximum time that the HiRDB client waits for a response from the HiRDB server when it connects with the HiRDB server.	
55	PDCLTPATH	Specifies the storage directory for SQL trace files and error log files created by the HiRDB client.	Troubleshooting
56	PDSQLTRACE#	Specifies the size (bytes) of the SQL trace file into which SQL trace information for the UAP is to be output.	
59	PDPRMTRC	Specifies whether parameter information and retrieval data are to be output in the SQL trace information.	
60	PDPRMTRCSIZE	Specifies the maximum data length of the parameter information and retrieval data to be output in the SQL trace information.	
62	PDUAPREPLVL	Specifies output information for UAP statistical reports.	
63	PDREPPATH	Specifies whether UAP statistical report files are to be output to a different directory from the directory specified by PDCLTPATH.	
64	PDTRCPATH	Specifies the storage directory for dynamic SQL trace files.	
66	PDSQLTEXTSIZE	Specifies the size of the SQL statement to be output to the SQL trace.	
68	PDRCTRACE	Specifies the size of the output file for the UAP reconnect trace information.	
69	PDWRTLNPATH	Specifies the storage directory for files to which value expression values of WRITE LINE statements are to be output.	

No.	Environment variable name	Function	Environment variable type
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.	
74	PDVWOPTMODE	Specifies whether the access path information file is to be retrieved.	Access path information file for the access path display utility
78	PDSTJTRNOUT	Specifies whether UAP statistical information is to be output to a statistical log file for each transaction.	Output unit for UAP statistical information
79	PDLOCKLIMIT	Specifies the maximum number of lock requests that a UAP can issue to one server.	Lock
80	PDDLKPRIO	Specifies the deadlock priority value of the UAP.	
81	PDLOCKSKIP	Specifies whether an unlocked conditional search is to be performed.	
82	PDFORUPDATEEXLOCK	Specifies whether WITH EXCLUSIVE LOCK is to be applied to the lock option of SQL statements in which the FOR UPDATE clause is specified (or assumed).	
83	PDISLLVL	Specifies the data guarantee level of an SQL statement	SQL-related
84	PDSQLOPTLVL	Specifies optimization methods (SQL optimization options) for determining the most efficient access path by taking the database status into consideration.	
85	PDADDITIONALOPTLVL	Specifies optimization methods (SQL extension optimizing methods) for determining the most efficient access path by taking the database status into consideration.	
86	PDHASHTBLSIZE	Specifies the hash table size when hash join, subquery hash execution is applied in SQL optimization.	

No.	Environment variable name	Function	Environment variable type
88	PDAGGR	Specifies the maximum number of groups allowed in each server so that the memory size used in GROUP BY processing can be determined.	
89	PDCMMTBFDDL	When a definition SQL is to be executed in a transaction that is executing a data manipulation SQL, specifies whether the transaction is to be committed automatically before the definition SQL is executed.	
90	PDPRPCRCLS	Specifies whether an open cursor is to be closed automatically if a PREPARE statement reuses the SQL identifier that is using that open cursor.	
92	PDDDLDEAPRP	Specifies whether definition information of a table being used by a closed holdable cursor can be changed by another UAP between transactions.	
94	PDDELRSVWDFILE	Specifies the name of the SQL reserved word deletion file when the SQL reserved word deletion facility is used.	
95	PDHJHASHINGMODE	Specifies the hashing method when application of **hash join, subquery hash execution** is selected as the SQL extension optimizing option.	
96	PDBLKF	Specifies the number of rows to be sent in one transfer when the HiRDB server transfers retrieval results to the HiRDB client.	Block transfer facility
97	PDBINARYBLKF	Specifies whether the block transfer facility is to be applied when a table with a BINARY-type selection expression with a defined length exceeding 32,00 bytes is searched.	
98	PDBLKBUFFSIZE	Specifies the size of the server-client communication buffer used by the block transfer facility.	
100	PDDBACCS	When the inner replica facility is being used and an RDAREA that is not the current RDAREA is to be accessed, specifies that RDAREA's generation number.	Inner replica facility

No.	Environment variable name	Function	Environment variable type
101	PDDBORGUAP	Specifies whether to execute a UAP on the original RDAREA that is in online reorganization hold status.	Updatable online reorganization
102	PDSPACELVL	Specifies the space conversion level for data storage, comparison, and search processing.	Data space conversion
106	PDCNSTRNTNAME	Specifies the position of the constraint name definition when a referential or check constraint is defined.	Referential and check constraints
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 transferred in one transfer operation when retrieval results are transferred from a distributed server to a distributed client.	Distributed database
117	PDPLGIXMK	Specifies whether delayed batch creation of plug-in indexes is to be used.	Plug-ins
118	PDPLUGINNSUB	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 log file output destination for Exception trace logs in the Type4 JDBC driver.	JDBC driver
122	PDJDBFILEOUTNUM	Specifies the number of Exception trace logs that the Type4 JDBC driver outputs to the log file.	
123	PDJDBONMEMNUM	Specifies the number of Exception trace logs acquired in memory by the Type4 JDBC driver.	

No.	Environment variable name	Function	Environment variable type
124	PDJDBTRACELEVEL	Specifies the trace acquisition level for Exception trace logs in the Type4 JDBC driver.	

#

The name of the SQL trace file is pdjsql*xxxxxxxx*_*ppppp*_1.trc or pdjsql*xxxxxxxx*_*ppppp*_2.trc.

xxxxxxxx: Name of connected server (up to 8 characters)

ppppp: Receive port number (5 characters) at the client side

This format is used even when the SQL trace file is acquired by the UAP statistical report facility (PDREPPATH specification) or by the SQL trace dynamic acquisition facility (PDTRCPATH specification). However, if the SQL trace file is acquired before connection to the FES or SDS, the file name becomes pdjsql1.trc or pdjsql2.trc.

17.11 Connection information priorities

(1) List of connection information priorities

The JDBC driver enables you to specify synonymous connection information by using multiple setup methods (for example, DBHOST specified in the URL and PDHOST specified in HiRDB client environment variables). Table 17-47 lists the connection information items that have multiple setup methods, and the priorities when items are set concurrently by multiple setup methods.

Table 17-47: Priorities for connection information

Meaning of connection information	**Setup method**	**Priority**		
		A	**B**	**C**
HiRDB host name	DBHOST in URL	1	--	--
	PDHOST in HiRDB client environment variables specified by HiRDB_for_Java_ENV_VARIABLES in Properties argument of DriverManager.getConnection	2	--	--
	PDHOST in HiRDB environment variable group specified by DBID in URL	3	--	--
	setDBHostName method of DataSource interface	--	1	1
	PDHOST in HiRDB client environment variables specified by setEnvironmentVariables method of DataSource interface	--	2	2
	PDHOST in HiRDB environment variable group specified by setDescription method of DataSource interface	--	3	--
	PDHOST in HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	3

Meaning of connection information	Setup method	Priority		
		A	B	C
HiRDB port number	DBID in URL	1	--	--
	PDNAMEPORT in HiRDB client environment variables specified by HiRDB_for_Java_ENV_VARIABLES in Properties argument of DriverManager.getConnection	2	--	--
	PDNAMEPORT in HiRDB environment variable group specified by DBID in URL	3	--	--
	setDescription method of DataSource interface	--	1	1
	PDNAMEPORT in HiRDB client environment variables specified by setEnvironmentVariables method of DataSource interface	--	2	2
	PDNAMEPORT in HiRDB environment variable group specified by setDescription method of DataSource interface	--	3	--
	PDNAMEPORT in HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	3
Authorization identifier used during connection#1	user argument or user in Properties argument of DriverManager.getConnection	1	--	--
	Argument in getConnection method of DataSource interface or argument in getPooledConnection method of ConnectionPoolDataSource interface	--	1	--
	Argument in getXAConnection method of XADataSource interface	--	--	1
	setUser method of DataSource interface	--	2	2
Password used during connection#1	password argument or password in Properties argument of DriverManager.getConnection	1	--	--
	Argument in getConnection method of DataSource interface, or argument in getPooledConnection method of ConnectionPoolDataSource interface	--	1	--
	Argument in getXAConnection method of XADataSource interface	--	--	1
	setPassword method of DataSource interface	--	2	2

Meaning of connection information	Setup method	Priority		
		A	B	C
UAP name[#2]	UAPNAME property in Properties argument of DriverManager.getConnectionoperties	1	--	--
	PDCLTAPNAME in HiRDB client environment variables specified by HiRDB_for_Java_ENV_VARIABLES in Properties argument of DriverManager.getConnection	2	--	--
	PDCLTAPNAME in HiRDB environment variable group specified by DBID in URL	3	--	--
	setUapName method of DataSource interface	--	1	1
	PDCLTAPNAME in HiRDB client environment variables specified by setEnvironmentVariables method of DataSource interface	--	2	2
	PDCLTAPNAME in HiRDB environment variable group specified by setDescription method of DataSource interface	--	3	--
	PDCLTAPNAME in HiRDB environment variable group specified by XADataSource.setXAOpenString	--	--	3
Conversion character set	ENCODELANG property in Properties argument of DriverManager.getConnection	1	--	--
	ENCODELANG in URL	2	--	--
	setEncodeLang of DataSource interface	--	1	1
Cursor operation mode	HIRDB_CURSOR property in Properties argument of DriverManager.getConnection	1	--	--
	HIRDB_CURSOR in URL	2	--	--
	setHiRDBCursorMode of DataSource interface	--	1	1
Status after statement commit execution	HiRDB_for_Java_STATEMENT_COMMIT_BEHAVIOR property in Properties argument of DriverManager.getConnection	1	--	--
	STATEMENT_COMMIT_BEHAVIOR in URL	2	--	--
	setStatementCommitBehavior of DataSource interface	--	1	1

Meaning of connection information	Setup method	Priority		
		A	B	C
Login wait time	`DriverManager.setLoginTimeout`	1	--	--
	`PDCONNECTWAITTIME` in HiRDB client environment variables specified by `HiRDB_for_Java_ENV_VARIABLES` in `Properties` argument of `DriverManager.getConnection`	2	--	--
	`PDCONNECTWAITTIME` in HiRDB environment variable group specified by `DBID` in URL	3	--	--
	`setLoginTimeout` of `DataSource` interface	--	1	1
	`PDCONNECTWAITTIME` in HiRDB client environment variables specified by `setEnvironmentVariables` method of `DataSource` interface	--	2	2
	`PDCONNECTWAITTIME` in HiRDB environment variable group specified by `setDescription` method of `DataSource` interface	--	3	--
	`PDCONNECTWAITTIME` in HiRDB environment variable group specified by `XADataSource.setXAOpenString`	--	--	3

Legend:

A: For connection that uses `DriverManager`

B: For non-XA connection that uses the `DataSource` interface

C: For XA connection that uses the `XADataSource` interface

--: Cannot be specified for the connection method.

#1

For details about priorities when no authorization identifier or password is specified, see *(2) Priorities when the authorization identifier or password is not specified*.

#2

If this information cannot be set with the setting method shown in this table, the JDBC driver operates with the information set by the HiRDB_Type4_JDBC_Driver, which is the product name of the JDBC driver.

(2) Priorities when the authorization identifier or password is not specified

This item shows the priorities when the authorization identifier or password is not specified.

When the authorization identifier is specified by the method shown in Table 17-47 and the password is not specified

The JDBC driver determines the authorization identifier according to the priority sequence shown in Table 17-47. The driver assumes that a password has not been specified.

When the authorization identifier is not specified

The specifications shown in Table 17-48 become effective whether or not a password is specified.

Table 17-48: Specifications that become effective when an authorization identifier is not specified

Specification of PDUSER in HiRDB environment variable group		Authorization identifier and password that become effective
Specification 1	**Specification 2**	
[A] PDUSER is specified in HiRDB client environment variables specified by HiRDB_for_Java_ENV_VARIABLES in Properties argument of DriverManager.getConnection.	--	Specification in [A] becomes effective.
[A] is not specified.	[B] PDUSER is specified in the HiRDB environment variable group specified by DBID in the URL.	Specification in [B] becomes effective.
	[B] is not specified.	Throws an SQLException.
[C] PDUSER is specified in HiRDB client environment variable group specified by setEnvironmentVariables method of DataSource interface.	--	Specification in [C] becomes effective.

Specification of PDUSER in HiRDB environment variable group		Authorization identifier and password that become effective
Specification 1	**Specification 2**	
[C] is not specified.	[D] `PDUSER` is specified in the HiRDB environment variable group specified by the `setDescription` method of the `DataSource` interface.	Specification in [D] becomes effective.
	[E] `PDUSER` is specified in the HiRDB environment variable group specified by the `setXAOpenString` method of the `XADataSource` interface.	Specification in [E] becomes effective.
	[D] and [E] are not specified.	Throws an `SQLException`.

Legend:

--: Not related to the specification value.

17.12 JDBC interface method trace

You can acquire a JDBC interface method trace as troubleshooting information when you call a method of the JDBC interface.

17.12.1 Setup for trace acquisition

(1) Connection with the DriverManager class

Specify a valid log writer by using the `setLogWrite` method of the `DriverManager` class, and specify acquisition of the JDBC interface method trace in the arguments (`Properties info`) of the `getConnection` method.

For details, see *17.2.2(2)(d) JDBC_IF* and *17.2.2(2)(e) TRC_NO*.

(2) Connection with the DataSource class

Specify a valid log writer by using the `setLogWriter` method provided by the `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` interfaces, and specify the `setJDBC_IF_TRC` method provided by the `DataSource`, `ConnectionPoolDataSource`, and `XADataSource` classes, which are provided by the JDBC2.0 Optional Package.

For details, see *17.7.5 setJDBC_IF_TRC* and *17.7.7 setTRC_NO*.

17.12.2 Acquisition rules

This section describes the rules for acquisition of the JDBC interface method trace.

- Trace information is acquired when a method of the JDBC interface is called and when processing is returned from that method.

  However, trace information is not acquired for methods executed before connection to the database.

  Trace information is not acquired for the following methods:

  Driver interface

  - `acceptsURL(String url)`
  - `getMajorVersion()`
  - `getMinorVersion()`
  - `getPropertyInfo(String url, Properties info)`
  - `jdbcCompliant()`

  DataSource interface

  - `getLoginTimeout()`

- getLogWriter()
- setLoginTimeout(int seconds)
- setLogWriter(PrintWriter out)

- Trace information is stored for the number of entries and is output to the specified log writer when the Connection.close method is called (normal termination), or when an SQLException, XAException, or BatchUpdateException is thrown (error occurrence).
- If the number of trace information items exceeds the number of entries, the stored trace information is discarded in chronological sequence and the newest trace information is retained.
- A JDBC interface method trace uses a single-entry trace area for each Entry and each Return.

17.12.3 Output example

Shown below is an output example of a JDBC interface method trace.

Output example

```
[1]                          [2]                        [3]
[HiRDB_Type4_JDBC_Driver][JDBC Interface Entry ][PrdbStatement.executeQuery]
                                                  [4]
[HiRDB_Type4_JDBC_Driver]                         sql=select * from pp
[HiRDB_Type4_JDBC_Driver][JDBC Interface Return][PrdbStatement.executeQuery]
                                                  [5]
[HiRDB_Type4_JDBC_Driver]
Return=JP.co.Hitachi.soft.HiRDB.JDBC.Prdb...
[HiRDB_Type4_JDBC_Driver][JDBC Interface Entry ][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC_Driver][JDBC Interface Return][PrdbResultSet.getMetaData]
[HiRDB_Type4_JDBC_Driver]
Return=JP.co.Hitachi.soft.HiRDB.JDBC.Prdb...
```

Explanation

1. [HiRDB_Type4_JDBC_Driver]

   Name of the JDBC driver

2. [JDBC Interface Entry], [JDBC Interface Return]

   [JDBC Interface Entry]: Calling of the JDBC method

   [JDBC Interface Return]: Return from the JDBC method

3. [*XXXXX*.*YYYYY*]

   YYYYY method of the *XXXXX* class

4. `select * from pp`

   Argument of the JDBC method (for the argument indicating the password, an asterisk (`*`) is output, as in `password=*`)

5. `JP.co.Hitachi.soft.HiRDB.JDBC.Prdb`

   Return value of the JDBC method

17.13 Exception trace log

You can acquire an Exception trace log as troubleshooting information. If a failure caused by an exception occurs in the JDBC driver, the failure cause is output to the Exception trace log.

The following constitute the output contents:

- Information (such as error messages) when an exception occurs
- Execution record of JDBC's API methods up to the point where an exception occurred

When this function is used, information about JDBC's API methods that are called from the UAP is stored in the JDBC driver memory. Then if an `SQLException`, `BatchUpdateException`, or `XAException` occurs, the information stored in memory can be output to a file before the exception is thrown.

17.13.1 Methods to be acquired and setup for log acquisition

(1) Methods to be acquired in the Exception trace log

The information to be acquired in the Exception trace log is the calling and return of methods described in the java.sql package found in the API specifications of Java 2 Platform, Standard Edition, Version 1.4.

Methods that satisfy all of the following conditions are acquired:

- The methods listed in Table 17-49, when a trace acquisition level required for acquisition is specified for each method.
- Methods of the `Blob` and `InputStream` classes when `LOCATOR` is specified in `LONGVARBINARY_ACCESS`.

Methods that only look up and return information found in objects or only store information into objects, such as the `ResultSet.get`*XXX*, `PreparedStatement.set`*XXX*, and `Connection.isClosed` methods, are not acquisition targets.

Table 17-49 lists the methods that are acquisition targets of the Exception trace log. The table also provides the trace acquisition levels of the methods.

Table 17-49: Methods that are acquisition targets of the Exception trace log and their trace acquisition levels

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
`Connection`	`void close()`	Y	Y	Y	Y	Y
	`void commit()`	--	Y	Y	Y	Y
	`Statement createStatement()`#2	Y	Y	Y	Y	Y
	`Statement createStatement(int resultSetType, int resultSetConcurrency)`#3	Y	Y	Y	Y	Y
	`DatabaseMetaData getMetaData()`	--	Y	Y	Y	Y
	`PreparedStatement prepareStatement(String sql)`#2	Y	Y	Y	Y	Y
	`PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)`#3	Y	Y	Y	Y	Y
	`void rollback()`#2	--	Y	Y	Y	Y
	`void setAutoCommit(boolean autoCommit)`	--	Y	Y	Y	Y
`DatabaseMetaData`	`ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)`	--	Y	Y	Y	Y
	`ResultSet getCatalogs()`	--	Y	Y	Y	Y
	`ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)`	--	Y	Y	Y	Y
	`ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)`	--	Y	Y	Y	Y
	`Connection getConnection()`	--	Y	Y	Y	Y
	`ResultSet getCrossReference(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)`	--	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
	ResultSet getExportedKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getImportedKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	--	Y	Y	Y	Y
	ResultSet getPrimaryKeys(String catalog, String schema, String table)	--	Y	Y	Y	Y
	ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	--	Y	Y	Y	Y
	ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	--	Y	Y	Y	Y
	ResultSet getSchemas()	--	Y	Y	Y	Y
	ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	--	Y	Y	Y	Y
	ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	--	Y	Y	Y	Y
	ResultSet getTableTypes()	--	Y	Y	Y	Y
	ResultSet getTypeInfo()	--	Y	Y	Y	Y
	ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	--	Y	Y	Y	Y
	ResultSet getVersionColumns(String catalog, String schema, String table)	--	Y	Y	Y	Y
Driver	Connection connect(String url, Properties info)	Y	Y	Y	Y	Y
PreparedStatement	boolean execute() #2	--	Y	Y	Y	Y
	ResultSet executeQuery() #2	--	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
	`int executeUpdate()` #2	--	Y	Y	Y	Y
	`ResultSetMetaData getMetaData()`	--	Y	Y	Y	Y
	`boolean execute(String sql)` #3, #5	Y	--	Y	Y	Y
	`int[] executeBatch()` #5	--	Y	Y	Y	Y
	`ResultSet executeQuery(String sql)` #3, #5	Y	Y	Y	Y	Y
	`int executeUpdate(String sql)` #3, #5	Y	Y	Y	Y	Y
`ResultSet`	`boolean absolute(int row)`	--	Y	Y	Y	Y
	`void afterLast()`	--	Y	Y	Y	Y
	`void beforeFirst()`	--	Y	Y	Y	Y
	`void close()`	--	Y	Y	Y	Y
	`boolean first()`	--	Y	Y	Y	Y
	`ResultSetMetaData getMetaData()`	--	Y	Y	Y	Y
	`Statement getStatement()`	--	Y	Y	Y	Y
	`boolean last()`	--	Y	Y	Y	Y
	`boolean next()`	--	Y	Y	Y	Y
	`boolean relative(int rows)`	--	Y	Y	Y	Y
	`boolean isAfterLast()`	--	Y	Y	Y	Y
	`boolean isBeforeFirst()`	--	Y	Y	Y	Y
	`boolean isLast()`	--	Y	Y	Y	Y
`Statement`	`void cancel()`	--	Y	Y	Y	Y
	`void close()`	Y	Y	Y	Y	Y
	`boolean execute(String sql)`	Y	Y	Y	Y	Y
	`int[] executeBatch()`	--	Y	Y	Y	Y
	`ResultSet executeQuery(String sql)`	Y	Y	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
	int executeUpdate(String sql)	Y	Y	Y	Y	Y
	ResultSet getResultSet()	--	Y	Y	Y	Y
Blob	long position(Blob pattern,long start)#2	--	Y	Y	Y	Y
	long position(byte[] pattern, long start)#3	--	Y	Y	Y	Y
	long length()	--	Y	Y	Y	Y
	byte[] getBytes(long pos, int length)	--	Y	Y	Y	Y
InputStream	int read()#2	--	Y	Y	Y	Y
	int read(byte[] data, int data_offset,int data_len)#3	--	Y	Y	Y	Y
	int read(byte[] data, int data_offset,int data_len)#4	--	Y	Y	Y	Y
DataSource	getConnection()#2	Y	Y	Y	Y	Y
	getConnection(String username, String password)#3	Y	Y	Y	Y	Y
ConnectionPoolDataSource	getPooledConnection()#2	Y	Y	Y	Y	Y
	getPooledConnection(String username, String password)#3	Y	Y	Y	Y	Y
PooledConnection	close()	Y	Y	Y	Y	Y
	getConnection()	Y	Y	Y	Y	Y
XADataSource	getXAConnection()#2	Y	Y	Y	Y	Y
	getXAConnection(String username, String password)#3	Y	Y	Y	Y	Y
XAConnection	getXAResource()	Y	Y	Y	Y	Y
XAResource	commit(Xid xid, boolean onePhase)	--	--	Y	Y	Y
	end(Xid xid, int flags)	--	--	Y	Y	Y
	forget(Xid xid)	--	--	Y	Y	Y

Class	Method	Trace acquisition level				
		1	2	3	4	5#1
	`isSameRM(XAResource xares)`	--	--	Y	Y	Y
	`prepare(Xid xid)`	--	--	Y	Y	Y
	`recover(int flag)`	--	--	Y	Y	Y
	`rollback(Xid xid)`	--	--	Y	Y	Y
	`start(Xid xid, int flags)`	--	--	Y	Y	Y

Legend:

Y: An Exception trace log is acquired.

--: An Exception trace log is not acquired.

#1

If the trace acquisition level is `5`, an Exception trace log that includes internal calling is acquired.

#2

method-name`(1)` is output as the method name.

#3

method-name`(2)` is output as the method name.

#4

method-name`(3)` is output as the method name.

#5

This method overrides the method of the `Statement` class.

(2) Setup for acquisition of the Exception trace log

Use the system properties or the client environment definitions to set the file output destination of the Exception trace log, the number of outputs to the file, the number of information items to be acquired in memory, and the trace acquisition level. The priorities are as follows:

1. System properties
2. Client environment definitions

(a) Setting client environment definitions

Specify the following items in the client environment definitions:

- `PDJDBFILEDIR`
- `PDJDBFILEOUTNUM`
- `PDJDBONMEMNUM`
- `PDJDBTRACELEVEL`

For details about the specification values, see *6.6.4 Environment definition information*.

If invalid values are specified in these client environment definitions, the facility for controlling the Exception trace log acquired when an `SQLException` is thrown assumes that values were not specified for these client environment definitions. In this situation, the defaults shown in Table 17-50 are assumed.

(b) Setting system properties

In the system properties, specify the items shown in Table 17-50.

Table 17-50: System property settings for acquisition of the Exception trace log

Item	System property	Description	Default#
File output destination	`HiRDB_for_Java_FileDIR`	Specify the absolute path of the directory to which the Exception trace log is to be output. The Exception trace log is output immediately under the specified directory.	Current directory
Number of outputs to the file	`HiRDB_for_Java_FileOutNUM`	Specify the maximum number of information items to be output to one file. Specify a value in the range from 1 to 50. The maximum number of information items to be output to one file is actually *number of outputs to the file* x *number of acquisition items to be acquired in memory*. For the number of outputs to the file, the formats from Format 2 to Format 4 shown in *17.13.2 Output formats* are each counted as one output. The information items are output to memory in the sequence they were stored. If information items exceeding the maximum value are to be output to a file, the items are wrapped around into two files. The file names are as follows: • `pdexc1.trc` • `pdexc2.trc` However, the output destination file does not change between Format 1 and Format 2 shown in *17.13.2 Output formats*.	5

Item	System property	Description	Default#
Number of information items to be acquired in memory	`HiRDB_for_Java_OnMemNUM`	Specify the maximum number of information items to be stored in memory. You can specify a value in the range from 500 to 10,000. For the information acquired in memory, each method shown in Table 17-49 is counted as one item. If the number of information items to be stored exceeds the maximum value, old information items are overwritten with new information items in chronological sequence.	1,000
Trace acquisition level	`HiRDB_for_Java_TraceLevel`	Specify a trace acquisition level. You can specify a level in the range from 0 to 5. If you specify `5`, all methods that are trace acquisition targets, including internally called methods, are acquired. If you specify `0`, an Exception trace log is not acquired.	1

#

When the Exception trace log is acquired in the following cases, the JDBC driver assumes that values were not specified in the system properties (and the defaults are assumed):

- When an invalid value is specified in the system properties and an `SQLException` is thrown during connection to the database
- When the Java Virtual Machine denies the JDBC driver permission to exchange system properties because of security manager reasons
- Before initial connection of the Java Virtual Machine is established

17.13.2 Output formats

The Exception trace log has the following four formats.

Format 1: Header section

```
[AA....AA] HiRDB_Type4_JDBC_Driver BB-CC
```

Format 2: Method execution history (execution start of a method)

```
AAAAAAAAAAAAAAAAAAAAAA  BB....BB:[C][DD....DD]
                               ConnectionID(EE....EE) : SID(FF....FF)
                               GG....GG
```

Format 3: Method execution history (normal termination of a method)

```
AAAAAAAAAAAAAAAAAAAAAAA  BB....BB:[C][DD....DD]
                         ConnectionID(EE....EE) : SID(FF....FF)
                         HH....HH
```

Format 4: Timing when output occurred

```
AAAAAAAAAAAAAAAAAAAAAAA  BB....BB:Exception:
II....II
```

Format 2 and Format 3 are output repeatedly in time series sequence for each method executed.

(1) Explanation of variables in Format 1

AA....AA

Indicates the sequence number of the output information.

The sequence number is incremented by 1 for each output (including failures caused by output errors). After the value reaches 2,147,483,647, the sequence returns to 0.

BB

Indicates the version of the JDBC driver.

CC

Indicates the revision of the JDBC driver.

(2) Explanation of variables in Format 2, Format 3, and Format 4

AAAAAAAAAAAAAAAAAAAAAAA

Indicates the acquisition date and time of the Exception trace log, in the following format (a value from 0 to 9 is set in each variable):

```
YYYY/MM/DD  hh:mm:ss.sss
```

YYYY: Year (Western calendar)

MM: Month

DD: Day

hh: Hour (24-hour clock format)

mm: Minute

ss.sss: Second (includes 3 digits after the decimal point)

BB....BB

Indicates thread identification information for the target thread, in the following format:

```
Thread[aa....aa]@bb....bb
```

aa....aa: Thread information, including the thread name, priority sequence, and thread group name. The Java Virtual Machine determines the format.

bb....bb: Hash code of the object. The Java Virtual Machine determines the format.

C

Indicates call identification information for the method:

`E`: Indicates that the information is history information for when the method was started.

`R`: Indicates that the information is history information for when the method terminated normally.

DD....DD

Indicates the object identifier and the method name, in the following format:

```
aa....aa.bb....bb
```

aa....aa: Object identifier (up to 32 characters)

The Java Virtual Machine determines the format.

bb....bb: Method name

EE....EE

Indicates the connection ID, in the following format:

```
aa....aa:bb....bb:cc....cc
```

aa....aa: Front-end server name or single-server name (up to 32 characters).

If this information cannot be retrieved, an asterisk (`*`) is output.

bb....bb: Connection sequence number (up to 10 characters) of the server identified by *aa....aa*.

If this information cannot be retrieved, an asterisk (`*`) is output.

cc....cc: Process ID (up to 10 characters) of the server identified by *aa....aa*.

If this information cannot be retrieved, an asterisk (`*`) is output.

FF....FF

Indicates the section ID (up to 4 characters).

GG....GG

Indicates the method arguments, in the following format (this information is not output for methods without arguments):

```
aa....aa=bb....bb
aa....aa=bb....bb
        :
aa....aa=bb....bb
```

aa....aa: Argument name.

bb....bb: Argument contents (up to 256 characters).

For reference type values, the object determines the format.

One asterisk (`*`) is output to *bb....bb* for the `password` argument of the following methods:

- `getConnection(String username, String password)` of the `DataSource` class
- `getPooledConnection(String username, String password)` of the `ConnectionPoolDataSource`
- `getXAConnection(String username, String password)` of `XADataSource`

For the `info` argument in `connect(String url,Properties info)` of the `Driver` class, the value of the each of the following properties is replaced by one asterisk (`*`) and then output:

- `password`
- `HiRDB_for_Java_ENV_VARIABLES`

HH....HH

Indicates the return value of the method, in the following format:

```
Return=aa....aa
```

aa....aa: Argument name.

This item is not output for methods that do not have a return value. If the return value is a reference-type value, the Java Virtual Machine determines the format.

II....II

Indicates troubleshooting information, in the following format:

```
ExceptionClass: aa....aa
UapEnvironment: bb....bb
Message: cc....cc
ErrorCode: dd....dd
SQLState: eeeee
UpdateCounts: ff....ff, ....<omitted>,ff....ff
Etc.: gg....gg, hh....hh, iiii
jj....jj
```

aa....aa: Effective class name of the exception object that was thrown.

bb....bb: Client environment definitions being used in the connection of the exception object. The definitions are output in the following format (if no definitions are to be output, this variable is replaced by an asterisk (`*`) and then output):

```
yy....yy (zz....zz), ...<omitted>, yy....yy (zz....zz)
```

yy....yy: Name of the client environment definition without the initial `PD` characters. The following client environment definitions are the output targets:

- `PDHOST`
- `PDNAMEPORT`
- `PDFESHOST`
- `PDSERVICEGRP`
- `PDSRVTYPE`

- PDSERVICEPORT
- PDCLTRCVPORT
- PDCLTRCVADDR
- PDUSER
- PDCWAITTIME
- PDSWAITTIME
- PDSWATCHTIME

zz....zz: Contents of the client environment definition. The password portion of PDUSER is not output.

cc....cc: Message of the exception object.

dd....dd: SQLCODE error code (for XAException, error code indicated by the errorCode field of the XAException object) (up to 11 characters).

This item is output when the effective class of the thrown exception object is one of the following classes or subclasses:

- SQLException
- XAException

eeeee: SQLSTATE (5 characters).

This item is output when the effective class of the thrown exception object is SQLException or a subclasss of SQLException.

ff....ff: Number of update rows for each update statement in a batch update that was executed normally before this exception occurred (up to 11 characters).

This item is output when the effective class of the exception object is BatchUpdateException.

If the number of update rows cannot be obtained, an asterisk (*) is output.

gg....gg: SQL counter value (up to 11 characters).

This information can be used for coordinating with the trace information output by the SQL trace facility.

If the SQL counter cannot be obtained, an asterisk (*) is output.

hh....hh: Failure information of the HiRDB server when an error occurs in the HiRDB server (up to 22 characters).

The failure information is used by maintenance personnel.

If no errors have occurred in the HiRDB server, an asterisk (*) is output.

iiii: Type of request (operation code) that the JDBC driver issued to the HiRDB server when an error occurred in the HiRDB server.

If no errors have occurred in the HiRDB server, an asterisk (*) is output.

jj....jj: Stack trace in which the exception-throwing method is set as the base point.

The Java Virtual Machine determines the format.

17.13.3 Output example and analysis method

(1) Output example

An output example of the Exception trace log is shown below:

```
[1] HiRDB_Type4_JDBC_Driver 08-00
2006/07/06 23:07:09.129
Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.createStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:09.160
Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.createStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement@1e4cbc4
2006/07/06 23:07:09.160
Thread[main,5,main]@1259414:[E][PrdbStatement@1e4cbc4.execute]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=DELETE FROM SEINO_TABLE
2006/07/06 23:07:14.285 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:14.301
Thread[main,5,main]@1259414:[R][PrdbStatement@1e4cbc4.execute]
                        ConnectionID(sds:23:20484) : SID(1)
                        Return=false
2006/07/06 23:07:14.301
Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=INSERT INTO SEINO_TABLE VALUES(?, ?)
2006/07/06 23:07:14.348
Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                   Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@15d56d5
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
```

```
2006/07/06 23:07:26.567 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:26.567
Thread[main,5,main]@1259414:[E][PrdbStatement@1e4cbc4.executeQuery]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:26.676
Thread[main,5,main]@1259414:[R][PrdbStatement@1e4cbc4.executeQuery]
                        ConnectionID(sds:23:20484) : SID(1)
                        Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbResultSet@3eca90
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(1)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][PrdbResultSet@3eca90.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:28.332
Thread[Thread-0,5,main]@30090737:[E][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:28.332
Thread[Thread-0,5,main]@30090737:[R][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                    Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@2808b3

2006/07/06 23:07:28.348
Thread[Thread-1,5,main]@5462872:[E][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                        sql=DELETE FROM SEINO_TABLE WHERE I1=?
2006/07/06 23:07:28.358
Thread[Thread-1,5,main]@5462872:[E][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:29.672
Thread[Thread-1,5,main]@5462872:[R][PrdbConnection@82c01f.commit]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:30.098
Thread[Thread-1,5,main]@5462872:[R][PrdbConnection@82c01f.prepareStatement(1)]
                        ConnectionID(sds:23:20484) : SID(0)
                    Return=JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement@922804
2006/07/06 23:07:30.332
Thread[Thread-2,5,main]@25253977:[E][PrdbConnection@82c01f.rollback(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[R][PrdbConnection@82c01f.rollback(1)]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[E][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
2006/07/06 23:07:42.098
Thread[Thread-2,5,main]@25253977:[R][PrdbConnection@82c01f.close]
                        ConnectionID(sds:23:20484) : SID(0)
```

```
2006/07/06 23:07:42.535 Thread[Thread-1,5,main]@5462872:Exception:
ExceptionClass: SQLException
UapEnvironment: *
Message: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
ErrorCode: -1020006
SQLState: R2400
Etc.: *,*,****
java.sql.SQLException: KFPJ20006-E Connection closed[PrdbPreparedStatement.setInt]
at
JP.co.Hitachi.soft.HiRDB.JDBC.JdbMakeException.generateSQLException(JdbMakeException
.java:31)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement.generateClosedSQLException(PrdbStatement
.java:3005)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement.setInt(PrdbPreparedStatement.jav
a:1170)
at Exception1.run(ExceptionTraceSample.java:57)
[2] HiRDB_Type4_JDBC_Driver 08-00
2006/07/06 23:07:25.723
Thread[Thread-3,5,main]@13249998:[E][PrdbConnection@119cca4.prepareStatement(1)]
                        ConnectionID(sds:24:20484) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2006/07/06 23:07:25.770
Thread[Thread-4,5,main]@25839584:[E][PrdbConnection@119cca4.rollback(1)]
                        ConnectionID(sds:24:20484) : SID(0)
2006/07/06 23:07:25.770
Thread[Thread-4,5,main]@25839584:[R][PrdbConnection@119cca4.rollback(1)]
                        ConnectionID(sds:24:20484) : SID(0)
2006/07/06 23:07:25.770
Thread[Thread-5,5,main]@24431647:[E][PrdbConnection@119cca4.prepareStatement(1)]
                        ConnectionID(sds:24:20484) : SID(0)
                        sql=SELECT ** FROM SEINO_TABLE
2006/07/06 23:07:25.863 Thread[Thread-5,5,main]@24431647:Exception:
ExceptionClass: SQLException
UapEnvironment: USER(USER1), NAMEPORT(20249), CWAITTIME(0), SWAITTIME(600),
HOST(dragon2), FESHOST( ),
SERVICEGRP(sds), SWATCHTIME( ), SERVICEPORT( ), SRVTYPE(WS), CLTRCVPORT( ),
CLTRCVADDR( ), FESGRP( )
Message: KFPA11105-E Invalid token "*" after token "*"[PrdbStatement.prepare]
ErrorCode: -105
SQLState: R0000
Etc.: 4,sqapyac1.c(651),SET
java.sql.SQLException: KFPA11105-E Invalid token "*" after token
"*"[PrdbStatement.prepare]
at JP.co.Hitachi.soft.HiRDB.JDBC.CltSection.prepare(CltSection.java:1497)
at JP.co.Hitachi.soft.HiRDB.JDBC.PrdbStatement.prepare(PrdbStatement.java:2834)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbPreparedStatement.<init>(PrdbPreparedStatement.jav
a:109)
at
JP.co.Hitachi.soft.HiRDB.JDBC.PrdbConnection.prepareStatement(PrdbConnection.java:10
41)
at Exception1.run(ExceptionTraceSample.java:64)
```

(2) Analysis method

This item explains the analysis method of the Exception trace log. You can use a text editor to reference the Exception trace log.

Described below is an example of analyzing the Exception trace log shown in *(1) Output example*.

Analysis example

To analyze the Exception trace log:

1. Extract the sequentially numbered information, including the exception to be investigated.
2. Categorize the information by using the thread identification information, and separate the information by thread.
3. Arrange the information in time sequence based on the acquisition time.

   Table 17-51 shows what the results look like.

   Table 17-51: Example in which the Exception trace log is arranged in time sequence

Date and time	Thread 1	Thread 2	Thread 3	Thread 4
	Thread[main,5,main] @1259414	Thread[Thread-0,5,main] @30090737	Thread[Thread-1,5,main] @5462872	Thread[Thread-2,5,main] @25253977
2006/07/06 23:07:09.129	PrdbConnection @82c01f.createStatement(1)			
2006/07/06 23:07:09.160	PrdbStatement @1e4cbc4.execute			
2006/07/06 23:07:14.285	PrdbConnection @82c01f.commit			
2006/07/06 23:07:14.301	PrdbConnection @82c01f.prepareStatement(1)			
2006/07/06 23:07:26.567	PrdbConnection @82c01f.commit			
2006/07/06 23:07:26.567	PrdbStatement @1e4cbc4.executeQuery			

Date and time	Thread 1	Thread 2	Thread 3	Thread 4
	Thread[main,5,main] @1259414	Thread[Thread-0,5,main] @30090737	Thread[Thread-1,5,main] @5462872	Thread[Thread-2,5,main] @25253977
2006/07/06 23:07:26.567	PrdbStatement @1e4cbc4.execute			
2006/07/06 23:07:28.332	PrdbResultSet @3eca90.close	PrdbConnection @82c01f.prepareStatement(1)		
2006/07/06 23:07:28.332	PrdbConnection @82c01f.commit			
2006/07/06 23:07:28.348			PrdbConnection @82c01f.prepareStatement(1)	
2006/07/06 23:07:28.358			PrdbConnection @82c01f.commit	
2006/07/06 23:07:30.332				PrdbConnection @82c01f.rollback
2006/07/06 23:07:42.098				PrdbConnection @82c01f.close
2006/07/06 23:07:42.535			SQLException occurred KFPJ20006-E Connection closed	

4. Check the contents of the exception error.

   The information indicates that an `SQLException` occurred in Thread 3 at 2006/07/06 23:07:42.535, and that a `Statement` or `Connection` object had already been closed.

5. Check the operation of the object in the time sequence.

   Because the object ID of the `Connection` object in the next thread is the same, we know that the four threads were being processed in the same connection.

   - Thread 1 at 2006/07/06 23:07:09.129
   - Thread 2 at 2006/07/06 23:07:28.332

- Thread 3 at 2006/07/06 23:07:28.348
- Thread 4 at 2006/07/06 23:07:30.332

6. Search for the location of the error cause.

   Because we know that the four threads have the same connection, we can search for the locations where the `Statement.close` or `Connection.close` method was executed, and learn that Thread 4 executed the `Connection.close` method at 2006/07/06 23:07:42.098. From this, we know that the cause of the SQLException that occurred in Thread 3 at 2006/07/06 23:07:42.535 was that Thread 4 executed the `Connection.close` method at 2006/07/06 23:07:42.098.

17.13.4 Required memory size and file size

(1) Required memory size

The memory size required for acquiring the Exception trace log is determined from the following formula:

Formula

$\uparrow 360 \text{ x } n \uparrow /1024$ (kilobytes)

Explanation

n: Number of information items to be acquired in memory

(2) Required file size

The file size for acquiring the Exception trace log is determined from the following formula:

Formula

$\uparrow 180 \text{ x } n \text{ x } m \uparrow /1024 + 1$ (kilobytes)

Explanation

n: Number of information items to be acquired in memory

m: File output information

17.13.5 Notes

(1) If the system properties and client environment definition settings are different

Table 17-52 shows how the method execution history that was accumulated in the JDBC driver memory before establishment of the first HiRDB connection is transferred if the system properties and client environment definition settings are different.

Table 17-52: Transfer of the method execution history accumulated in the JDBC driver memory

Item	Relationship between system properties and client environment definition	Transfer operation
Number of information items to be acquired in memory	`HiRDB_for_Java_OnMemNUM` < `PDJDBONMEMNUM`	The JDBC driver re-allocates memory for accumulation of the method execution history based on the `PDJDBONMEMNUM` specification, and then copies the execution history accumulated up to that point to the re-allocated area. However, if the `PDJDBTRACELEVEL` specification is `0`, memory is not re-allocated.
	`HiRDB_for_Java_OnMemNUM` > `PDJDBONMEMNUM`	The JDBC driver re-allocates memory for accumulation of the method execution history based on the `PDJDBONMEMNUM` specification. The driver then destroys any accumulated execution history information that cannot be stored in the re-allocated area, and copies the remaining information to the re-allocated area. However, if the `PDJDBTRACELEVEL` specification is `0`, memory is not re-allocated.

Item	Relationship between system properties and client environment definition	Transfer operation
Trace acquisition level	HiRDB_for_Java_TraceLevel < PDJDBTRACELEVEL	The driver simply transfers the execution history that was accumulated up to that point.
	HiRDB_for_Java_TraceLevel > PDJDBTRACELEVEL	If the PDJDBTRACELEVEL specification is 1 or greater, the JDBC driver simply transfers the execution history accumulated up to that point. The driver also transfers the execution history of methods that are not targeted by the trace acquisition level specified by PDJDBTRACELEVEL. If the PDJDBTRACELEVEL specification is 0, the JDBC driver destroys the accumulated execution history for each accumulation memory.

(2) First output after startup of the Java Virtual Machine

The first time the Exception trace log is output to a file after the Java Virtual Machine is started, the log is output to the file with the older update date and time. If the date and time are the same for both files, the log is output to `pdexc1.trc`.

(3) Specification of the file output destination

If the same file output destination is specified when Exception trace logs are being acquired from multiple processes, trace information for the different processes is output to the same file. To acquire a trace for each process, specify a different file output destination for each process.

The JDBC driver uses the facilities of the Java Virtual Machine to create log files in the file system provided by the OS. Therefore, the following items depend on the Java Virtual Machine and file system being used:

- Prefix for the absolute path name
- Path delimiter character
- Maximum number of characters for the output destination file (absolute path)
- Size per file

(4) Processing when an error occurs

Information is not output to the Exception trace log when file creation or output fails. An error message may be returned to the UAP and file output may be retried.

(5) Character encoding

The Exception trace log is output with the default conversion character set of the Java Virtual Machine being used.

Chapter

18. SQLJ

This chapter explains how to use SQLJ to develop a UAP. Note that SQLJ cannot be used in the Linux for AP8000 version of a client.

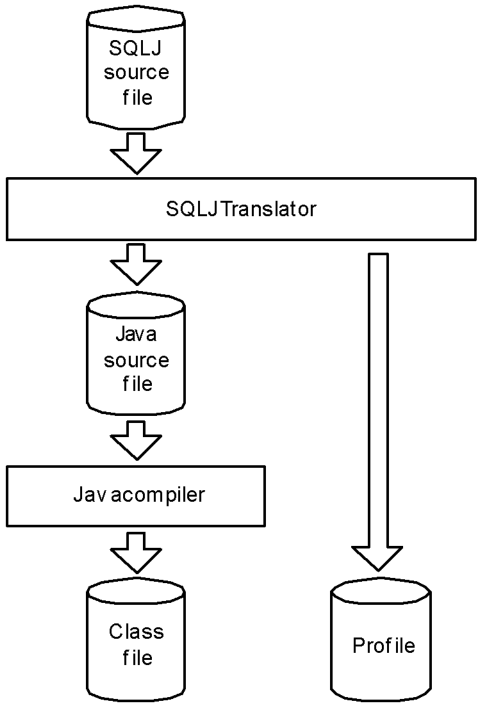
18.1 Overview

18.1.1 What is SQLJ?

SQLJ is a language specification for coding a static SQL statement as an embedded SQL statement in Java and executing it.

Figure 18-1 shows the flow of UAP development that uses SQLJ.

Figure 18-1: Flow of UAP development that uses SQLJ



SQLJ consists of SQLJ Translator and SQLJ Runtime Library.

SQLJ Translator

SQLJ Translator analyzes an SQLJ source program and replaces SQL statements with standard Java instructions for accessing a database through SQLJ Runtime Library.

SQLJ Translator generates a Java source file and a profile that stores SQL information. The user uses the Java compiler to compile the Java source file to create a `class` file (executable file).

SQLJ Runtime Library

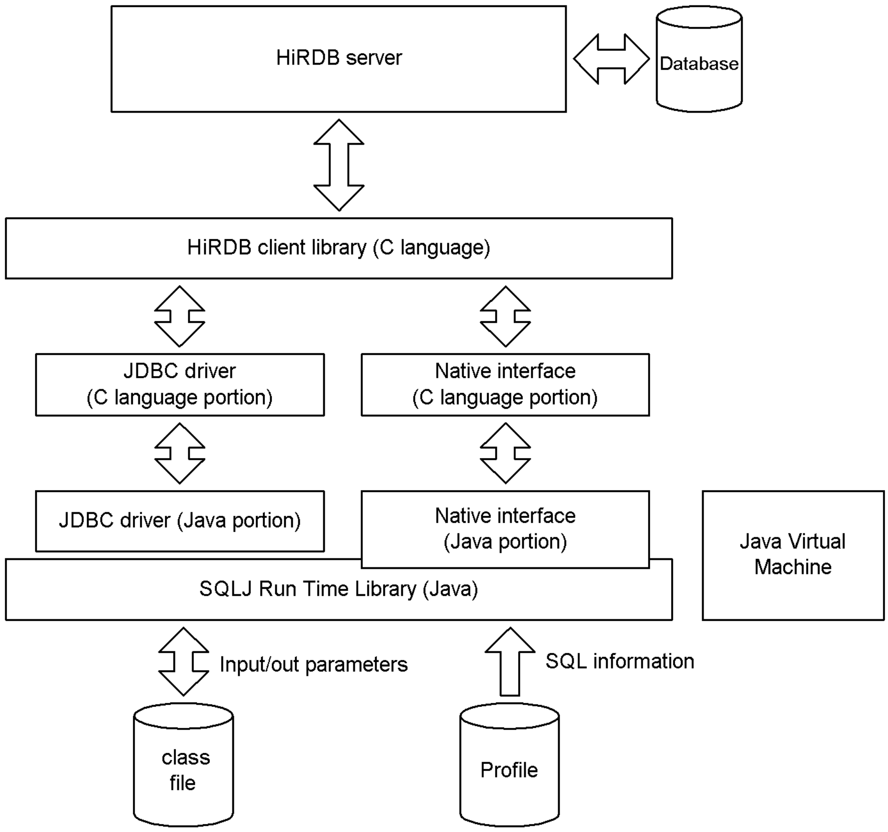
SQLJ Runtime Library is used for executing a compiled `class` file.

SQLJ Runtime Library can be used in either of the following ways, depending on the access interface used:

- Invoke the JDBC interface API (standard interface version) and execute the SQL statements.
- Invoke an original interface (native interface version), not the JDBC interface, and execute the SQL statements.

Figure 18-2 shows UAP execution using SQLJ.

Figure 18-2: Execution of a UAP that uses SQLJ



Explanation:

- A `class` file of the SQLJ source file compiled by the Java compiler accesses a database through the SQLJ Runtime Library.

- When the native interface is used, SQLJ Runtime Library directly invokes a HiRDB client library instead of invoking JDBC. In this case, you cannot use coding that directly invokes the JDBC API and shares connection and result sets with JDBC.
- Because the SQLJ Runtime Library loads a profile during execution, the `class` file and profile must be stored in the same directory. Also, when the `class` file is stored in the `jar` file, you must also store the profile in the `jar` file.

18.1.2 Environment settings

The environment variable settings required for SQLJ operation are shown below. Since SQLJ uses the JDBC driver, environment settings for the JDBC driver must also be specified.

(1) Environment settings for the UNIX version

Set the following information in the environment variables for the execution environment.

(a) HiRDB/Developer's Kit

`CLASSPATH=$CLASSPATH:[`*installation-directory*`]/pdsqlj.jar`*

* For the 32-bit mode HP-UX (IPF) version, specify `pdsqlj32.jar`.

(b) HiRDB/Run Time

`CLASSPATH=$CLASSPATH:[`*installation-directory*`]/pdruntime.jar`[1]
`CLASSPATH=$CLASSPATH:[`*installation-directory*`]/pdnativert.jar`[2]

Note

When using the 32-bit mode HP-UX (IPF) version, do not specify the following pairs of files at the same time:

- `pdsqlj.jar` and `pdsqlj32.jar`
- `pdruntime.jar` and `pdruntime32.jar`
- `pdnativert.jar` and `pdnativert32.jar`

1 For the 32-bit mode HP-UX (IPF) version, specify `pdruntime32.jar`.

2 For the 32-bit mode HP-UX (IPF) version, specify `pdnativert32.jar`.

(2) Environment settings in the Windows version

In sequence, choose **Control Panel**, **System**, **System Properties**, and **Environment**, and then set the contents shown below.

(a) HiRDB/Developer's Kit

```
CLASSPATH=%CLASSPATH%;[installation-directory]\pdsqlj.jar
```

(b) HiRDB/Run Time

```
CLASSPATH=%CLASSPATH%;[installation-directory]\pdruntime.jar
CLASSPATH=%CLASSPATH%;[installation-directory]\pdnativert.jar
```

18.2 SQLJ Translator

SQLJ Translator analyzes an SQLJ source program and generates a Java source file and a profile.

SQL statements are replaced with Java instructions, including the invocation of the JDBC API, and are output as a Java source file.

SQL character strings, number of parameters, types and modes of individual parameters, and the description of the columns to be output to a profile. The profile is referenced from the SQLJ Runtime Library. The entity of a profile is an instance of the `java.sql.runtime.Profile` class.

Table 18-1 lists the files that are generated and referenced by the SQLJ Translator.

Table 18-1: Files that are generated and referenced by the SQLJ Translator

File classification	File name format	Explanation	Type
SQLJ source file	*file-name*`.sqlj`	Indicates an SQLJ source file.	Referenced
Java source file	*file-name*`.java`	Indicates a Java source file.	Generated
Profile	*file-name*`_SJProfile` *profile-number*`.ser`	Stores the information of each SQL statement extracted from an SQLJ source file. A profile number is assigned to each context. The cardinal number is `0` for all.	Generated

The prefixes of the classes and variables internally generated by SQLJ Translator are as follows:

`_sJT_`: Names of the variables internally generated

`_SJ`: Names of the classes and profiles internally generated

18.3 UAP coding rule

This section explains the coding rule for SQLJ source files.

18.3.1 Labeling rule

The following labels cannot be used:

- Label that begins with `_sJT_`
- Label that begins with `_SJf`
- Label that begins with `p_rdb`

Other rules are based on the Java language rules.

18.3.2 SQL coding rule

(1) SQL statement coding rule

Each SQL statement must be enclosed between the SQL leading character string (`#sql`) and the SQL trailing character (`;`). The SQL statement itself must further be enclosed between curly brackets. Connection class and cursor declarations must also be enclosed between the SQL leading character string and the SQL trailing character.

Table 18-2 shows the SQL statement coding formats.

Table 18-2: SQL statement coding formats

Function	Format	Purpose
SQL execution	`#sql [`*context*`] {` *SQL-statement* `} ;`	Executes an SQL statement. The SQL statements that can be used differ for the standard interface version and the native interface version. For details, see *18.3.3 SQL statements that can be used in SQLJ*.
Declaration of an iterator class with column specification	• Standard interface version `#sql` *modifier* `iterator` *class-name* `(`*data-type column-name*`,...) ;` • Native interface version This function cannot be used.	Declares the class to be used for cursor declaration. Cannot be used in a `FETCH` statement.

Function	Format	Purpose

Declaration of an iterator class with a position specification	• Standard interface version
	`#sql` *modifier* `iterator` *class-name*
	`(`*data-type*`, ...) ;`
	• Native interface version
	`#sql` *modifier* `iterator` *class-name*
	`[implements`
	`JP.co.Hitachi.soft.HiRDB.pdjpp.runti`
	`me.`
	`ForUpdate]`
	`[with (`*keyword*`=`*value*`, ...)]`
	`(`*data-type*`, ...) ;`
		Declares the class to be used in the cursor declaration. This function is used in a `FETCH` statement.

Declaration of a connection class	`#sql` *modifier* `context` *class-name* `;`	Declares the class to be used for connection.
Declaration of a cursor	`#sql` *iterator-object* `= {` *SELECT-statement* `} ;`	Defines and opens a cursor.
Conversion of a result set	• Standard interface version `#sql [`*context*`]` *iterator-object* `= {CAST :`*JDBC-result-set*`} ;` • Native interface version This function cannot be used.	Converts a JDBC result set into one that can be used by SQLJ.

Notes

modifier

Combination of `private`, `public`, `protected`, `final`, `abstract`, `protected`, `static`, `native`, `synchronized`, `transient`, and `volatile`.

context

`{`*connection-context* | *connection-context*,*execution-context* | *execution-context*`}`

keyword

`holdability` or `updateColumns`

value

`true`, `false`, or `"`*column-name-1*`,`*column-name-2*`,...`"`

data-type

Java data type

column-name

Retrieval item

(2) Explicitly specifying connection context when using the multi-connection facility

When you use the multi-connection facility, insert the connection context surrounded by square brackets between the SQL leading character string and the SQL statement, to explicitly specify the connection to be used. An example follows:

```
#sql [connCtx] { DELETE FROM EMP WHERE SAL > 1000};
```

If no connection context is explicitly specified, the default connection context is assumed.

(3) Explicitly specifying an execution environment

In SQLJ, a user can explicitly specify an execution environment instead of using the default one. To specify an execution environment, insert the execution connection context surrounded by square brackets between the SQL leading character string and the SQL statement.

If SQL statements are simultaneously being executed in multiple threads for a single connection, using separate multiple execution environments can prevent an execution result from being overwritten by another SQL statement. An example follows:

```
ExecutionContext execCtx = new ExecutionContext();
try {
 #sql [execCtx] { DELETE FROM STOCK WHERE PCODE > 1000 };
 System.out.println
  ("removed " + execCtx.getUpdateCount() + "goods");
}
catch(SQLException e){
 System.out.println("SQLException has occurred with "+ " exception " + e);
}
```

If no execution connection context is explicitly specified, the default execution environment is used.

The values described in the following table are maintained in the execution environments. These values are set using the `set`*<name>* method and determined using the `get`*<name>* method.

Name	Details
`MaxRows`	Maximum number of rows to be returned from a search.
`MaxFieldSize`	Maximum size of data in units of bytes to be returned in columns and `OUTPUT` variable value.
`QueryTimeout`	Maximum wait time until SQL execution is completed. This is invalid in HiRDB.
`UpdateCount`	Number of updated, inserted, or deleted rows (reference only).

Name	Details
`SQLWarnings`	Correspond to `SQLWARN0`-`SQLWARNF` (reference only).

If multi-connection is also specified, the connection context and execution connection context must be specified in that order, delimited by a comma. An example follows:

```
#sql [connCtX, execCtx] { DELETE FROM STOCK WHERE PCODE > 1000 };
```

(4) Specifying embedded variables

In SQLJ, `BEGIN DECLARE SECTION` for declaring embedded variables is not used.

Any variables, parameters, and object fields can be used as embedded variables. In an SQL statement, a variable is described as "`:`*variable-name*" with a colon at the front. The colon can be separated by blank spaces from the variable name.

The `IN`, `OUT`, or `INOUT` parameter of a `CALL` statement is described as "`:{IN|OUT|INOUT}`*variable-name*".

Additionally, in SQLJ, you can use "`:(`*expression*`)`" as an embedded variable. The expression must be enclosed by parentheses. This is a Java method and not an SQL method. An example follows:

```
#sql { SELECT COL1, COL2 FROM TABLE1 WHERE :(x[--i]) > COL3 };
```

(5) Specifying indicator variables

SQLJ has no indicator variable. Therefore, to set a null value for an embedded variable, use the `Wrapper` type defined in the `sql.lang` package instead of the basic data type. If a null value is received by a Java variable of the basic data type, the `SQLNullException` exception occurs.

(6) Exception handling

SQLJ cannot handle exceptions from an embedded SQL `WHENEVER` statement. Therefore, Java exception handling (`try...catch`) is used instead of `WHENEVER`. An example follows:

```
try{
   #sql { DELETE FROM STOCK WHERE PCODE > 1000 };
}
catch(SQLException e){
   System.out.println("SQLCODE:" + e.getErrorCode() +
                      "\nERRMSG:" + e.getMessage() );
}
```

If an error occurs during SQL execution, the JDBC exception object (`java.sql.SQLException`) is issued.

SQLCODE, SQLSTATE, and error messages are stored in exception objects, and their values can be obtained using the getErrorCode, getSQLState, and getMessage methods.

(7) Static SQL statements and dynamic SQL statements

In SQLJ, only static SQL statements can be described. Dynamic SQL statements cannot be described.

To use a dynamic SQL statement, use the JDBC API.

(8) Reading out the result set of a dynamic cursor

You can use a CAST statement to convert and read out the result set of a dynamic cursor created using the JDBC API as the result set of an SQLJ cursor. An example follows:

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String Query="SELECT pname, pcode FROM stock WHERE pcode > 1000";
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps ={CAST :rs };
```

The CAST statement cannot be used with the native interface version. If the statement is used, a translation error results.

(9) Connecting to and disconnecting from a HiRDB server

The CONNECT and DISCONNECT statements can be used in the native interface version but not in the standard interface version. For both the standard interface and native interface versions, Java instructions can be used to connect to or disconnect from a HiRDB server.

(10) Exception generation conditions

In HiRDB embedded SQL statements, an alarm is issued in the following cases. In contrast, exceptions occur in SQLJ.

- In a single-row SELECT statement, the number of search items does not match the number of variables specified in an INTO clause.
- In a single-row SELECT statement, the retrieval result has zero rows.
- In a single-row SELECT statement, the retrieval result has multiple rows.
- In a FETCH statement, the number of search items does not match the number of variables specified in an INTO clause.
- The number of columns defined by an iterator with a position specification does not match the number of retrieval items.
- The number of columns defined by an iterator with a column name specification

is greater than the number of retrieval items.

(11) Comments and handling of SQL optimization specification

Comments (`/*-*/`) described between the SQL leading character string and the SQL trailing character are deleted. However, in cursor declaration and SQL statement execution, the SQL optimization specification (`/*>>-<<*/`) described between curly brackets is not deleted and handled as an SQL statement. All other SQL optimization specifications (`/*>>-<<*/`) are treated as comments. For details on comments and SQL optimization specifications inside SQL statements, see the manual *HiRDB Version 8 SQL Reference*.

18.3.3 SQL statements that can be used in SQLJ

Table 18-3 lists the SQL statements that can be used in SQLJ.

Table 18-3: SQL statements that can be used in SQLJ

Type	SQL statement	Usability		Alternate means
		Standard interface version	Native interface version	
Definition SQL statement	All	Y	Y	None
Data manipulation SQL statements	`ASSIGN LIST` statement	N (SQLJ)	N (SQLJ)	Use JDBC.
	`CALL` statement	Y	Y	None
	`CLOSE` statement	N (SQLJ)	N (SQLJ)	Use an iterator.
	`DECLARE CURSOR`	N (SQLJ)	N (SQLJ)	
	`DELETE` statement	Y	Y	None
	`DESCRIBE` statement	N (JDBC)	N (JDBC)*	Use JDBC.
	`DESCRIBE TYPE` statement	N (JDBC)	N (JDBC)*	
	`DROP LIST` statement	N (JDBC)	N (JDBC)*	
	`EXECUTE` statement	N (JDBC)	N (JDBC)*	
	`EXECUTE IMMEDIATE` statement	N (JDBC)	N (JDBC)*	
	`FETCH` statement (Format 1 or 3)	Y	Y	None
	`FETCH` statement (Format 2)	N	N	None
	`INSERT` statement	Y	Y	None

Type	SQL statement	Usability		Alternate means
		Standard interface version	Native interface version	
	OPEN statement (Format 1)	N (SQLJ)	N (SQLJ)	Use an iterator.
	OPEN statement (Format 2)	N (SQLJ)	N (SQLJ)	Use JDBC.
	PREPARE statement	N (SQLJ)	N (SQLJ)	
	PURGE TABLE statement	Y	Y	None
	Single-row SELECT statement	Y	Y	None
	Dynamic SELECT statement	N (JDBC)	N (JDBC)*	Use JDBC.
	UPDATE statement	Y	Y	None
Control SQL statements	COMMIT statement	Y	Y	None
	COMMIT statement (RELEASE specified)	N (SQLJ)	N (SQLJ)	Split into COMMIT and DISCONNECT.
	CONNECT statement	N	Y	None
	DISCONNECT statement	N	Y	None
	LOCK statement	Y	Y	None
	CONNECT statement with RD-node specification	N	N	None
	DISCONNECT statement with RD-node specification	N	N	None
	ROLLBACK statement	Y	Y	None
	ROLLBACK statement (RELEASE specified)	N (SQLJ)	N (SQLJ)	Split into ROLLBACK and DISCONNECT.
	SET CONNECTION statement	N	N	None
	SET SESSION AUTHORIZATION statement	N	N	None
Embedded language syntax	BEGIN DECLARE SECTION	N	N	None

Type	SQL statement	Usability		Alternate means
		Standard interface version	Native interface version	
	`END DECLARE SECTION`	N	N	None
	`ALLOCATE CONNECTION HANDLE`	N (SQLJ)	N (SQLJ)	Use a connection context.
	`DECLARE CONNECTION HANDLE`	N (SQLJ)	N (SQLJ)	
	`FREE CONNECTION HANDLE`	N (SQLJ)	N (SQLJ)	
	`GET CONNECTION HANDLE`	N	N	None
	`COPY`	N	N	None
	`GET DIAGNOSTICS`	N	N	None
	`WHENEVER`	N (SQLJ)	N (SQLJ)	Implement using `try ... catch`.

Legend:

Y: Can be used in SQLJ.

N (SQLJ): Cannot be used in SQLJ, but a similar function is available in the functions provided by SQLJ or JAVA.

N (JDBC): Cannot be used in SQLJ, but a similar function is available when JDBC is used.

N: Cannot be used in SQLJ.

None: There is no alternate means.

Note

SQLJ cannot use HiRDB functions that are not provided by the JDBC driver. The following functions cannot be used:

- `UPDATE` statement and `DELETE` statement that use an iterator
- Specification of a keyword in a `WITH` clause during the declaration of an iterator
- `INSERT` function that uses an array

* If you use a JDBC connection object to create a connection context, you can also use the alternate means with the native interface. If you do not use a JDBC connection object to create a connection context, you cannot use the alternate means.

18.3.4 Correspondence between HiRDB data types and SQLJ data types

Table 18-4 shows the correspondence between the HiRDB data types and the SQLJ data types. To use embedded variables in SQLJ, declare variables according to this table.

Table 18-4: Correspondence between HiRDB data types and SQLJ data types

HiRDB data types	SQLJ data types (Java data types)	
	When a null value is included	When a null value is not included
CHAR1	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR[4]	N/A
VARCHAR	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR[4]	N/A
NCHAR1	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR[4]	N/A
NVARCHAR[1]	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR[4]	N/A
MCHAR1	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR[4]	N/A
MVARCHAR1	java.lang.String	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR[4]	N/A
DECIMAL2	java.math.BigDecimal	N/A
	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL[4]	N/A
SMALLINT	java.lang.Short	short
INTEGER	java.lang.Integer	int

HiRDB data types	SQLJ data types (Java data types)	
	When a null value is included	When a null value is not included
REAL, SMALLFLT	java.lang.Float	float
FLOAT, DOUBLE PRECISION	java.lang.Double	double
DATE	java.sql.Date	N/A
TIME	java.sql.Time	N/A
TIMESTAMP	java.sql.Timestamp	N/A
INTERVAL HOUR TO SECOND	N/A	N/A
INTERVAL YEAR TO DAY	N/A	N/A
BLOB[3]	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB[4]	byte[]
BINARY	JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY[4]	byte[]

Legend:

N/A: Cannot be used or not applicable

Note

Repetition columns cannot be used.

[1] When `java.lang.String` is specified in the native interface version, the data type requested to the server is `VARCHAR`. When the data type is specified in an output variable, the length of the data acceptance area is assumed to be 32,000 bytes.

[2] When `java.math.BigDecimal` is used as an output variable in the native interface version, the precision is set to `15` and the scale to `0`.

[3] When the data type is specified with `byte[]` in the native interface version, the data type requested to the server is `BINARY` type. If the HiRDB server is version 06-02 or earlier and the `BLOB` type is to be used, specify `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB`, which is a HiRDB data type. An error occurs if `byte[]` is specified.

[4] This type can be specified for the native interface version.

18.3.5 Output variable settings (limited to the native interface version)

When an execution request is sent to the server, the data length set in the SQL descriptor area for an output variable used in single-line searches, and in the OUT parameters of CALL statements when an execution request is sent to the server, differs depending on the initial value of the output variable. Table 18-5 lists the initial value of each data type and the data length set in the SQL descriptor area.

Table 18-5: Initial value for each data type and the data length set in SQL Descriptor Area

Data type	Initial value	Length of data set in SQL Descriptor Area
HiRDBCHAR	*variable* = null;	30,000 bytes
	variable = new HiRDBCHAR(int *n*);	*n* bytes (1 ≤ *n* ≤ 30,000)
	variable = new HiRDBCHAR(String *t*);	Length of *t* (length of byte array obtained with t.getBytes())
HiRDBVARCHAR	*variable* = null;	32,000 bytes
	variable = new HiRDBVARCHAR(int *n*);	*n* bytes (1 ≤ *n* ≤ 32,000)
	variable = new HiRDBVARCHAR(String *t*)	Length of *t* (length of byte array obtained with t.getBytes())
HiRDBNCHAR	*variable* = null;	30,000 bytes (15,000 double-byte characters)
	variable = new HiRDBNCHAR(int *n*);	(*n**2) bytes (*n* double-byte characters) (1 ≤ *n* ≤ 15,000)
	variable = new HiRDBNCHAR(String *t*)	Length of *t* (length of (byte array/2) obtained with t.getBytes())
HiRDBNVARCHAR	*variable* = null;	32,000 bytes (16,000 double-byte characters)
	variable = new HiRDBNVARCHAR(int *n*);	(*n* × 2) bytes (*n* double-byte characters) (1 ≤ *n* ≤ 16,000)
	variable = new HiRDBNVARCHAR(String *t*)	Length of *t* (length of (byte array/2) obtained with t.getBytes())

Data type	Initial value	Length of data set in SQL Descriptor Area
`HiRDBMCHAR`	*variable* = `null;`	30,000 bytes
	variable = `new HiRDBMCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 30,000)
	variable = `new HiRDBMCHAR(String` *t*`)`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)
`HiRDBMVARCHAR`	*variable* = `null;`	32,000 bytes
	variable = `new HiRDBMVARCHAR(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 32,000)
	variable = `new HiRDBMVARCHAR(String` *t*`)`	Length of *t* (length of `byte` array obtained with `t.getBytes()`)
`HiRDBDECIMAL`	*variable* = `null;`	Precision 15, scale 0
	variable = `new HiRDBDECIMAL(int` *p*`,int` *s*`);`	Precision *p*, scale *s* (1 ≤ *p* ≤ 29, 0 ≤ *s* ≤ *p*)
	variable = `new HiRDBDECIMAL(String` *t*`)`	The precision is the character string length obtained when the sign and period characters are subtracted from *t*. The scale is the character string length after the period (excluding the period).
	variable = `newHiRDBDECIMAL(java.math.BigDecimal` *t*`)`	The precision is the character string length obtained when the flag and period characters of the character string retrieved with `toString()` are subtracted from *t*. The scale is the value retrieved by the `scale()` method of the `BigDecimal` object.
`HiRDBBLOB`	*variable* = `null;`	1 megabyte
	variable = `new HiRDBBLOB(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)
	variable = `new HiRDBBLOB(byte[]` *t*`)`	Length of *t*
`HiRDBBINARY`	*variable* = `null;`	1 megabyte
	variable = `new HiRDBBINARY(int` *n*`);`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)
	variable = `new HiRDBBINARY(byte[]` *t*`)`	Length of *t*

Data type	Initial value	Length of data set in SQL Descriptor Area
`Java.math.BigDecimal`	*variable* `= null;`	Precision 15, scale 0
	variable `= new java.math.BigDecimal;`	The precision is set to the character string length obtained when the flag and period characters in the character string retrieved by `toString()` are subtracted from the `BigDecimal` object. The scale is set to the value retrieved by the `scale()` method.
`byte[]`	*variable* `= null;`	1 megabyte
	variable `= new byte[int` *n*`]`	*n* bytes (1 ≤ *n* ≤ 2,147,483,647)

18.3.6 Using data types when a cursor is declared (limited to the native interface version)

When using HiRDB data types when a cursor is declared, specify the data types as shown in Table 18-6.

Table 18-6: Description when a cursor is declared, and the acceptance area setting

Data type	Description when cursor declared	Acceptance area setting
`HiRDBCHAR`	`#sql iterator` *cursor-name*`(HiRDBCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 30,000)
`HiRDBVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBVARCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 32,000)
`HiRDBNCHAR`	`#sql iterator` *cursor-name*`(HiRDBNCHAR(int n));`	(n × 2) bytes (*n* double-byte characters) (1 ≤ n ≤ 15,000)
`HiRDBNVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBNVARCHAR(int n));`	(*n* × 2) bytes (*n* double-byte characters) (1 ≤ n ≤ 16,000)
`HiRDBMCHAR`	`#sql iterator` *cursor-name*`(HiRDBMCHAR(int n));`	*n* bytes (1 ≤ n ≤ 30,000)
`HiRDBMVARCHAR`	`#sql iterator` *cursor-name*`(HiRDBMVARCHAR(int n));`	*n* bytes (1 ≤ *n* ≤ 32,000)
`HiRDBDECIMAL`	`#sql iterator` *cursor-name*`(HiRDBMVARCHAR(int p,int s));`	Precision *p*, scale *s* (1 ≤ *p* ≤ 29, 0 ≤ *s* ≤ p)

Data type	Description when cursor declared	Acceptance area setting
`HiRDBBLOB`	`#sql iterator` *cursor-name*`(HiRDBBLOB(int n));`	*n* bytes ($1 \leq n \leq$ 2,147,483,647)
`HiRDBBINARY`	`#sql iterator` *cursor-name*`(HiRDBBINARY(int n));`	*n* bytes ($1 \leq n \leq$ 2,147,483,647)

18.3.7 Description of connection to and disconnection from a HiRDB server

SQLJ has no `CONNECT` or `DISCONNECT` statement. Therefore, connection to or disconnection from a HiRDB server is coded as Java instructions.

(1) Connection to a HiRDB server

To connect to a HiRDB server, use the following coding using a connection context.

(a) Defining a connection context class

Define a class for the connection context. *Class-name* indicates a Java identifier. The defined class inherits `sqlj.runtime.ConnectionContext`.

```
#sql modifier context class-name ;
```

(b) Declaring connection context

Using the declared class, declare the connection context (as a Java variable declaration). *Connection-context* indicates a Java identifier.

```
modifier class-name connection-context ;
```

(c) Connecting to a HiRDB server

Create a connection context object using a new operator. During this step, connection is made to the HiRDB server. For the connection parameters, describe the HiRDB server at the connection destination, port number, authorization identifier, and password in the same format as that used for JDBC.

```
connection-context = new class-name(connection-parameter) ;
```

(d) Connecting to the HiRDB server when the native interface is used

When the native interface is used, there are three ways of connecting to the HiRDB server:

- Describing the connection as a Java instruction
- Using the `CONNECT` statement

- Using the JDBC connection object (`Connection`)

These connection methods are described below.

1. Describing the connection as a Java instruction

Use the `new` operator to generate a connection context object. However, since JDBC is not being used, specify an authorization identifier, a password, a server name, and a port number in the connection parameters.

```
connection-context = new class-name(connection-parameters);
```

If no connection parameters are specified, the HiRDB server checks the client environment variables.

```
connection-context = new class-name();
```

An example of creating a connection context follows:

```
#sql context Ctx;
String Userid=new String("user1");
String Passwd=new String("puser1");
String Host=new String("HiRDB_SV");
short port=22000;

Ctx con = new Ctx(:Userid,:Passwd,:Host,:port);
```

2. Using the CONNECT statement

Specify an authorization identifier and a password in the connection parameters.

The HiRDB server checks the client environment definitions for the port number and the server name.

```
#sql [connection-context]{CONNECT USER :embedded variable USING :embedded variable};
or
#sql [connection-context]{CONNECT :embedded variable IDENTIFIED BY :embedded variable};
```

If connection parameters are not specified, the HiRDB server checks the client environment definitions.

```
#sql [connection-context]{CONNECT};
```

An example of the `CONNECT` statement follows:

```
#sql context Ctx;
String Userid=new String("user1");
String Passwd=new String("puser1");
Ctx con;

#sql [con] {CONNECT USER :Userid USING :Passwd };
```

3. Using the JDBC connection object (Connection)

Use the `new` operator to generate a connection context object. In the connection parameters, specify the JDBC connection object (`java.sql.Connection`).

```
connection-context = new class-name(connection-object);
```

An example of creating a connection context follows:

```
#sql context Ctx;
java sql.Connection con =

java.sql.DriverManager.getConnection("jdbc:hitachi:PrdbDrive://DBID=22200,
DBHOST=HiRDB_SV","user1","user1");

Ctx ctx = new Ctx(con);
```

(2) Disconnecting from a HiRDB server

To disconnect from the HiRDB server, invoke the `close` method for the connection context. Note that there is no reconnection method. To reconnect, create a new object.

```
connection-context.close() ;
```

An example of invoking the `close` method for the connection context follows:

```
#sql context DeptContext;
           ...
{
    DeptContext deptCtx = new DeptContext(deptURL,true);
    #sql [deptCtx] { DELETE FROM TAB };
    deptCtx.close();
}
```

When using the native interface version, you can use the `DISCONNECT` statement instead of invoking the close method for the connection context.

```
#sql[connection-context]{DISCONNECT};
```

An example of the `DISCONNECT` statement follows:

```
#sql context Ctx;
Ctx con;
#sql[con]{CONNECT};

#sql[con]{DISCONNECT};
```

(3) Default connection

(a) Standard interface version

For the standard interface version, the default connection context is assumed if no connection context is specified in an SQL statement.

To use the default connection context, a UAP must create a connection context in advance, and set it as the default connection context. Once the default connection context is set, it remains valid until the `close()` method for the default connection context is issued or a new connection context is set as the default connection context.

The default connection context is held by a variable inside the default connection context class (`JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext`).

The default connection context has multiple constructors that have the different arguments described as follows.

- Constructor that has a JDBC connection object as an argument
- Constructor that has the URL of the connection destination, authorization identifier, password, and auto commit specification as arguments
- Constructor that has the specifications of the URL of the connection destination, `Properties` object, and `autoCommit` as arguments
- Constructor that has connection context as an argument

To specify the URL of the connection destination, authorization identifier, and password, use the same format as is used for the JDBC driver of HiRDB.

In SQLJ, to use a constructor that includes a connection URL during the creation of a connection context, you must specify `autoCommit`, and specify `TRUE` to enable it and `FALSE` to disable it.

If the default connection context is created from the JDBC connection context, the `autoCommit` setting in the JDBC connection context is inherited.

- Creating and setting the default connection context

  An example for creating and setting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                         ...
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
```

- Releasing and resetting the default connection context

  An example of releasing and resetting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext;
                           ...
PrdbContext pctx = new PrdbContext(url,user,passwd,autoCommit);
PrdbContext.setDefaultContext(pctx);
                 ...
pctx.close();
PrdbContext new_pctx = new PrdbContext(url.use,passwd,autoCommit);

PrdbContext.setDefaultContext(new_pctx);
```

- Acquiring the default connection context

  When the following method is invoked, the connection context can be acquired:

  `JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefault Context();`

  An example of specifying the default context follows:

```
void print_address(String name) throws SQLException;
{
  String telno;
  sqlj.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno
               FROM PERSON
               WHERE :name = NAME } ;
}
```

(b) Native interface version

For the native interface version, the default connection context class is held in a variable of `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext`.

The default connection context class has the following constructors:

- Constructors with JDBC connection objects as arguments
- Constructors with the authorization identifier, password, server name, and port number of the connection destination as arguments

- Constructors with the authorization identifier and password specification of the connection destination as arguments
- Constructors with connection contexts as arguments
- Constructors without arguments

■ Creating and setting the default connection context

An example of creating and setting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                         :
PrdbContext pctx = new PrdbContext();
PrdbContext.setDefaultContext(pctx);
```

■ Releasing and resetting the default connection context

An example of releasing and resetting the default connection context follows:

```
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext;
                         :
PrdbContext pctx = new PrdbContext(user,passwd,host,port);
PrdbContext.setDefaultContext(pctx);
                         :
pctx.close();
PrdbContext new_pctx = new PrdbContext(user,passwd,host,port);

PrdbContext.setDefaultContext(new_pctx);
```

■ Getting the default connection context

To get the default connection context, invoke the following method:

```
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext.getDefau
ltContext();
```

A coding example in which the default context is implicitly specified follows:

```
void print_address(String name) throws SQLException;
{
  String telno;
  JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ConnectionContext ctx;
  ctx = JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext.getDefaultContext();
  #sql [ctx] { SELECT TELNO INTO :telno FROM PERSON WHERE :name = NAME } ;
}
```

18.3.8 Description of cursor-based retrieval

Because SQLJ has no `DECLARE CURSOR`, `OPEN`, or `CLOSE` statements, cursor declaration, opening, and closing must be coded as Java instructions. During this step, an iterator object is used in place of a cursor name. Because the iterator object is declared as a reference variable to an object, the same naming rule and valid range as in the Java rules apply here.

Depending on the iterator object type used, the retrieval result can be obtained using or not using a `FETCH` statement. A `FETCH` statement uses an object in the iterator type with a position specification and cannot use an object in the iterator type with a column name specification.

(1) Retrieval using a FETCH statement

The method for describing a retrieval using a `FETCH` statement is explained as follows.

(a) Defining a class for an iterator with position specification and declaring an iterator object

- Standard interface version

For the standard interface version, define a class for an iterator with a position specification and declare an iterator object. *class-name* indicates a Java identifier. *data-type-N* indicates the data type of a Java variable that stores the *N*-th retrieval item in the `FETCH` statement.

```
#sql modifier iterator class-name
              (data-type-1,data-type-2,...) ;
modifier class-name iterator-object ;
```

- Native interface version

For the native interface version, the specification is as follows:

```
#sql modifier iterator class-name
          [ implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate ]
          [ with keyword=setting-value,...]
              (data-type-1,data-type-2,...) ;
modifier class-name iterator-object ;
```

If an iterator is used in an `UPDATE` or `DELETE` statement, the `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate` interface is inherited.

keyword in the `WITH` clause indicates the function of the iterator. Only a constant can be specified. Table 18-7 shows the combinations of *keyword* in the `WITH` clause and setting values.

Table 18-7: Combinations of keyword in the WITH clause and setting values

Keyword in the WITH clause	Function	Setting
`holdability`	Indicates a holdable cursor.	`TRUE`
`updateColumns`	Indicates the column to be updated.	`"`*column-name,column-name*`,...“`

(b) Defining and opening a cursor

Substitute the result set from the `SELECT` statement into the declared iterator object.

```
#sql [context] iterator-object = { SELECT-statement } ;
```

(c) Extracting the retrieval result

Specify an iterator object instead of a cursor name and execute a `FETCH` statement. The iterator object must be preceded by a colon.

```
#sql [context] {
   FETCH :iterator-object INTO :variable-1,:variable-2,...} ;
```

(d) Determining NOT FOUND

Invoke the `endFetch` method for the iterator object and determine whether the result is `NOT FOUND`. If there is no row to be retrieved, `true` is returned. If the next row is found, `false` is returned. If the `endFetch` method is invoked after the cursor is closed, `true` is returned.

```
while(! iterator-object.endFetch()) {
     processing-on-the-extracted-row
}
```

(e) Closing the cursor

To close the cursor, invoke the `close` method.

```
iterator-object.close() ;
```

An example of a retrieval using a `FETCH` statement follows:

```
#sql public iterator ByPos(String, int);
                  :
{
  ByPos positer;
  String name = null;
  int code = 0;

  #sql positer = { SELECT PNAME,PCODE FROM STOCK };
  #sql { FETCH :positer INTO :name,:code };
  while( !positer.endFetch() ){
    System.out.println(name + ":" + code);
    #sql { FETCH :positer INTO :name,:pcode };
  }
  positer.close();
}
```

(2) Retrieval without using a FETCH statement

Using the fields in the iterator with a column name specification, read out each column of the retrieval result.

(a) Defining a class for an iterator with column name specification

Define the same name (not case sensitive) as the retrieval item as the class field. For the data type, specify the data type of the Java variable that receives the retrieval result. This class cannot be used for the native interface.

If the retrieval item is a value expression, a column name that includes a character that cannot be used in Java, for example, use an `AS` clause to define an alias for the retrieval item, and use that alias.

```
#sql modifier iterator class-name
            (data-type-1 column-name-1,
             data-type-2 column-name-2,...) ;
modifier class-name iterator-object;
```

(b) Defining and opening a cursor

Substitute the result set from the `SELECT` statement into the declared iterator object.

```
#sql [context] iterator-object = { SELECT-statement } ;
```

(c) Extracting the next row and determining NOT FOUND

Invoke the `next` method for the iterator object and determine whether the result is `NOT FOUND`. If the result is `NOT FOUND`, `TRUE` is returned. If a row is found, `FALSE` is returned. After the cursor is opened, it is not positioned on the first line of the retrieval result until the first `next` method is executed.

```
while(iterator-object.next()){
      processing-on-the-extracted-row
}
```

(d) Acquiring the retrieval result

Read data out from each field of the iterator object. If the result is `NOT FOUND` or if data is read out after the cursor is closed, the result is undetermined. If data is read out when the data type of the field is the Java basic data type and the retrieval result is a null value, the `SQLNullException` object occurs.

Data substituted into a field is not reflected in the database.

```
variable-1 = iterator-object.column-name-1 ;
variable-2 = iterator-object.column-name-2 ;
                ...
```

(e) Closing the cursor

To close the cursor, invoke the `close` method for the iterator object.

```
iterator-object.close() ;
```

An example of retrieval without using a `FETCH` statement follows:

Example:

```
#sql public iterator ByName(String pname,
                           int pcode);
                 :
{
   ByName nameiter;
   String s;
   int i;

   #sql nameiter = { SELECT PNAME, PCODE FROM STOCK };
   while( nameiter.next() ){
     s = nameiter.pname();
     i = nameiter.pcode();
     System.out.println(s + ":" + i);
   }
   nameiter.close();
}
```

(3) Updating using the cursor

For the native interface, a cursor can be used to update data.

To use an `UPDATE` or `DELETE` statement to manipulate the row on which the cursor is positioned, specify an iterator instead of a cursor name. Note that when the class for the iterator is being defined, it must inherit the `ForUpdate` interface.

```
#sql [context] { DELETE-statement WHERE CURRENT OF :iterator-object } ;

#sql [context] { UPDATE-statement WHERE CURRENT OF :iterator-object } ;
```

An example of an update that uses an iterator follows:

```
#sql public iterator ByPos
            implements JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate
            (String, int);
             :

{
  ByPos positer;
  String name = null;
  int year = 0;
  int newyear;

  #sql positer = { SELECT FULLNAME, BIRTHYEAR FROM PEOPLE };
  #sql { FETCH :positer INTO :name,:year };
  while( !positer.endFetch() ){
    newyear=year+10;
    #sql { UPDATE PEOPLE SET YEAR=:newyear WHERE CURRENT OF :positer; };
  }
  positer.close();
}
```

18.3.9 Receiving a dynamic result set

To receive a dynamic result set by invoking a procedure that returns a dynamic result set, use the `getNextResultSet()` method for the execution context. For the native interface version, a procedure that returns a dynamic result set cannot be used because JDBC result sets cannot be used.

The `getNextResultSet()` method returns a dynamic result set (`ResultSet` object) as a return value. Every time this method is invoked, it returns the next result set. After it returns the last result set, it returns a null value.

For a procedure or SQL statement that does not return a dynamic result set, a null value is returned. A null value is returned also when SQL execution is not normally terminated.

If an error occurs during the execution of the `getNextResultSet()` method, the `SQLException` occurs.

An example follows:

```
#sql  [execCtx] { CALL MULTI_RESULTS() };
 ResultSet rs;
 while((rs == execCtx.getNextResultSet() ) != null){
   processing-of-the-retrieval-result;
   rs.close();
 }
```

18.3.10 Using JDBC and SQLJ together

This subsection explains how to use JDBC and SQLJ together.

(1) Acquiring a JDBC result set from an SQLJ iterator

You can convert an SQLJ iterator into a JDBC result set (`ResultSet` object) and use the JDBC API to obtain the retrieval result. For the native interface version, JDBC result sets cannot be obtained.

To obtain a JDBC result set, use the `getResultSet` method for the iterator class (`ResultSetiterator`). This method returns a JDBC result set as a return value. After executing the `next` method for the iterator, do not invoke the `getResultSet` method.

After you have used the `getResultSet` method to convert an SQLJ iterator into a JDBC result set, do not receive a retrieval result using the original iterator.

An example follows:

```
public void showEmployeeName() throws SQLException
{
sqlj.runtime.ResultSetIterator iter;
#sql iter = { SELECT ename FROM rmp } ;
ResultSet rs = iter.getResultSet();
while(rs.next()){
System.out.println("employee name: " + rs.getString(1));
}
iter.close();
}
```

(2) Reading a JDBC result set as an iterator result set of SQLJ (limited to the standard interface version)

The JDBC result set (`ResultSet`) that was created using the JDBC API is converted with the `CAST` statement and read as a result set of the SQLJ cursor.

The coding example follows:

```
#sql iterator Employees(String ename, double sal);
Statement stmt=conn.createStatement();
String query="SELECT pname, pcode FROM stock WHERE pcode > 1000";
ResultSet rs=stmt.executeQuery(query);
Employees emps;
#sql emps ={CAST :rs };
```

(3) Converting JDBC connection into SQLJ connection context

The SQLJ connection context defines a constructor for generating an object from a JDBC connection. Using this constructor, you can convert a JDBC connection into an SQLJ connection context. Note that JDBC connection is transferred as an argument of the constructor. You can also use both types of connection together.

An example follows:

```
java.sql.Connection jdbcConCtx =java.sql.DriverManager.getConnection(...);
 #sql context Inventory;
Inventory sljConCtx = new Inventory(jdbcConCtx);
```

(4) Converting SQLJ connection into JDBC connection

You can use the `getConnection` method to get the JDBC connection from an SQLJ connection. You can also use both types of connection together.

With the native interface version, an SQLJ connection cannot be converted into a JDBC connection. To use the same connection as JDBC, you must create a connection in JDBC beforehand, and then convert the connection into an SQLJ connection context.

An example follows:

```
#sql context Inventory;
Inventory sljConCtx = new Inventory(url);
java.sql.Connection jdbcConCtx = sqljConCtx.getConnection();
```

(5) Dynamic SQL statement

SQLJ can describe only static SQL statements. Therefore, to execute a dynamic SQL statement, you must use the JDBC API.

(a) Executing a dynamic SQL statement

A dynamic SQL statement is executed using a `PreparedStatement` object in JDBC.

When the `prepareStatement` method for the connection context is executed using the SQL as an argument, a `PreparedStatement` object is returned as a return value.

To set a parameter in a dynamic SQL statement, use the `set` method of `PreparedStatement`. To execute the dynamic SQL statement, use the `execute` method of the `PreparedStatement` object.

An example of dynamic SQL execution follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
          "INSERT INTO FOO_TABLE VALUES(?, ?)");
pstmt.setInt(1, 100);
pstmt.setString(2, "test");
pstmt.execute();
```

(b) Retrieving a dynamic cursor

Only static cursors can be used in SQLJ. Therefore, to use a dynamic cursor, you must use the JDBC API.

When the `prepareStatement` method for the connection context is executed for a character string that indicates a `SELECT` statement, a `PreparedStatement` object is returned as a return value.

To set a parameter, use the `set` method of `PreparedStatement`. To execute the SQL statement, use the `executeQuery` method of the `PreparedStatement` object. The `executeQuery` method returns the JDBC result set.

To receive a retrieval result, use the `get` method for result sets.

An example of retrieval using a dynamic cursor follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
          "SELECT NAME, POINT FROM FOO_TABLE WHERE BAR=100");
ResultSet rs = pstmt.executeQuery();
String name;
Integer point;
rs.next();
name = pstmt.getString(1);
point = pstmt.getInteger(2);
```

(c) Executing a DESCRIBE statement

To determine the column name and data type of each retrieval item of a dynamic cursor, use a `ResultSetMetaData` object. You can obtain a `ResultSetMetaData` object from the `getMetaData` object of the result set.

You can also use the `getColumnClassName` method of the `ResultSetMetaData` object to obtain the character string that indicates the data type of each retrieval item.

You can use the `getColumnName` method to obtain column names.

Specify the items to be retrieved using numbers (beginning with `1`). You can use the `getColumnCount` method to obtain the number of columns.

An example of executing the `DESCRIBE` statement follows:

```
java.sql.PreparedStatement pstmt = con.prepareStatement(
     "SELECT * FROM FOO_TABLE");
java.sql.ResultSetMetaData aMeta = pstmt.getMetaData();
int columCount = aMeta.getColumnCount();
Vector nameList = new Vector();
Vector classLis = new Vector();
for(int i = 1; i <= columnCount; i++){
     nameList.addElement(aMeta.getColumnName(i));
     classList.addElement(a.Meta.getColumnClassName(i));
}
Vector dataList = new Vector();
rs.next();
for(int i = 1; i <= columnCount; i++){
     dataList.addElement(rs.getObject(i));
}
```

18.3.11 Creating and executing a UAP

(1) Executing the SQLJE translator

1.Set environment variables.

When the HiRDB client is the IPF version of UNIX:

Set environment variables as shown below. The underlined portion is the default installation directory.

- For HiRDB/Developer's Kit

`CLASSPATH=$CLASSPATH:/`*HiRDB*`/pdsqlj.jar`[1]

- For HiRDB/Run Time

`CLASSPATH=$CLASSPATH:/`*HiRDB*`/pdruntime.jar`[2]

`CLASSPATH=$CLASSPATH:/`*HiRDB*`/pdnativert.jar`[3]

[1] For the 32-bit mode HP-UX (IPF) version, this setting becomes `pdsqlj32.jar`.

[2] For the 32-bit mode HP-UX (IPF) version, this setting becomes `pdruntime32.jar`.

[3] For the 32-bit mode HP-UX (IPF) version, this setting becomes `pdnativert32.jar`.

When the HiRDB client is Windows:

Choose **Control Panel**, **System, System Properties**, and **Environment** in that order, and then specify as shown below. The underlined portion is the default installation directory:

- For HiRDB/Developer's Kit

`CLASSPATH=%CLASSPATH%:\`*HiRDB*`\pdsqlj.jar`

- For HiRDB/Run Time

  `CLASSPATH=%CLASSPATH%\`*HiRDB*`\pdruntime.jar`

  `CLASSPATH=%CLASSPATH%\`*HiRDB*`\pdnativert.jar`

2. Executing the SQLJ Translator

The SQLJ Translator runs on a Java virtual machine.

Format

```
pdjava [option] file-name-1.sqlj [file-name-2.java]
```

Description

option

Table 18-8 lists the SQLJ Translator options.

file-name-1

This is a UAP source file that describes SQLJ.

file-name-2

This is a post-source file.

file-name-1 and *file-name-2* may contain a path. If *file-name-2*`.java` is not specified, *file-name-1*`.java` is assumed.

Table 18-8: SQLJ Translator options

Options	Coding format	Explanation
`-dir`	`-dir=`*directory-name*	Specifies the direction in which to create the post-source file.
`-d`	`-d=`*directory-name*	
`-status`	`-status`	Displays the internal status for preprocessing. This is a debugging option.
`-J`	`-J-`*option*	Specifies a Java virtual machine option to be used during the execution of the SQLJ Translator.
`-version`	`-version`	Displays the version of the SQLJ translator. No translation is performed.
`-help`	`-help`	Specified to display an option explanation. No translation is performed.

Options	Coding format	Explanation
`-native`	`-native`	Generates a post source for the native interface. If you are specifying multiple options, be sure to specify this option first.
`-d 64`	`-d 64`	Specifies that the SQLJ translator is to be executed with the 64-bit mode HP-UX (IPF) version.

Notes

1. When specifying multiple options, use spaces to separate the options.

   Up to two options can be specified for the standard interface version, and up to three options (including `-native`) for the native interface version. If more options are specified, an error occurs.
2. The `-native` option for using the native interface version must be specified first. If the `-native` option is not specified first, an error occurs.
3. If the `-help` or `-version` option is specified, the other options are ignored. However, it both `-help` and `-version` are specified at the same time, both are valid.

Execution example

Execution examples are shown below.

- For the standard interface version

```
Example 1: pdjava file-name.sqlj
Example 2: pdjava -dir=d:\sqljsrc file-name.sqlj
Example 3: pdjava -d64 file-name.sqlj*
```

* This example is for the 64-bit mode HP-UX (IPF) version.

- For the native interface version

```
Example 1: pdjava -native file name.sqlj
Example 2: pdjava -native -dir=d:\sqljsrc file name.sqlj
Example 3: pdjava -native -d64 file name.sqlj*
```

* This example is for the 64-bit mode HP-UX (IPF) version.

(2) Compiling and executing an UAP

1. Setting environment variables

   See step 1 in *(1) Executing the SQLJ Translator*.

2. Compiling the post-source file

   Use the Java compiler to compile the post-source file generated by the SQLJ Translator. The format used for compilation follows:

```
javac file-name-2.java
```

3. Setting the path to the JDBC driver in CLASSPATH

   For details on setting up a path for the JDBC driver, see *16.1 Installation and environment setup*.

4. Using DriveManager to connect to a database

   For details about database connection using `DriverManager`, see *16.2.1 Driver class*.

5. Using the Java Virtual Machine to execute the CLASS file

   Use the Java Virtual Machine to execute the `Class` file. The execution format follows:

```
java file-name-2
```

When the 32-bit mode HP-UX (IPF) version is used, the execution format is as follows:

```
java -d64 file-name-2
```

18.3.12 Migrating an SQLJ source from the standard interface version to the native interface version

Some portions must be revised to migrate an SQLJ source from the standard interface version to the native interface version. Table 18-9 shows where revision is required to migrate to the native interface version.

Table 18-9: Migrating an SQLJ source to the native interface version

Command name	Standard interface version	Native interface version	Revision needed?
UAP (input) source	*file-name*`.sqlj`	*file-name*`.sqlj`	N

Command name	Standard interface version	Native interface version	Revision needed?
UAP (output) source	JAVA*source-file-name*. java*profile-name*.ser	JAVA*source-file*.java	N
Option	Specification of output file name, others	Specification of output file name, others	N
SQL prefix	#sql	#sql	N
SQL terminator	;	;	N
SQL declare section	Unnecessary	Unnecessary	N
Embedded variable	:*variable-name*	:*variable-name*	N
Declaration statement	#sql context *class-name* #sql iterator *class-name*	#sql context *class-name* #sql iterator *class-name*[1]	N[2]
Connection context creation	A JDBC connection object can be specified in a parameter.	A JDBC connection object can be specified in a parameter.	N
	An object other than a JDBC connection object can be specified in a parameter.	There is no object that obtains the same parameter.	Y[3]
Use of default connection context	JP.co.Hitachi.soft.HiR DB.sqj.runtime. PrdbContext	JP.co.Hitachi.soft.HiRDB.pd jpp.runtime. PrdbContext	Y[4]
Explicit specification of execution context	sqlj.runtime.Execution Context	JP.co.Hitachi.soft.HiRDB.pd jpp.runtime. ExecutionContext	Y[5]
Use of the CAST statement (acceptance of a JDBC result set)	Can be executed.	Cannot be executed.	Y[6]
Acceptance of dynamic result set	Can be executed.	Cannot be executed.	Y[7]
Data type	byte[] java.math.BigDecimal java.lang.String	JP.co.Hitachi.soft.HiRDB.pd jpp.runtime. HiRDBBLOB JP.co.Hitachi.soft.HiRDB.pd jpp.runtime. HiRDBDECIMAL JP.co.Hitachi.soft.HiRDB.pd jpp.runtime. HiRDBCHAR and others	Y[8]

Command name	Standard interface version	Native interface version	Revision needed?
Execution of different `SELECT` statements that use the same iterator object name	Can be executed.	Cannot be executed.	Y[9]

Legend:

Y: Need for revision.

N: No need for revision.

[1] A name iterator cannot be used. A position iterator can be used but not in an inner class.

[2] Revision becomes necessary when a name iterator or an inner class is used.

[3] Change the connection process. For details, see *18.3.7(1)(d) Connecting to the HiRDB server when the native interface is used*.

[4] Change the `JP.co.Hitachi.soft.HiRDB.sqj.runtime.PrdbContext` package name to `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.PrdbContext`.

[5] Change `sqlj.runtime.ExecutionContext` to `JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ExecutionContext`.

[6] `CAST` statements result in errors when translated. Therefore, delete all `CAST` statements. Modify the UAP so that it operates the JDBC result set directly and does not use an SQLJ iterator.

[7] Since there is no method that accepts a dynamic result set, an error occurs during Java compilation. Therefore, delete the section that issues `ExecutionContext.getNextResultSet()`. To get the dynamic result set, change the UAP so that is uses JDBC directly.

[8] `byte[]` is requested to the HiRDB server as the `BINARY` type. Modification is necessary if the HIRDB server is version 06-02 or earlier.

When the `BigDecimal` type is specified in an acceptance variable, the precision in set to `15` and the scale to `0`. Therefore, if any other precision or scale value is set, the value must be changed.

When `String` is specified in an input variable, it is requested to the HiRDB server as the `VARCHAR` type. If you want to associate the data type with a data type of the HiRDB server, you must change the data type.

9 The same iterator object name cannot be used to execute different SELECT statements. In this case, a separate iterator object name must be specified for each SELECT statement.

```
#sql iterator pos(HiRDBCHAR(10));
   :
pos positer = null
pos positer2 = null;
HiRDBCHAR out = null;
   :
#sql positer = {SELECT * FROM T1};
#sql {FETCH :positer INTO :out}
positer.close();

#sql positer2 = {SELECT * FROM T2};
#sql {FETCH :positer2 INTO :out}
positer2.close();
```

18.3.13 Notes about UAP development

When developing a UAP that uses multiple threads, do not use the default connection text as the connection context. If multiple threads use the same connection context, an error occurs.

When using multiple threads, be sure to specify the connection context explicitly. An example in which the connection context is specified explicitly follows:

```
#sql context Ctx;

public class sample{
 public void main(String args[]){
   Ctx con = null;
   #sql [con] {CONNECT};                          //Explicit
specification of connection context
   ...
   int data = 100;
   #sql [con] {INSERT INTO T1 VALUES(:data)}; //Explicit
specification of connection context

   #sql [con] {DISCONNECT};                       //Explicit
specification of connection context
 }
}
```

When using the SQLJ native interface version, match the number of SELECT statement retrieval items with the number of columns of the iterator object to be used. If the two do not match, false errors may occur.

18.4 Native Runtime

The SQLJ runtime library used by the native interface is called *Native Runtime*.

Native Runtime provides the following functions:

- Classes and interfaces used in compilation when the `-native` option is specified
- Access to HiRDB

18.4.1 Package configuration

Table 18-10 shows the configuration of the Native Runtime packages.

Table 18-10: Configuration of the Native Runtime packages

Package name	Collected contents
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	Classes and interfaces
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.error`	Error class

18.4.2 Public classes of Native Runtime

Table 18-11 lists the public classes of Native Runtime.

Table 18-11: Public classes of Native Runtime

Package	Class or interface name	Function
N/A	Connection context	This class is generated by `#sql context` *class-name*`;` of the SQLJ translator. This corresponds to a connection context of SQLJ.
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	`PrdbContext`	This is the default connection context. This corresponds to the default connection context of SQLJ.
`JP.co.Hitachi.soft.HiRDB.pdjpp.runtime`	`ExecutionContext`	This is an execution context. This class corresponds to an execution context of SQLJ and is used in managing SQL execution.
N/A	Iterator	This class is generated by `#sql iterator` *class-name*`;` of the SQLJ translator. This corresponds to an iterator of SQLJ.

Package	Class or interface name	Function
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	RTResultSet	This is a result set object. This class corresponds to a result set of JDBC and is used in managing results.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	ForUpdate	This interface is implemented by an iterator declaration when cursor update using an iterator is used.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBCHAR	Indicates the CHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBVARCHAR	Indicates the VARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNCHAR	Indicates the NCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBNVARCHAR	Indicates the NVARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMCHAR	Indicates the MCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBMVARCHAR	Indicates the MVARCHAR type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBDECIMAL	Indicates the DECIMAL type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBLOB	Indicates the BLOB type of HiRDB.
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime	HiRDBBINARY	Indicates the BINARY type of HiRDB.

Legend:

N/A: No package is available.

18.4.3 Cluster specifications

This section describes the method and field values of each class.

(1) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBCHAR class

Description

This class corresponds to the CHAR type of HiRDB.

Constructors

Return value	Method	Function description
HiRDBCHAR	HiRDBCHAR(String *s*) throws SQLException	Generates a new HiRDBCHAR class. If the length of the specified character string is 30,001 bytes or greater, SQLException is thrown.
HiRDBCHAR	HiRDBCHAR(int *len*) throws SQLException	Returns a HiRDBCHAR class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 30,000, SQLException is thrown.

Methods

Return value	Method	Function description
String	getString()	Returns the String object.
int	length()	Returns the character string length.

(2) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBVARCHAR class

Description

This class corresponds to the VARCHAR type of HiRDB.

Constructors

Return value	Method	Function description
HiRDBVARCHAR	HiRDBVARCHAR(String *s*) throws SQLException	Generates a new HiRDBVARCHAR class. If the length of the specified character string is 32,001 bytes or greater, SQLException is thrown.
HiRDBVARCHAR	HiRDBVARCHAR(int *len*) throws SQLException	Returns a HiRDBVARCHAR class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 32,000, SQLException is thrown.

Methods

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(3) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNCHAR class

Description

This class corresponds to the `NCHAR` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBNCHAR`	`HiRDBNCHAR(String` *s*`)` `throws SQLException`	Generates a new `HiRDBNCHAR` class. If the length of the specified character string is 15,001 bytes or greater, `SQLException` is thrown.
`HiRDBNCHAR`	`HiRDBNCHAR(int` *len*`)` `throws SQLException`	Returns a `HiRDBNCHAR` class that has a length of *len* (*len* is the number of double-byte characters). This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (double-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 15,000, `SQLException` is thrown.

Methods

Return value	Methods	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(4) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBNVARCHAR class

Description

This class corresponds to the `NVARCHAR` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBNVARCHAR`	`HiRDBNVARCHAR(String` *s*`)` `throws SQLException`	Generates a new `HiRDBNVARCHAR` class. If the length of the specified character string is 16,001 characters or greater, `SQLException` is thrown.

Return value	Method	Function description
HiRDBNVARCHAR	HiRDBNVARCHAR(int *len*) throws SQLException	Returns a HiRDBNVARCHAR class that has a length of *len* (*len* is the number of double-byte characters). This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (double-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 16,000, SQLException is thrown.

Methods

Return value	Method	Function description
String	getString()	Returns the String object.
int	length()	Returns the character string length.

(5) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMCHAR class

Description

This class corresponds to the MCHAR type of HiRDB.

Constructors

Return value	Method	Function description
HiRDBMCHAR	HiRDBMCHAR(String *s*) throws SQLException	Generates a new HiRDBMCHAR class. If the length of the specified character string is 30,001 bytes or greater, SQLException is thrown.
HiRDBMCHAR	HiRDBMCHAR(int *len*) throws SQLException	Returns a HiRDBMCHAR class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the OUT parameter of a CALL statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 30,000, SQLException is thrown.

Methods

Return value	Method	Function description
String	getString()	Returns the String object.
int	length()	Returns the character string length.

(6) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBMVARCHAR class

Description

This class corresponds to the `MVARCHAR` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBMVARCHAR`	`HiRDBMVARCHAR(String` *s*`) throws SQLException`	Generates a new `HiRDBMVARCHAR` class. If the length of the specified character string is 32,001 bytes or greater, `SQLException` is thrown.
`HiRDBMVARCHAR`	`HiRDBMVARCHAR(int` *len*`) throws SQLException`	Returns a `HiRDBMVARCHAR` class that has a length of `len`. This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that (single-byte space character × *len*) was specified. If the specified *len* value is not in the range from 1 to 32,000, `SQLException` is thrown.

Method

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`int`	`length()`	Returns the character string length.

(7) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBLOB class

Description

This class corresponds to the `BLOB` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBBLOB`	`HiRDBBLOB(byte[]` *b*`)`	Generates a new `HiRDBBLOB` class.
`HiRDBBLOB`	`HiRDBBLOB(int` *len*`) throws SQLException`	Returns a `HiRDBBLOB` class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that the number (`0(0x30)` × *len*) was specified. If the specified *len* value is 0 or less, `SQLException` is thrown.

Methods

Return value	Method	Function description
`byte[]`	`getBytes[]`	Returns `byte[]`.
`int`	`length()`	Returns the `byte[]` length.

(8) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBBINARY class

Description

This class corresponds to the `BINARY` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBBINARY`	`HiRDBBINARY(byte[]` *b*`)`	Generates a new `HiRDBBINARY` class.
`HiRDBBINARY`	`HiRDBBINARY(int` *len*`)` `throws SQLException`	Returns a `HiRDBBINARY` class that has a length of *len*. This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that the number `(0(0x30)` × *len*) was specified. If the specified *len* value is 0 or less, `SQLException` is thrown.

Methods

Return value	Method	Function description
`byte[]`	`getBytes()`	Returns `byte[]`.
`int`	`length()`	Returns the `byte[]` length.

(9) JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.HiRDBDECIMAL class

Description

This class corresponds to the `DECIMAL` type of HiRDB.

Constructors

Return value	Method	Function description
`HiRDBDECIMAL`	`HiRDBDECIMAL(String` *s*`)` `throws SQLException`	Generates a new `HiRDBDECIMAL` class. If the character string of the argument contains a character string other than a number, a period, and a sign or if the precision and scale values obtained from the character string are 30 or higher, `SQLException` occurs.

Return value	Method	Function description
`HiRDBDECIMAL`	`HiRDBDECIMAL (java.math.BigDecimal) throws SQLException`	Generates a new `HiRDBDECIMAL` class. If the precision and scale values in the arguments are 30 or higher, `SQLException` occurs.
`HiRDBDECIMAL`	`HiRDBDECIMAL(int` *x*`,int` *y*`) throws SQLException`	Returns a `HiRDBDECIMAL` class with precision *x* and scale *y*. This constructor is used to specify this class in a single-row retrieval or the `OUT` parameter of a `CALL` statement. If this constructor is specified in an input variable, the system assumes that `0` was specified. If *x* is not in the range from 1 to 29, *y* is not in the range from 0 to 29, and *x* is less than *y*, `SQLException` occurs.

Methods

Return value	Method	Function description
`String`	`getString()`	Returns the `String` object.
`java.math.BigDecimal`	`getBigDecimal()`	Returns the `java.math.BigDecimal` object.
`int`	`precision()`	Returns the precision.
`int`	`scale()`	Returns the scale.

18.4.4 Coding examples using the native interface

(1) Data insertion and retrieval

A coding example (`sample1.sqlj`) of data insertion and retrieval follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
//Declare iterator (cursor)
#sql iterator Pos(int,HiRDBCHAR(10),HiRDBNCHAR(5),HiRDBDECIMAL(10,5));

public class sample1{
  public static void main(String args[]){

    //Connect and create table
    try{
      #sql{CONNECT};      //Refer to client environment variable and connect
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};
```

```
    //Insert data
    try{
      int InInt = 100;
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("NCHAR");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());};


    //Retrieve data (FETCH)
    try{
      Pos sampleCur = null;
      int OutInt = 0;
      HiRDBCHAR OutChar = null;
      HiRDBNCHAR OutNchar = null;
      HiRDBDECIMAL OutDecimal = null;

      #sql sampleCur  = {SELECT * FROM SAMPLE1};
      while(true){
        #sql {FETCH :sampleCur INTO :OutInt ,:OutChar ,:OutNchar ,:OutDecimal };
        if(sampleCur.endFetch()) break;
          System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
             " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
      }
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(2) Data insertion and single-row retrieval

A coding example (`sample2.sqlj`) of data insertion and single-row retrieval follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
//Iterator (cursor) declaration
#sql iterator Pos(int,HiRDBCHAR(10),HiRDBNCHAR(5),HiRDBDECIMAL(10,5));

public class sample1{
  public static void main(String args[]){

    //Connection and table creation
    try{
      #sql{CONNECT};     //Refer to the client environment variables and connect.
      #sql{CREATE TABLE SAMPLE1(c1 int,c2 char(10),c3 nchar(5),c4 decimal(10,5))};
    }catch(SQLException e){System.out.println(e.getMessage());};
```

```
    //Insert data
    try{
      int InInt = 100;
      HiRDBCHAR InChar = new HiRDBCHAR("CHAR");
      HiRDBNCHAR InNchar = new HiRDBNCHAR("NCHAR");
      HiRDBDECIMAL InDecimal = new HiRDBDECIMAL("12345.678");

      #sql{INSERT INTO SAMPLE1 VALUES(:InInt,:InChar,:InNchar,:InDecimal)};
      #sql{COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());};


    //Retrieve data (single-row retrieval)
    try{
      //Declare output variables
      int OutInt = 0;
      HiRDBCHAR OutChar = new HiRDBCHAR(10);
      HiRDBNCHAR OutNchar = new HiRDBNCHAR(5);
      HiRDBDECIMAL OutDecimal = new HiRDBDECIMAL(10,5);

      #sql {SELECT * INTO :OutInt,:OutChar,:OutNchar,:OutDecimal FROM SAMPLE1};
      System.out.println("c1="+ OutInt +" c2="+ OutChar.getString() +
        " c3="+ OutNchar.getString() + " c4="+ OutDecimal.getString());
    }catch(SQLException e){System.out.println(e.getMessage());};
    try{#sql{DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(3) CALL statement execution

A coding example (`sample3.sqlj`) of `CALL` statement execution follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;

public class sample3{
  public static void main(String args[]){

    Integer PInteger1 = new Integer(99);
    Integer PInteger2 = new Integer(100);
    Integer PInteger3 = new Integer(101);try{
      #sql {CONNECT};
    }catch(SQLException e){System.out.println(e.getMessage());}
```

```
    try{
      #sql {DROP PROCEDURE PROCSQLJ};
      #sql {DROP TABLE PROCTABLE};
    }catch(SQLException e1){}

    try{
      #sql {CREATE TABLE PROCTABLE(c1 int, c2 int)};
      #sql {CREATE PROCEDURE PROC1(in p1 int,out p2 int,inout p3 int)
        begin
        insert into PROCTABLE values(p1,p3);
        select * into p2,p3 from PROCTABLE;
        end};
      #sql {COMMIT};
    }catch(SQLException e){System.out.println(e.getMessage());}


    try{
      #sql {CALL PROC1(in :PInteger1 ,out :PInteger2 ,inout :PInteger3 )};
    }catch(SQLException e){System.out.println(e.getMessage());}

    System.out.println("IN parameter PInteger1 = " + PInteger1 );
    System.out.println("OUT parameter PInteger2 = " + PInteger2 );
    System.out.println("INOUT parameter PInteger3 = " + PInteger3 );

    try{#sql {DISCONNECT};}catch(SQLException e){System.out.println(e.getMessage());}
  }
}
```

(4) Update using a cursor

A coding example (`sample4.sqlj`) of update using a cursor follows:

```
import java.sql.*;
import JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.*;
#sql iterator iterP implements
JP.co.Hitachi.soft.HiRDB.pdjpp.runtime.ForUpdate(short);

public class sample4{
  public static void main(String args[]){
    iterP positer = null;
    iterP positer2 = null;
    short indata;
    short indata2 = 0;
    short indata3 = 999;
    try{
      #sql {CONNECT};
      #sql {DROP TABLE CURTABLE};
    }catch(SQLException e){System.out.println(e.getMessage());}
```

```
    //Create table
    try{#sql {CREATE TABLE CURTABLE(c1 smallint)};
    }catch(SQLException e){System.out.println(e.getMessage());}
    //Insert data
    for(short i = 0;i < 5;i++){
      indata = i;
      try{#sql{INSERT INTO CURTABLE VALUES(:indata)};}catch(SQLException e){}
    }

    //Execute SELECT and update using cursor
    try{
      #sql positer = {SELECT * FROM CURTABLE};
    }catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer INTO :indata2};
        if(positer.endFetch()) break;
        System.out.println(indata2);
        #sql { UPDATE CURTABLE SET C1=:indata3 WHERE CURRENT OF :positer };
      }
    }catch(SQLException e){e.getMessage();}


    //Check update results
    try{#sql positer2 = {SELECT * FROM CURTABLE};}catch(SQLException e){}
    try{
      while(true){
        #sql {FETCH :positer2 INTO :indata2};
        if(positer2.endFetch()) break;
        System.out.println(indata2);
      }
    }catch(SQLException e){System.out.println(e.getMessage());}
    try{#sql{DISCONNECT};}catch(SQLException e){}
  }
}
```

Appendixes

A. SQL Communications Area

When SQL statements are executed, HiRDB sends a return code and related information to the UAP indicating whether or not the SQL statements executed normally. The area that receives this information is called the *SQL Communications Area*. This appendix explains the organization and contents of the SQL Communications Area and the expansion of the area.

For details about the use of the SQL Communications Area, see *3.6 SQL error identification and corrective measures*.

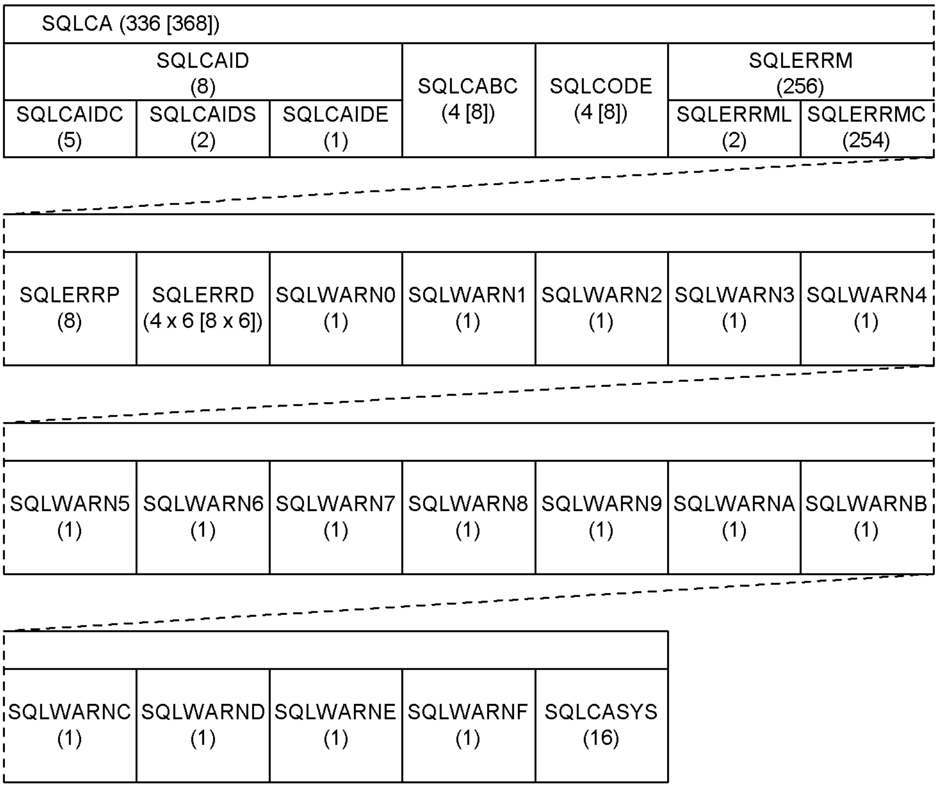
A.1 Organization and contents of the SQL Communications Area

The organization and contents of the area that receives SQL execution information are explained as follows.

(1) Organization of the SQL Communications Area

Figure A-1 shows how the SQL Communications Area is organized.

Figure A-1: Configuration of SQL Communications Area



Note

1. Numbers in parentheses indicate length (in bytes).
2. Brackets [ ] in parentheses enclose a value for 64-bit mode. For 64-bit mode Windows, SQLCA is 336 bytes.
3. In 64-bit mode, the length of `SQLCABC`, `SQLCODE`, and `SQLERRD` becomes the size of the `long` type for each platform.

(2) Contents of the SQL Communications Area

Table A-1 shows the contents of the SQL Communications Area.

Table A-1: Contents of the SQL Communications Area

Level number[1]	Communications area name	Data type	Length (bytes)	Description
1	SQLCA	—	336 [368]	Denotes the overall SQL Communications Area.
2	SQLCAID	—	8	Denotes the SQLCAIDC, SQLCAIDS, and SQLCAIDE areas.
3	SQLCAIDC	char	5	Contains a character string (SQLCA) indicating that the area is the SQL Communications Area.
3	SQLCAIDS	char	2	Used by HiRDB.
3	SQLCAIDE	char	1	Used by HiRDB.[2]
2	SQLCABC	long	4 [8][6]	Sets the size (336 [368] bytes) of the SQL Communications Area.
2	SQLCODE	long	4 [8][6]	Receives one of the following return codes from HiRDB after SQL statements have been executed: Negative: Abnormal termination 0: Normal termination Positive: Normal termination with a message For details about the messages associated with return codes, see the manual *HiRDB Version 8 Messages*. Return codes associated with messages are retrieved as follows: Return code Associated message ID -*yyy* KFPA11*yyy* -1*yyy* KFPA19*yyy* -3*yyy* KFPA18*yyy* *yyy* KFPA12*yyy* 3*yyy* KFPA13*yyy* Examples: Return code Message ID -125 KFPA11125 -1200 KFPA19200 -3200 KFPA18200 100 KFPA12100 3010 KFPA13010

Level number[1]	Communications area name	Data type	Length (bytes)	Description
2	SQLERRM	—	256	Denotes the SQLERRML and SQLERRMC areas. The contents of these areas vary depending on whether the return code returned to the SQLCODE area is positive or negative: • If the return code is negative, a character string indicating the location or the cause of the error can be returned • If the return code is positive, a character string indicating message information can be returned.
3	SQLERRML	short	2	Contains the length of the message returned to the SQLERRMC area.
3	SQLERRMC	char	254	Contains the message associated with the return code returned to the SQLCODE area; for the contents of this area, see the manual *HiRDB Version 8 Messages*.
2	SQLERRP	char	8	Used by HiRDB.
2	SQLERRD	long	4×6 $[8 \times 6]$[6]	Contains the internal status of HiRDB. This area is an array of six areas of the long data type: SQLERRD[0]: Not used SQLERRD[1]: Not used SQLERRD[2]: One of the following values.[3] • Number of rows retrieved by the SELECT statement • Number of rows updated by the UPDATE statement • Number of rows deleted by the DELETE statement • Number of rows inserted by the INSERT statement • Number of rows fetched by the FETCH statement • Number of rows created by the ASSIGN LIST statement SQLERRD[3]: Not used SQLERRD[4]: Not used SQLERRD[5]: Not used
2	SQLWARN0	char	1	W is set in this area when a warning flag (W) is set in any of the areas SQLWARN1-SQLWARNF.

Level number[1]	Communications area name	Data type	Length (bytes)	Description
2	SQLWARN1	char	1	W is set in this area if an embedded variable for receiving data during character data retrieval was shorter than the data, and the truncated value was received. W is also set if the embedded variable for receiving data during repetition retrieval had a smaller element count than the data and values of the discarded elements that were received; otherwise, this area is blank.
2	SQLWARN2	char	1	W is set in this area if the null value was ignored in set function processing; otherwise, this area is blank. However, in either of the following cases, a blank may be set in this area even if the null value was ignored during set function processing: • When a table that defines an index that recognizes a null value as an exception value is retrieved. • When the rapid grouping facility is used. This area is not used in remote database access.
2	SQLWARN3	char	1	W is set in this area if the number of columns containing the results of a retrieval did not match the number of embedded variables that received the results of the retrieval; otherwise, this area is blank[4].
2	SQLWARN4	char	1	W is set in this area if an UPDATE or DELETE statement without a WHERE clause was executed; otherwise, this area is blank. This area is not used in remote database access.
2	SQLWARN5	char	1	Spare
2	SQLWARN6	char	1	W is set in this area if the transaction was cancelled implicitly; otherwise, this area is blank.
2	SQLWARN7	char	1	W is set in this area if a repetition column with subscripts is specified in the SET or DELETE clause of the UPDATE statement, and the update is ignored because there are no elements in the row to be updated; otherwise, this area is blank. This area is not used in remote database access.
2	SQLWARN8	char	1	Spare
2	SQLWARN9	char	1	Spare

Level number[1]	Communications area name	Data type	Length (bytes)	Description
2	SQLWARNA[5]	char	1	W is set in this area if an invalid date occurred as a result of a date operation and HiRDB modified the date automatically to the last day of the affected month; otherwise, this area is blank. This area is not used in remote database access.
2	SQLWARNB[5]	char	1	W is set in this area if either an overflow error or division by zero error occurred in a computation during SQL statements execution and the result of the computation was set as a null value. Otherwise, this area is blank. This area is not used in remote database access.
2	SQLWARNC[5]	char	1	W is set in this area when the value for a day in a date interval is more than two digits after a date operation has been completed; otherwise, this area is blank. This area is not used in remote database access.
2	SQLWARND	char	1	W is set in this area when a warning that occurs in a foreign server cannot be classified into other types of SQLWARN.
2	SQLWARNE	char	1	Spare
2	SQLWARNF	char	1	Spare
2	SQLCASYS	char	16	Used by HiRDB.

—: Not Applicable.

Note

Value in brackets [ ] indicates the length for 64-bit mode. For 64-bite mode Windows, SQLCA is 336 bytes.

[1] Level numbers indicate the set inclusion relationships of the SQL Communications Area. The level 1 Communications Area is composed of level 2 Communications Areas.

[2] This area stores the type of database management system at the server with which remote database access was performed. The following values can be set in the SQLCAIDE area:

Value	Server's database management system	Remote database access protocol
K	SQL/K	OSI-RDA

Value	Server's database management system	Remote database access protocol
`O`	ORACLE	OSI-RDA
`P`	HiRDB	OSI-RDA
`R`	XDM/RD	OSI-RDA
`1`	RDB1 E2	OSI-RDA
(Blank)	Local access	Not applicable
Other	One of the following: • A database management system other than the above • Not connected to a server system (database management system not applicable)	OSI-RDA

[3] For remote database access, information depends on the distributed server as follows:

Returned value	DBMS at server	
	HiRDB and XDM/RD	Other than HiRDB or XDM/RD
Number of rows fetched by the `SELECT` statement	These values are set.	These values are set when a rows count is returned from the server DBMS; otherwise, `0` is set.
Number of rows updated by the `UPDATE` statement		
Number of rows deleted by the `DELETE` statement		
Number of rows inserted by the `INSERT` statement		
Number of rows fetched by the `FETCH` statement		These values are set.
Number of rows created by the `ASSIGN LIST` statement	These values are not usable.	These values are not usable.

[4] If the server is either HiRDB or XDM/RD, a remote database access operation results in an SQL error.

[5] The first `FETCH` statement returns `W` when an SQL statement containing sort processing or an SQL statement containing the `EXISTS` predicate is executed.

In the HiRDB/Parallel Server environment, the row that returns `W` cannot be determined if a warning is generated at the `WHERE` clause.

[6] In 64-bit mode, the length is the size of the `long` type for each platform.

A.2 Expanding the SQL Communications Area

The SQL Communications Area need not be described in the UAP, because it is expanded by the SQL preprocessor in the source program written in a high-level language.

The format of the SQL Communications Area expanded by the SQL preprocessor in a source program is shown as follows.

(1) C

This example shows SQL Communications Area expansion when C language is used.

```
#define  SQLCAIDE sqlca.sqlcaide
#define  SQLCODE  sqlca.sqlcode
#define  SQLERRML sqlca.sqlerrml
#define  SQLERRMC sqlca.sqlerrmc
#define  SQLERRMD sqlca.sqlerrmd
#define  SQLERRD0 sqlca.sqlerrd[0]
#define  SQLERRD1 sqlca.sqlerrd[1]
#define  SQLERRD2 sqlca.sqlerrd[2]
#define  SQLERRD3 sqlca.sqlerrd[3]
#define  SQLERRD4 sqlca.sqlerrd[4]
#define  SQLERRD5 sqlca.sqlerrd[5]
#define  SQLWARN0 sqlca.sqlwarn0
#define  SQLWARN1 sqlca.sqlwarn1
#define  SQLWARN2 sqlca.sqlwarn2
#define  SQLWARN3 sqlca.sqlwarn3
#define  SQLWARN4 sqlca.sqlwarn4
#define  SQLWARN5 sqlca.sqlwarn5
#define  SQLWARN6 sqlca.sqlwarn6
#define  SQLWARN7 sqlca.sqlwarn7
#define  SQLWARN8 sqlca.sqlwarn8
#define  SQLWARN9 sqlca.sqlwarn9
#define  SQLWARNA sqlca.sqlwarna
#define  SQLWARNB sqlca.sqlwarnb
#define  SQLWARNC sqlca.sqlwarnc
#define  SQLWARND sqlca.sqlwarnd
#define  SQLWARNE sqlca.sqlwarne
#define  SQLWARNF sqlca.sqlwarnf
typedef struct sqlca {
  char      sqlcaidc[5];   /* Table ID                   */
  char      sqlcaids[2];   /* Used by HiRDB              */
  char      sqlcaide;      /* Used by HiRDB              */
  long      sqlcabc;       /* SQLCA area length          */
  long      sqlcode;       /* SQLCODE                    */
  short     sqlerrml;      /* Effective message length   */
  char      sqlerrmc[254]; /* Message text               */
  char      sqlerrp[8];    /* Used by HiRDB              */
  long      sqlerrd[6];    /* HiRDB internal status      */
```

```
  char      sqlwarn0;       /* Warning information flag             */
  char      sqlwarn1;       /* Warning information 1                */
  char      sqlwarn2;       /* Warning information 2               */
  char      sqlwarn3;       /* Warning information 3                */
  char      sqlwarn4;       /* Warning information 4                */
  char      sqlwarn5;       /* Warning information 5                */
  char      sqlwarn6;       /* Warning information 6                */
  char      sqlwarn7;       /* Warning information 7                */
  char      sqlwarn8;       /* Warning information 8                */
  char      sqlwarn9;       /* Warning information 9                */
  char      sqlwarna;       /* Warning information 10               */
  char      sqlwarnb;       /* Warning information 11               */
  char      sqlwarnc;       /* Warning information 12               */
  char      sqlwarnd;       /* Warning information 13 (reserved)    */
  char      sqlwarne;       /* Warning information 14 (reserved)    */
  char      sqlwarnf;       /* Warning information 15 (reserved)    */
  char      sqlcasys1[16];  /* Reserved                             */
}SQLCA;
extern SQLCA sqlca;
```

(2) COBOL

The next example shows SQL Communications Area expansion when COBOL is used.

```
01  SQLCA IS EXTERNAL.
  02  SQLCAID      PIC X(8).
  02  FILLER       REDEFINES SQLCAID.
    03  SQLCAIDC   PIC X(5).
    03  SQLCAIDS   PIC X(2).
    03  SQLCAIDE   PIC X(1).
  02  SQLCABC      PIC S9(9) COMP.
  02  SQLCODE      PIC S9(9) COMP.
  02  SQLERRM.
    03  SQLERRML   PIC S9(4) COMP.
    03  SQLERRMC   PIC X(254).
  02  SQLERRP      PIC X(8).
  02  SQLERRD      PIC S9(9) COMP OCCURS 6 TIMES.
  02  SQLWARN.
    03  SQLWARN0   PIC X.
    03  SQLWARN1   PIC X.
    03  SQLWARN2   PIC X.
    03  SQLWARN3   PIC X.
    03  SQLWARN4   PIC X.
    03  SQLWARN5   PIC X.
    03  SQLWARN6   PIC X.
    03  SQLWARN7   PIC X.
  02  SQLEXT.
    03  SQLWARN8   PIC X.
```

```
  03  SQLWARN9   PIC X.
  03  SQLWARNA   PIC X.
  03  SQLWARNB   PIC X.
  03  SQLWARNC   PIC X.
  03  SQLWARND   PIC X.
  03  SQLWARNE   PIC X.
  03  SQLWARNF   PIC X.
02  SQLCASYS1    PIC X(16).
```

B. SQL Descriptor Area

Sometimes when SQL statements are assembled dynamically during execution of a UAP, the number and attributes of the I/O variables (data exchange areas) necessary for executing the SQL statements can be determined only when the UAP is executed. Therefore, HiRDB requires an area in which I/O variables are determined dynamically during UAP execution. The information in the area (the number, attributes, and I/O variable addresses) is posted to HiRDB via the `OPEN`, `FETCH`, or `EXECUTE` statement. The area is called the *SQL Descriptor Area*. The area can also be used by the `DESCRIBE` statement to receive information on SQL retrieval items that were preprocessed for dynamic execution.

For details about the UAP description languages that can use the SQL Descriptor Area, see *3.2 Overview of UAPs*.

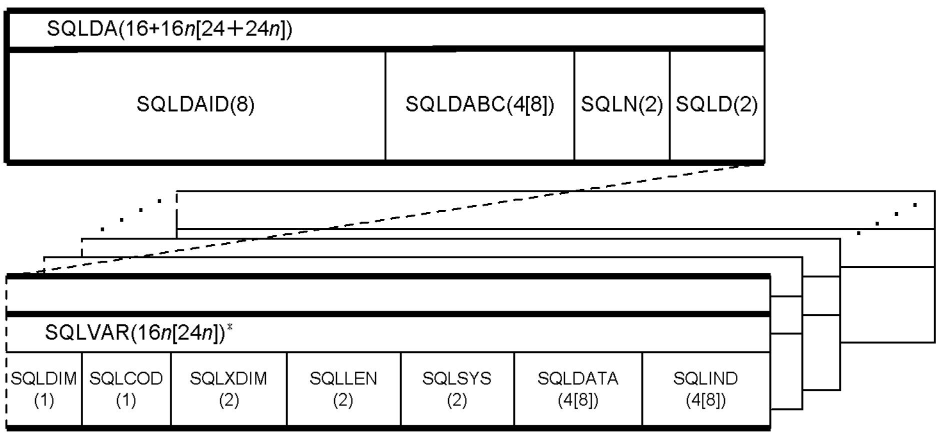
B.1 Organization and contents of the SQL Descriptor Area

This appendix explains the organization and contents of the areas that are described in the information on I/O variables, determined dynamically at the time of UAP execution.

(1) Organization of the SQL Descriptor Area

Figure B-1 shows how the SQL Descriptor Area is organized.

Figure B-1: Organization of the SQL Descriptor Area



Notes

1. Numbers in parentheses indicate length (in bytes).
2. *n* indicates the number of `SQLVAR`s specified in `SQLN`.

3. Square brackets ([ ]) enclose the length for 64-bit mode. For 64-bit mode Windows, SQLDA is 16 + 24*n* bytes.
4. In 64-bit mode, the length of `SQLDABC` is the size of the `long` type for each platform.

* If `BLOB`- or `BINARY`-type data is used, the area name is `SQLVAR_LOB`, which consists of `SQLDIM(1)`, `SQLCOD(1)`, `SQLXDIM(2)`, `SQLLOBLEN(4)`, `SQLDATA(4 [8])`, and `SQLLOBIND(4 [8])`.

Define the `SQLVAR_LOB` area in the `SQLVAR` area, and use it by overwriting the `SQLVAR` area during the input/output of `BLOB`-type data. For the contents of `SQLVAR_LOB`, see *Table B-3 Contents of SQLVAR_LOB*.

(2) Contents of the SQL Descriptor Area

Table B-1 shows the contents of the SQL Descriptor Area; for details about SQL data, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

Table B-1: Contents of the SQL Descriptor Area

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
1	`SQLDA`	—	16+16*n* [24+24*n*]	—	Denotes the overall SQL Descriptor Area.
2	`SQLDAID`	`char`	8	HiRDB	Contains the `SQLDA ID` (`SQLDA` ∆ ∆ ∆), indicating the `SQLDA`. This parameter is set when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
2	`SQLDABC`	`long`	4 [8][6]	HiRDB	Contains the length of `SQLDA`. This parameter is set when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
2	`SQLN`[2]	`short`	2	UAP	When an `SQLDA` area is allocated or SQLDA is used, this parameter specifies the number of `SQLVAR`s (1 to 4000) for the allocated SQLDA area.
				HiRDB	Binary `0` is set in this area if there is not enough SQLDA area (`SQLN` < `SQLD`) when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued.

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
2	SQLD	short	2	UAP	When the OPEN or EXECUTE statement is issued, specifies the number of input ? parameters in SQLD of the SQL Descriptor Area that is specified in the USING clause. When the EXECUTE statement is issued, specifies the number of output ? parameters in SQLDA of the SQL Descriptor Area that is specified in the INTO clause. When the FETCH statement is issued, specifies the number of retrieval items (1-4000).

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
				HiRDB	Binary 0, the number of retrieval items, or the number of output ? parameters is set when the DESCRIBE [OUTPUT] statement is issued: 0: The SQL statement that was preprocessed was a statement other than the SELECT statement and was not a CALL statement containing an output ? parameter Number of retrieval items: The SQL statement that was preprocessed was the SELECT statement Number of output ? parameters: The SQL statement that was preprocessed was the CALL statement The number of input ? parameters is set when the DESCRIBE INPUT statement is issued. The total number of user-defined type configuration elements that the system tried to receive and high-order user-defined type configuration elements being inherited is set when the DESCRIBE TYPE statement is issued. However, if the number of configuration elements exceeds 30,000, 30001 is set.
2	SQLVAR	--	16*n* [24*n*]	--	Area composed of the SQLDIM, SQLCOD, SQLXDIM, SQLLEN, SQLSYS, SQLDATA, and SQLIND areas. This set of areas should be defined at least as many times as the value specified in the SQLN area.
3	SQLDIM	unsigned char	1	--	Not used.

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
3	`SQLCOD`	`unsigned char`	1	UAP	A data code[3] is specified in this area when an `EXECUTE`, `OPEN`, or `FETCH` statement is issued.
				HiRDB	A data code[3] is set in this area after a `DESCRIBE` or `DESCRIBE TYPE` statement is issued.
3	`SQLXDIM`	`short`	2	UAP	One of the following values is specified, depending on the structure type of the area for the variable specified by `SQLDA` when the `EXECUTE`, `OPEN`, or `FETCH` statement is issued. Simple structure: `1` Repetition structure: `2` to `30000` (integer indicating maximum number of elements in the area) For details about data area structures, see *E. SQL Data Types and Data Descriptions*.
				HiRDB	One of the following values is set depending on the structure type of the retrieval item or `?` parameter when the `DESCRIBE` or `DESCRIBE TYPE` statement is issued. Simple structure: `1` Repetition structure: `2` to `30000` (integer indicating maximum number of members in the area)
3	`SQLLEN`[3,4]	`short`	2	UAP	A data length[3] is set in this area when an `EXECUTE`, `OPEN`, or `FETCH` statement is issued.
				HiRDB	A data length[3] is set in this area after a `DESCRIBE` or `DESCRIBE TYPE` statement is issued.

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
3	SQLSYS	short	2	UAP	The following value is specified when the EXECUTE, OPEN, or FETCH statement is issued: • Length of the area for one element that includes a gap when a variable-length character string type (VARCHAR, NVARCHAR, MVARCHAR) of a repetition structure or array structure is specified. • 0 for all other cases
				HiRDB	0 is set when the DESCRIBE or DESCRIBE TYPE statement is issued.
3	SQLDATA[5]	unsigned char	4 [8]	UAP	Specifies the address of the data area that stores the value of the ? parameter when either an EXECUTE or an OPEN statement is issued.[5] When a FETCH statement is issued, this area specifies the address of the data area that receives the data.
3	SQLIND[5]	short	4 [8]	UAP	Specifies the address of the area for receiving the value of the indicator variable only if a data code with an indicator variable is set in SQLCODE when an EXECUTE, OPEN, or FETCH statement is issued. The area for receiving the value of the indicator variable is 2 bytes. For details about indicator variable specification, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

Legend:

Δ : One blank.

— : Not applicable.

Note

Square brackets ([ ]) enclose the length for 64-bit mode. For 64-bit mode Windows, SQLCA is 16 + 24*n* bytes.

[1] Level numbers indicate the set inclusion relationships of the SQL Descriptor Area. For example, the level 1 data area is composed of level 2 data areas.

[2] The number of `SQLVAR`s set by a UAP in the `SQLN` area should be either the number of `?` parameters set in the `SQLD` area or a value greater than the number of retrieval items. If the number of `SQLVAR`s is less than the number of `?` parameters or less than the number of retrieval items, HiRDB posts this fact by returning binary `0` to the `SQLN` area.

[3] For details about the data codes and data lengths, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[4] For a packed decimal number (`DECIMAL`, `INTERVAL YEAR TO DAY`, or `INTERVAL HOUR TO SECOND`), the `SQLLEN` area is composed of the following areas:

Data area name	Data type	Length (bytes)	Description
`SQLPRCSN`	B	1	Precision (*p*)
`SQLSCALE`	B	1	Decimal scaling position (*s*)

[5] Because the `SQLDATA` and `SQLIND` areas are cleared when a `DESCRIBE` statement is executed, a value must be reset after the `DESCRIBE` statement has executed. For repetition columns, use the following structure to set a value:

Structure of variables for setting repetition columns in `SQLDATA`

4-byte, binary number area (area for storing the current element count)	Area for the first element of the data type indicated by `SQLCOD`	Area for the second element of the data type indicated by `SQLCOD`		Area for the *n*-th element of the data type indicated by `SQLCOD`

n indicates the maximum element count for the variable.

[6] In 64-bit mode, the length is the size of the `long` type for each platform.

Table B-2: Data codes and data lengths set in the SQL Descriptor Area

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit
0	00	—	Data type not available in HiRDB[1]	0	Bytes
48	30	No	C VARCHAR(*n*)[4]	1 ≤ *n* ≤ 32000[2]	
49	31	Yes			
68	44	No	ROW	Row length *L* in table to be operated on: 1 ≤ *L* ≤ 30,000	
69	45	Yes			
100	64	No	INTERVAL YEAR TO DAY	Precision 8 Decimal scaling factor 0	Digits
101	65	Yes			
110	6E	No	INTERVAL HOUR TO SECOND	Precision 6 Decimal scaling factor 0	
111	6F	Yes			
112	70	No	DATE	4	Bytes
113	71	Yes			
120	78	No	TIME	3	
121	79	Yes			
124	7C	No	TIMESTAMP[(p)]	7 ÷ ↑ *p* ÷ 2 ↑ *p* = 0, 2, 4, or 6	
125	7D	Yes			
131	83	—	Abstract data type[3]	—	—

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit
144	90	No	BINARY(*n*)	$1 \leq n \leq 2147483647$[2]	Bytes
145	91	Yes			
146	92	No	BLOB[(*n*)]	$1 \leq n \leq 2147483647$	
147	93	Yes			
154	9A	No	BINARY locator	4	
155	9B	Yes			
158	9E	No	BLOB locator	4	
159	9F	Yes			
160	A0	No	MVARCHAR(*n*)	$1 \leq n \leq 32000$[2]	
161	A1	Yes			
164	A4	No	MCHAR[(*n*)]	$1 \leq n \leq 30000$	
165	A5	Yes			
176	B0	No	NVARCHAR(*n*)	$1 \leq n \leq 16000$[2]	Characters
177	B1	Yes			
180	B4	No	NCHAR(*n*) or NATIONAL CHAR[ACTER](*n*)	$1 \leq n \leq 15{,}000$	
181	B5	Yes			
192	C0	No	VARCHAR(*n*)	$1 \leq n \leq 32{,}000$[2]	Bytes
193	C1	Yes			
196	C4	No	CHAR[ACTER](*n*)	$1 \leq n \leq 30{,}000$	
197	C5	Yes			
224	E0	No	FLOAT or DOUBLE PRECISION	8	
225	E1	Yes			
226	E2	No	SMALLFLT or REAL	4	
227	E3	Yes			

Decimal data code	Hexadecimal data code	Indicator variable	Data type	Data length	Unit
228	E4	No	[LARGE]DEC[IMAL] [(*p* [,*s*])]	Precision *p* Decimal scaling factor *s* $1 \leq p \leq 29, 0 \leq s \leq p$	Digits
229	E5	Yes			
234	EA	No	DISPLAY SIGN LEADING SEPARATE[5]	Precision *p* Decimal scaling factor *s* $1 \leq p \leq 29, 0 \leq s \leq p$	
235	EB	Yes			
240	F0	No	INT[EGER]	4	Bytes
241	F1	Yes			
244	F4	No	SMALLINT	2	
245	F5	Yes			

Legend:

—: Not applicable.

Note

During a remote database access to a distributed server other than HiRDB, the `DESCRIBE` statement converts the data types for the distributed server to the corresponding HiRDB data types. For details about data type conversion by the `DESCRIBE` statement, see *11.4 Available data types*.

[1] When a `DESCRIBE` statement that accesses remotely a DBMS other than HiRDB is executed, and there is no HiRDB data type corresponding to the data type at the server, data code `0` is set. Remote database access cannot access the data in a column for which this data code is set. You cannot set data code `0` in a UAP for SQL statements other than the `DESCRIBE` statement. Also, the HiRDB dictionary table does not include data code `0`. For details about the association between a distributed server and the data type when remote database access is performed using the distributed client facility, see *11.4 Available data types*.

[2] When a variable-length character string of `0` length is set in the UAP, `1` must be set in the `SQLLEN` area.

[3] When the `DESCRIBE` statement is executed, a data type is returned from the server. The UAP can reference data types. Data type setup and data length setup and

referencing are disabled.

[4] This data type can be set in C.

[5] This data type can be set in COBOL.

Table B-3: Contents of SQLVAR_LOB

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
2	SQLVAR_LOB	—	16*n* [24*n*]	—	Area that consists of SQLDIM, SQLCOD, SQLXDIM, SQLLOBLEN, SQLDATA, SQLDATA, and SQLLOBIND. Define this area in the SQLVAR area, and use it by overwriting the SQLVAR area during the input/out of BLOB- and BINARY-type data.
3	SQLDIM	unsigned char	1	—	Not used.
3	SQLCOD	unsigned char	1	UAP	Specifies a data code[2] when the EXECUTE, OPEN, or FETCH statement is issued.
				HiRDB	Contains a data code[2] after the DESCRIBE or DESCRIBE TYPE statement has been issued.
3	SQLXDIM	short	2	UAP	Specifies 1 when the EXECUTE, OPEN, or FETCH statement is issued. For details about the data area structures, see *E. SQL Data Types and Data Descriptions*.
				HiRDB	Contains the value 1 after the DESCRIBE or DESCRIBE TYPE statement has been issued.
3	SQLLOBLEN[2]	long [int]	4	UAP	Specifies the data length[2] when the EXECUTE, OPEN, or FETCH statement is issued.
				HiRDB	Contains the data length[2] after the DESCRIBE or DESCRIBE TYPE statement is issued.

Level number[1]	Data Area name	Data type	Length (bytes)	Source of value	Description
3	`SQLDATA`[3]	`unsigned char *`	4 [8]	UAP	When the `EXECUTE` or `OPEN` statement is issued, specifies the address of the data area in which a `?` parameter value is stored. When the `FETCH` statement is issued, specifies the address of the data area that receives the data.
3	`SQLLOBIND`[3]	`long *` `[int *]`	4 [8]	UAP	Specifies the address of the area for receiving the value of the indicator variable only if a data code with an indicator variable is set in `SQLCODE` when an `EXECUTE`, `OPEN`, or `FETCH` statement is issued. The area for receiving the value of the indicator variable is 4 bytes. For details about indicator variable specification, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

— : Not applicable.

Note

The square brackets in the data type and length columns indicate the data type and length in the 64-bit mode.

[1] Level numbers indicate the set inclusion relationships of the SQL Descriptor Area. For example, the level 2 data area is composed of the level 3 data areas.

[2] For details on data length and data codes, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[3] The `SQLDATA` and `SQLLOBIND` data areas are cleared when a `DESCRIBE` statement is executed. Therefore, if you use a `DESCRIBE` statement, reset the values for these data areas after executing the `DESCRIBE` statement. For the structure for setting a value in a repetition column, see footnote #5 in *Table B-1 Contents of the SQL Descriptor Area*.

B.2 Expanding the SQL Descriptor Area

The SQL Descriptor Area is allocated by means of a declaration within the UAP.

The format of the SQL Descriptor Area expanded in a source program is shown below, followed by an example.

(1) Expansion format of the SQL Descriptor Area

(a) C

This example shows SQL Descriptor Area expansion when C is used.

```
struct {
   char    sqldaid[8]; /* Table ID                                   */
   long    sqldabc;    /* Table length                               */
   short   sqln;       /* Elements count in SQLVAR array             */
   short   sqld;       /* ? parameters count, retrieval items count  */
   struct  sqlvar{     /* Data information area                      */
     unsigned char  sqldim;       /* Unused                   */
     unsigned char  sqlcod;       /* Data code                */
     short          sqlxdim;      /* Maximum elements count   */
     union {
       short          sqllen;     /* Data length              */
       struct {
         unsigned char  sqlprcsn; /* Precision                */
         unsigned char  sqlscale; /* Scale                    */
       } s_sqllen;
     } sqllen;
     short          sqlsys;       /* Unused                   */
     unsigned char  *sqldata;     /* Data area address        */
     short          *sqlind;      /* Indicator variable address */
   } SQLVAR[n];1
} sqlda;2
```

[1] *n* indicates the required number (1-30000).

[2] Any desired character string can be specified for the structure name (`sqlda` portion), except that no character string beginning with `SQL` is allowed.

(b) COBOL

This example shows SQL Descriptor Area expansion when COBOL is used.

```
01  USQLDA1
  02  USQLDAID          PIC  X(8)  VALUE  'SQLDA'.
  02  USQLDABC          PIC  S9(9)  COMP.
  02  USQLN             PIC  S9(4)  COMP.
  02  USQLD             PIC  S9(4)  COMP.
  02  USQLVAR   OCCURS  n  TIMES.2
    03  USQLTYPE        PIC  S9(4)  COMP.
  03  FILLER  REDEFINES  USQLTYPE.
      04  USQLDIM       PIC  X(1).
      04  USQLCOD       PIC  X(1).
    03  USQLXDIM        PIC  S9(4)  COMP  VALUE  IS  1.
 03  USQLATTR.
      04  USQLLEN       PIC  S9(4)  COMP.
```

```
    04  FILLER  REDEFINES  USQLLEN.
      05  USQLPRCSN  PIC  X(1).
      05  USQLSCALE  PIC  X(1).
    04  USQLSYS       PIC  S9(4)  COMP.
  03  FILLER  REDEFINES  USQLATTR.
    04  USQLLOBLEN   PIC  S9(9)  COMP.
  03  USQLDATA  USAGE  IS  ADDRESS.
  03  USQLIND   USAGE  IS  ADDRESS.
```

[1] Any name can be specified as the name of the set item (USQLDA area); however, a character string that begins with SQL cannot be used for a data item.

[2] *n* indicates the required number (1-30000).

(2) SQL Descriptor Area example

(a) Declaration and area allocation for using the SQL Descriptor Area

The SQL Descriptor Area is declared and allocated in the UAP.

(b) Collection of retrieval item information

This next example illustrates collecting retrieval item information. The items identified by numbers in the code are explained as follows.

```
EXEC SQL BEGIN DECLARE SECTION;   ................... 1
struct{   .......................................... 1
long*  cmd_len;   ................................. 1
char  cmd_data[1000];   .......................... 1
}XCMND;   .......................................... 1
EXEC SQL END DECLARE SECTION;   ..................... 1
XCMND.cmd_len=(long*)sprintf(XCMND.cmd_data,
                 "SELECT*FROM stock WHERE GNO=1")   .. 2
EXEC SQL WHENEVER SQLERROR GO TO :RERROR;   .......... 3
EXEC SQL PREPARE ST1 FROM :XCMND;   .................. 4
EXEC SQL DESCRIBE ST1 INTO :DAREA;   ................. 5
```

Notes

1. When a DESCRIBE statement is executed, binary 0 or the number of retrieval items is set in the SQLD area:

   - 0 is set when the SQL statement that was preprocessed is not a SELECT statement.

   - Number of retrieval items is set when the SQL statement that was preprocessed is a SELECT statement.

2. The data code, data length, and maximum elements count of each retrieval item are set in SQLCOD, SQLLEN, and SQLXDIM, respectively.

* In 64-bit mode, this is `int`.

Explanation

1. Declares an embedded variable (`XCMND`) for storing the SQL statements.
2. Sets the SQL statement in the variable (`XCMND`).
3. Specifies the action to be taken if an error occurs after SQL statement execution.
4. Preprocesses the SQL statements specified as the variable `XCMND` and assigns an SQL statement identifier (`ST1`).
5. Collects the information of items retrieved by the SQL statements (`ST1`) into the SQL Descriptor Area (`DAREA`).

(c) Fetching retrieval results with dynamic receive area allocation

In this example, retrieval results are fetched into areas allocated based on the information obtained using a `DESCRIBE` statement. The items in italics in the figure are explained as follows.

```
for(n=0;n<DAREA.sqld;n++){    .................... 1
   DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);
.. 1
    DAREA.SQLVAR[n].sqlind=&(X_IND[n]);    ....... 1
}   ............................................. 1
EXEC SQL DECLARE CR1 CURSOR FROM ST1;   ......... 2
EXEC SQL OPEN CR1   ............................. 3
EXEC SQL WHENEVER NOT FOUND GO TO:FEND;   ....... 4
for(;;){   ...................................... 5
    EXEC SQL FETCH CR1 USING DESCRIPTOR:DAREA   . 5
                    :   ......................... 5,6
}   .............................................. 5
    EXEC SQL WHENEVER NOT FOUND CONTINUE;   ...... 7
FEND:EXEC SQL CLOSE CR1;   ....................... 8
```

Notes

Before the `FETCH` statement is executed, the following information must be set in `DAREA`:

- Size of `SQLVAR` array (`SQLN`)
- Number of areas to receive retrieval results (`SQLD`): executing a `DESCRIBE` statement sets this value
- Data type of receive area (`SQLCOD`): executing a `DESCRIBE` statement sets this value
- Data length of receive area (`SQLLEN`): executing a `DESCRIBE` statement sets this value.

Explanation

1. Sets the address of the allocated area in the SQL Descriptor Area (`DAREA`).
2. Declares a cursor (`CR1`) for the SQL statement identifier (`ST1`).
3. Opens the cursor (`CR1`).
4. Specifies the action to be taken (branching to `FEND`) at the termination of retrieval.
5. Advances the cursor (`CR1`) to the next line, and fetches that line into the area specified by the SQL Descriptor Area (`DAREA`).
6. Specifies the processing to be performed on the retrieval result (e.g., editing and output).
7. Invalidates the action at the termination of retrieval.
8. Closes the cursor (`CR1`).

(d) Dynamic allocation of a data area for specifying ? parameter values

This is an example of inserting data into a dynamically specified table. The items in italics in the figure are explained as follows.

```
char TNAME[30];   ........................................ 1
scanf("%S",TNAME);   ................................ 2
XCMND.cmd_len=(long*)sprint(XCMND.cmd_data,
                    "SELECT * FROM %S",TNAME);   ..... 3

EXEC SQL PREPARE ST1 FROM:XCMND;   .................. 3

EXEC SQL DESCRIBE ST1 INTO:DAREA;   ................. 3
         :  ......................................... 4
for(n=0;n<DAREA.sqld;n++){   ........................ 5
DAREA.SQLVAR[n].sqldata=(unsigned char *)&(X_INT_DATA[n]);
............................... 5
DAREA.SQLVAR[n].sqlind=&(X_IND[n]);   ............... 5
}   ................................................. 3
XCMND.cmd_len=(long*)sprit(XCMND.cmd_data,
              "INSERT INTO %S VALUES(?,...,?)",TNAME);
.......................... 6
 EXEC SQL PREPARE ST2 FROM:XCMND;   ................. 7
for(;;){   .......................................... 8
   [Input insertion data (branched to IEND if there is no data)];   ..... 8
   [Insert data in the data area and the indicator variable area];   ..... 8
  EXEC SQL EXECUTE ST2 USING DESCRIPTOR:DAREA;   ..... 8
}   ................................................. 8
IEND:
```

* In 64-bit mode, this is `int`.

Explanation

1. Declares the variable (TNAME) that stores the table name.
2. Loads the table name from the input data into the variable (TNAME).
3. Uses the DESCRIBE statement to set the columns count of the table specified in 2 data type, data length, and maximum elements count of each column, as the number of ? parameters, the data type, data length, and maximum elements count of the data area for each ? parameter, respectively, in the SQL Descriptor Area (DAREA).
4. Allocates data area for each ? parameter.
5. Sets the address of the allocated area in the SQL Descriptor Area (DAREA).
6. Creates an INSERT statement for inserting data into the specified table.
7. Preprocesses the INSERT statement in XCMND and assigns the SQL statement identifier (ST2).
8. Repeats data insertion on a row basis, setting in the data area, and execution using the EXECUTE statement, as long as data to be inserted exists.

(e) Retrieving DECIMAL data using a FETCH statement

In this example, DECIMAL data is retrieved using a FETCH statement.

1. Declares a data area and an indicator variable.

```
EXEC  SQL  BEGIN  DECLARE  SECTION ;

   SQL TYPE IS DECIMAL(20,0) xdec1 ;      /* Data area      */
   short                     xdec1_i ;    /* Indicator variable  */

EXEC  SQL  END  DECLARE  SECTION ;
```

2. Sets an SQL Descriptor Area.

```
PDSQLCOD(usrsqlda, 2)=PDSQL_DECIMAL_I  ;  /* Sets a data code  */

  PDSQLPRCSN(usrsqlda, 2)=20           ;  /* Sets precision      */
  PDSQLSCALE(usrsqlda, 2)= 0           ;  /* Sets a scale     */
  PDSQLDATA(usrsqlda, 2)=(void*)xdec1  ;  /* Embedded variable
address*/
                                          /* Setting          */
  PDSQLXDIM(usrsqlda, 2)=1;              /* Not a repetition column  */

PDSQLIND(usrsqlda, 2)=(void*)&xdec1_i ; /* Indicator variable address
*/
                                          /* Setting          */
```

(3) SQL Descriptor Area expansion

Table B-4 shows the procedure for expanding the SQL Descriptor Area.

Table B-4: SQL Descriptor Area expansion procedure

Language	When include file is used	When directly coded by user
C	`#include <pdbsqlda.h> PDUSRSQLDA(`*n*`) usrsqlda;`	Expansion of SQL Descriptor Area is coded directly.
COBOL	`COPY SQLDA [ REPLACING 255 BY `*n*` ].`	Expansion of SQL Descriptor Area is coded directly. Level 01 must always be specified first.

Following is a COBOL coding example in which parameters are specified in the SQL Descriptor Area:

```
EXEC  SQL
    BEGIN  DECLARE  SECTION
END   EXEC
01  IN-CHR1  PIC  X(15).
01  IN-IND1  PIC  S9(4)  COMP.
EXEC  SQL
    END  DECLARE  SECTION
END-EXEC
COPY  SQLDA.
         :
COMPUTE  USQLDABC=32
COMPUTE  USQLN=1
COMPUTE  USQLD=1
COMPUTE  USQLDATA(1)=FUNCTION  ADDR(IN-CHR1)
MOVE  SQLCNST0    TO USQLDIM(1)
MOVE  SQLDCOD197  TO USQLCOD(1)
COMPUTE  USQLXDIM(1)=1
COMPUTE  USQLLEN(1)=15
COMPUTE  USQLIND(1)=FUNCTION  ADDR(IN-INT1)
EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :USQLDA
END-EXEC
```

(4) SQL Descriptor Area operation macros

Various macros for declaring the `SQLDA` and for setting/referencing values are defined in C language. These macros can be used by including the unique header file (`pdbsqlda.h`) in the UAP. Table B-5 shows the SQL Descriptor Area operation macros, and Table B-6 shows the macros for specifying data types.

Table B-5: SQL Descriptor Area operation macros

Macro	Function
PDUSRSQLD(*m*)	Declares a user SQLDA.
PDSETSIZE(usrsqlda,*m*)	Specifies the SQLDA size.
PDSQLN(usrsqlda)	Specifies the ? parameter.
PDSQLD(usrsqlda)	Specifies/references the ? parameter and the number of retrieval items.
PDSQLCOD(usrsqlda,*n*)	Specifies/references the data code.
PDSQLLEN(usrsqlda,*n*)	Specifies/references the data length (other than BLOB and decimal number).
PDSQPRCSN(usrsqlda,*n*)	Specifies/references the precision (decimal number only).
PDSQLSCALE(usrsqlda,*n*)	Specifies/references the scale (decimal number only).
PDSQLDATA(usrsqlda,*n*)	Specifies the address of the data area.
PDSQLIND(usrsqlda,*n*)	Specifies an indicator variable address.
PDSQLLOBLEN(usrsqlda,*n*)	Specifies/references the BLOB data length.
PDSQLDIM(usrsqldata,*n*)	Specifies/references the value in unused area.
PDSQLXDIM(usrsqldata,*n*)	Specifies/references the maximum number of elements for the repetition structure.
PDSQLSYS(usrsqldata,*n*)	Specifies the length of one element that includes the gap in variable-length character string type for the repetition structure or array structure.

Legend:

usrsqlda: User-defined SQL Descriptor Area name; any name can be specified.

m: Number of ? parameters (1-30000).

n: Number of ? parameters to be specified or referenced (0-29999).

Table B-6: Macros for specifying data types

Macro	Indicator variable	Corresponding data type
PDSQL_FLOAT PDSQL_FLOAT_I	No Yes	FLOAT
PDSQL_SMALLFLT PDSQL_SMALLFLT_I	No Yes	SMALLFLT

Macro	Indicator variable	Corresponding data type
PDSQL_DECIMAL PDSQL_DECIMAL_I	No Yes	DECIMAL
PDSQL_INTEGER PDSQL_INTEGER_I	No Yes	INTEGER
PDSQL_SMALLINT PDSQL_SMALLINT_I	No Yes	SMALLINT
PDSQL_VARCHAR PDSQL_VARCHAR_I	No Yes	VARCHAR
PDSQL_CHAR PDSQL_CHAR_I	No Yes	CHAR
PDSQL_NVARCHAR PDSQL_NVARCHAR_I	No Yes	NVARCHAR
PDSQL_NCHAR PDSQL_NCHAR_I	No Yes	NCHAR
PDSQL_MVARCHAR PDSQL_MVARCHAR_I	No Yes	MVARCHAR
PDSQL_MCHAR PDSQL_MCHAR_I	No Yes	MCHAR
PDSQL_DATE PDSQL_DATE_I	No Yes	DATE
PDSQL_TIME PDSQL_TIME_I	No Yes	TIME
PDSQL_YEARTODAY PDSQL_YEARTODAY_I	No Yes	INTERVAL YEAR TO DAY
PDSQL_HOURTOSEC PDSQL_HOURTOSEC_I	No Yes	INTERVAL HOUR TO SECOND
PDSQL_ROW PDSQL_ROW_I	No Yes	ROW
PDSQL_BLOB PDSQL_BLOB_I	No Yes	BLOB
PDSQL_TIMESTAMP PDSQL_TIMESTAMP_I	No Yes	TIMESTAMP
PDSQL_BINARY PDSQL_BINARY_I	No Yes	BINARY

Macro	Indicator variable	Corresponding data type
PDSQL_BLOB_LOC PDSQL_BLOB_LOC_I	No Yes	BLOB locator
PDSQL_BINARY_LOC PDSQL_BINARY_LOC_I	No Yes	BINARY locator
PDSQL_CVARCHAR PDSQL_CVARCHAR_I	No Yes	VARCHAR for C

Following is a C coding example in which parameters are specified in the SQL Descriptor Area:

```
#include <pdbsqlda.h>                        /* Includes header file. */

EXEC  SQL  BEGIN  DECLARE  SECTION ;
short  xint1 ;
char   xchr1[16] ;
EXEC  SQL  END  DECLARE  SECTION ;
PDUSRSQLDA(2)  usrsqlda ;                    /* Declares SQL Descriptor
                                            Area. */
          :
ClearSqlda(2);                              /* Clears SQL Descriptor Area */
PDSQLCOD(usrsqlda, 0)=PDSQL_SMALLINT ;      /* Sets data code.          */
PDSQLLEN(usrsqlda, 0)=sizeof(short) ;       /* Sets data code.          */
PDSQLDATA(usrsqlda, 0)=(void*)&xint ;       /* Sets embedded variable
                                            address.   */
PDSQLIND(usrsqlda, 0)=NULL ;                /* Sets indicator variable
                                            address. */
PDSQLCOD(usrsqlda, 1)=PDSQL_CHAR ;          /* Sets data code. */
PDSQLLEN(usrsqlda, 1)=sizeof(xchar)-1 ;/* Sets data code. */
PDSQLDATA(usrsqlda, 1)=(void*)xchr ;        /* Sets embedded variable
                                            address.   */
PDSQLIND(usrsqlda, 1)=NULL ;                /* Sets indicator variable
                                            address.  */

EXEC  SQL
    EXECUTE  ST1  USING  DESCRIPTOR  :usrsqlda ;
```

(5) Expansion format of repetition columns

During compilation, embedded variables in a repetition column are expanded into the structures shown in Table B-7 based on macro definition. The explanation here applies to the C language.

The macro for manipulating a repetition column uses the members of the expanded structures to reference the elements of the repetition column.

If the user wishes to directly set an address in the SQL Descriptor Area by securing an area, the area must be assigned to a language boundary. `FLOAT ARRAY` explicitly includes a free area for language boundary adjustment. However, when setting an address in the SQL Descriptor Area, you must set it by taking a free area into consideration.

Specify these expansion formats only when adjusting a boundary or determining a size during this type of area allocation. When specifying a repetition column as an embedded variable, do not specify an expansion format. Instead, use the macros described in *E. SQL Data Types and Data Descriptions*.

Table B-7: Repetition column expansion format

<table>
<tr><th>SQL data type</th><th>Macro name</th><th>Expansion format</th></tr>
<tr><td><code>SMALL INT ARRAY[m]</code></td><td><code>PD_MV_SINT(m)</code></td><td><pre>struct {
   long mcnt;
   short data[m];
}</pre></td></tr>
<tr><td><code>INTEGER ARRAY[m]</code></td><td><code>PD_MV_INT(m)</code></td><td><pre>struct{
   long mcnt
   long data[m];
}</pre></td></tr>
<tr><td><code>SMALL FLT ARRAY[m]</code></td><td><code>PD_MV_SFLT(m)</code></td><td><pre>struct {
   long mcnt;
   float data[m];
}</pre></td></tr>
<tr><td><code>FLOAT ARRAY[m]</code></td><td><code>PD_MV_FLT(m)</code></td><td><pre>struct
   union {
     double resv1;
       struct {
         long resv2;
         long mcnt;
     }mcnt_dmy2;
   } mcnt_dmy1;
   double data[m];
}</pre></td></tr>
<tr><td><code>CHAR(n) ARRAY[m]</code></td><td><code>PD_MV_CHAR(m, n)</code></td><td><pre>struct {
   long mcnt;
   char data[m][(n)+1];
}</pre></td></tr>
<tr><td><code>NCHAR(n) ARRAY[m]</code></td><td><code>PD_MV_NCHAR(m, n)</code></td><td><pre>struct {
   long mcnt;
   char data[m][2*(n)+1];
}</pre></td></tr>
</table>

SQL data type	Macro name	Expansion format

VARCHAR(*n*) ARRAY[*m*]	PD_MV_VCHAR(*m*, *n*)	struct {
		   long mcnt;
		   struct {
		     short len;
		     char str[*n*];
		   } data[*m*];
		}

	PD_MV_CVCHAR(*m*, *n*)	struct {
		   long mcnt;
		   char data[*m*][(*n*)+1];
		}

NVARCHAR(*n*) ARRAY[*m*]	PD_MV_NVCHAR(*m*, *n*)	struct {
		   long mcnt;
		   struct {
		     short len;
		     char str[2*(*n*)+1];
		   } data[*m*];
		}

DECIMAL
  [(*p*[,*s*])]ARRAY[*m*]
	PD_MV_DEC(*m*, *p*, *s*)	struct {
		   long mcnt;
		   unsigned char data[*m*][(*p*)/
		2+1];
		}

C. Column Name Descriptor Area

When using the SQL Descriptor Area to receive the following information, you can also receive the column name information and routine parameter information by specifying a Column Name Descriptor Area:

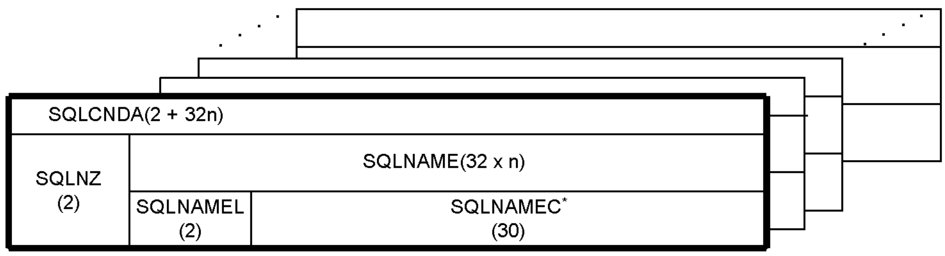
- Retrieval item information (number of retrieval items, as well as each retrieval item's data type, data length, and maximum number of elements)
- `CALL` statement's input/output `?` parameter information (number of `?` parameters, as well as the `?` parameter's data type and data length)

C.1 Organization and contents of the Column Name Descriptor Area

(1) Organization of the Column Name Descriptor Area

Figure C-1 shows how the Column Name Descriptor Area (`SQLCNDA`) is organized.

Figure C-1: Organization of the Column Name Descriptor Area



Note

Numbers in parentheses indicate length (in bytes).

* `SQLNAMEC` is an array of variable-length character strings with a maximum length of 30 bytes. The array should be the same length as the `SQLVAR` array in the SQL Descriptor Area. For details about the size of the `SQLVAR` array, see *B.1 Organization and contents of the SQL Descriptor Area*.

(2) Contents of the Column Name Descriptor Area

Table C-1 shows what the Column Name Descriptor Area contains.

Table C-1: Contents of the Column Name Descriptor Area

<table>
<tr><th>Acquired information item</th><th>Type of retrieval item</th><th>Description</th></tr>
<tr><td rowspan="7">Retrieval item</td><td>Column (without subscript specification)</td><td>column-name</td></tr>
<tr><td>Column (with subscript specification)</td><td>column-name [subscript]</td></tr>
<tr><td>Set function</td><td>△ △ COUNT(*)
{△ {■ |△ } function-name (column-name) |
△ { ■ |△ } function-name
(DISTINCT-column-name) | △ △function-name (△
EXP)}</td></tr>
<tr><td>Window function</td><td>△ △ EXP (integer)</td></tr>
<tr><td>Value expression (including literal)</td><td>△ △ EXP(integer)</td></tr>
<tr><td>WRITE specification</td><td>△ △ EXP(integer)</td></tr>
<tr><td>ROW</td><td>△ △ ROW</td></tr>
<tr><td>CALL statement's input ? parameter</td><td rowspan="2">? parameter</td><td rowspan="2">routine's-parameter-name</td></tr>
<tr><td>CALL statement's output ? parameter</td></tr>
<tr><td>User-defined type configuration element</td><td>Attribute</td><td>attribute-name</td></tr>
</table>

Legend:

■ : X'FF'

△ : One blank

Notes

1. The *i*th element of the Column Name Descriptor Area stores column name information for the *i*th retrieval item.
2. If a retrieval item is a column, the column name is assigned to the retrieval item from the beginning of the retrieval item. If the length of the column name, including subscripts, is greater than 30 bytes, the excess bytes of the column name are truncated.

   If a retrieval item is not a column, one or two blanks (indicated by △ in the table) are set at the beginning of the retrieval item. If the column is greater than 30 bytes, X'FF' is set in byte 2. The symbol ■ in the table denotes the value X'FF'.

3. Integer indicates the ordinal number of a retrieval item.
4. For UNION[ALL] or EXCEPT[ALL], the contents of the retrieval item in the query specified first are set in the Column Name Descriptor Areas.
5. If an AS column name is specified, the specified column name is set.
6. The routine's parameter name is set only when the? parameter is specified independently in the CALL statement's argument. If a value expression including the? parameter is specified, SQLNAMEL is set to 0.
7. If the retrieval item is a column of a derived table, the derived column list is omitted after the derived table, and the derived column has no column name in the query selection expression, ΔNONAME is set.

C.2 Expanding the Column Name Descriptor Area

The Column Name Descriptor Area is allocated as static area by declaring it in the UAP.

(1) C

The following code shows the format of the Column Name Descriptor Area that is to be expanded in the source program when C language is used:

```
struct {
    short        sqlnz;          /* Effective arrays count  */
    struct {
        short    sqlnamel;       /* Effective column name length */
        char     sqlnamec[30];   /* Column name storage area   */
    } SQLNAME[n];1
  }XXXXX;2
```

[1] *n* indicates the same number (1-30000) as the size of the SQLVAR array in the SQL Descriptor Area.

[2] Any desired character string can be specified as the structure name (*XXXXX* portion), except that a character string beginning with SQL cannot be specified. When Column Name Descriptor Areas are specified using a DESCRIBE statement, the name of the allocated areas must be specified.

(2) COBOL

The following code shows the format of the Column Name Descriptor Area that is to be expanded in the source program when COBOL is used:

```
01  SQLCNDA. 1
  02  SQLNZ  PIC  S9(4) COMP.
```

```
02  SQLNAME  OCCURS  1  TIMES  n. 2
  03  SQLNAMEL  PIC  S9(4)  COMP.
  03  SQLNAMEC  PIC  X(30).
```

[1] Any name can be specified as the name of the set item (`SQLCNDA` area); however, a character string that begins with `SQL` cannot be used for a data item. In addition, the set item level must always be set to `01`.

[2] *n* indicates the required number (1-30000).

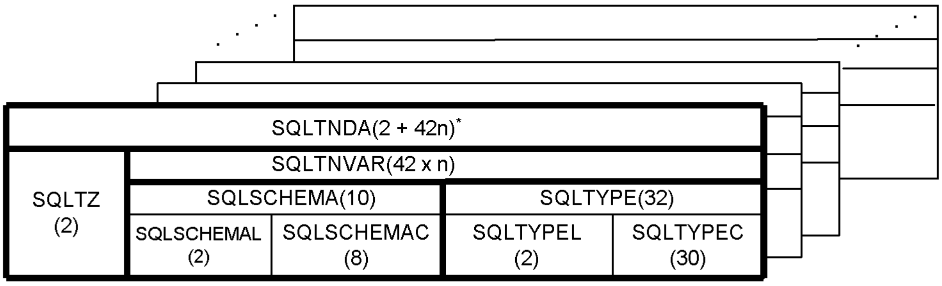
D. Type Name Descriptor Area

When the SQL Descriptor Area is used to receive retrieval item information and user-defined type definition information, user-defined type data type names can also be received by specifying a Type Name Descriptor Area (`SQLTNDA`).

D.1 Organization of the Type Name Descriptor Area

Figure D-1 shows how the Type Name Descriptor Area (`SQLTNDA`) is organized.

Figure D-1: Organization of the Type Name Descriptor Area



Note

Parentheses enclose a length in bytes.

* `SQLTNVAR` is an array of a structure composed of a variable length character string `SQLSCHEMA` with a maximum length of 10 bytes, and a variable-length character string `SQLTYPE` with a maximum length of 32 bytes. The array should be the same length as the `SQLVAR` array in the SQL Descriptor Area. For details about the size of the `SQLVAR` array, see *B.1 Organization and contents of the SQL Descriptor Area*.

D.2 Contents of the Type Name Descriptor Area

Table D-1 shows what the Type Name Descriptor Area contains.

Table D-1: Contents of the Type Name Descriptor Area

Level number*	Type name area name	Data type	Length (bytes)	Description
1	`SQLTNDA`	—	$2+42 \times n$	Indicates the name of the entire Type Name Descriptor Area.
2	`SQLTZ`	`short`	2	Specifies the number of retrieval items.
2	`SQLTNVAR`	—	$42 \times n$	Area composed of the authorization identifier and data type identifiers.

Level number*	Type name area name	Data type	Length (bytes)	Description
3	SQLSCHEMA	—	10	Area storing information about the user-defined type authorization identifier.
4	SQLSCHEMAL	short	2	The authorization identifier is set in this area. 0 is set if the data type of the corresponding retrieval item is not a user-defined type.
4	SQLSCHEMAC	char	8	The authorization identifier is set in this area.
3	SQLTYPE	—	32	Area storing information about the user-defined type data type identifier.
4	SQLTYPEL	short	2	The length of the user-defined type is set in this area. 0 is set if a data type of the corresponding retrieval item is not the user-defined type.
4	SQLTYPEC	char	30	The data type identifier of the user-defined type is set in this area.

Legend:

— : Not applicable

* Level numbers indicate the set inclusion relationships of the Type Name Descriptor Area. For example, the level 2 data area is composed of the level 3 data areas.

D.3 Expanding the Type Name Descriptor Area

The Type Name Descriptor Area is allocated by declaring it in the UAP.

(1) C

The following code shows the format of the Type Name Descriptor Area that is to be expanded in the source program when C is used.

```
struct {
    short       sqlnz;           /* Effective array count */
    struct {
      struct {
        short   sqlchemal;       /* Effective authorization identifier length */
        char    sqlschemac[8];   /* Authorization identifier storage area */
     } sqlschema;
      struct {
        short   sqltypel;        /* Effective length of user-defined type
                                    name */
        char    sqltypec[30];    /* User-defined type name storage area */
    } sqltnvar[n];1
} Usrsqltnda;2
```

[1] *n* indicates the same number (1-30000) as the size of the `SQLVAR` array in the SQL Descriptor Area.

[2] Any desired character string can be specified as the structure name (`usrsqltnda` portion), except that a character string beginning with `SQL` cannot be specified. When the Type Name Descriptor Area is specified using a `DESCRIBE` statement, the name of the allocated area must be specified.

(2) COBOL

The following code shows the format of the Type Name Descriptor Area that is to be expanded in the source program when COBOL is used.

```
01  USQLTNDA.1
  02  USQLTZ  PIC  S9(4) COMP.
  02  USQLTNVAR  OCCURS  1  TIMES  n.2
    03  USQLSCHEMA.
      04  USQLSCHEMAL  PIC  S9(4)  COMP.
      04  USQLSCHEMAC  PIC  X(8).
    03  USQLTYPE.
      04  USQLTYPEL  PIC  S9(4)  COMP.
      04  USQLTYPEC  PIC  X(30).
```

[1] Any name can be specified as the name of the set item (`USQLTNDA` portion); however, a character string that begins with `SQL` cannot be used for a data item.

[2] *n* indicates the same number (1-30000) as the size of the `SQLVAR` array in the SQL Descriptor Area.

E. SQL Data Types and Data Descriptions

This appendix shows the correspondence between SQL data types and C or COBOL data descriptions.

E.1 SQL data types and C data descriptions

This section provides the correspondence between SQL data types and C data descriptions. Data can be exchanged between variables of compatible data types and between variables of either convertible or assignable data types.

Table E-1 shows how SQL data types relate to C data descriptions. Table E-2 shows how SQL data types relate to C data descriptions when arrays are used. Table E-3 shows SQL data types and C data descriptions when repetition columns are used.

Table E-1: SQL data types and C data descriptions

SQL data type	C data description	Remarks
`SMALLINT`	`short` *variable-name*`;`	—
`INTEGER`	`long` *variable-name*`;`	—
`DECIMAL [(`*p*`[,`*s*`])]`	`SQL TYPE IS DECIMAL(`*p*`,`*s*`)` *variable-name*`;`[5]	$1 \leq p \leq 29, 0 \leq s \leq$ p
`SMALLFLT, REAL`	`float` *variable-name*`;`	—
`FLOAT (DOUBLE PRECISION)`	`double` *variable-name*`;`	—
`CHAR [(`*n*`)]`	`char` *variable-name*`[`*n*`+1];`[1]	$1 \leq n \leq 30000$
`VARCHAR(`*n*`)`	`struct {` `short` *variable-name-1*`;` `char` *variable-name-2*`[`*n*`];` `}` *structure-name*`;`	$1 \leq n \leq 32000$
	`SQL TYPE IS VARCHAR(`*n*`)` *variable-name*`;`[6]	
	`VARCHAR` *variable-name*`[`*n*`+1]`[9]	
`NCHAR [(`*n*`)]`	`char` *variable-name-2*`[2`*n*`+1];`[1]	$1 \leq n \leq 15000$

SQL data type	C data description	Remarks

`NVARCHAR(`*n*`)`	`struct {`
	`short` *variable-name-1*`;`
	`char` *variable-name-2*`[2`*n*`];`
	`}` *structure-name*`;`
		1 ≤ *n* ≤ 16000

	`SQL TYPE IS`
	`NVARCHAR(`*n*`)`
	variable-name`;`[6]	

`MCHAR[(`*n*`)]`	`char` *variable-name*`[`*n*`+1];`[1]	1 ≤ *n* ≤ 30000

`MVARCHAR(`*n*`)`	`struct {`
	`short` *variable-name-1*`;`
	`char` *variable-name-2*`[`*n*`];`
	`}` *structure-name*`;`
		1 ≤ *n* ≤ 32000

	`SQL TYPE IS`
	`MVARCHAR(`*n*`)`
	variable-name`;`[6]	

`DATE`	`char` *variable-name*`[11];`[2]	—
`TIME`	`char` *variable-name*`[9];`[2]	—
`INTERVAL YEAR TO DAY`	`SQL TYPE IS DECIMAL(8,0)` *variable-name*`;`[5]	—
`INTERVAL HOUR TO SECOND`	`SQL TYPE IS DECIMAL(6,0)` *variable-name*`;`[5]	—
`TIMESTAMP[(`*p*`)]`	`char` *variable-name*`[`*n* `+ 1];`[2]	If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
`ROW`[3]	`char` *variable-name*`[`*n* `+ 1];`	1 ≤ total length ≤ 30000
`BLOB`	`SQL TYPE IS BLOB(`*n*`[K|M|G])]` *variable-name*`;`[4]	Default: 1 ≤ *n* ≤ 2147483647 In units of K: 1 ≤ *n* ≤ 2097152 In units of M: 1 ≤ *n* ≤ 2048 In units of G: 1 ≤ *n* ≤ 2

SQL data type		C data description	Remarks

`BINARY(`*n*`)`		`struct {`
		`long` *variable-name-1*`;`
		`char` *variable-name-2*`[`*n*`];`
		`}` *structure-name*`;`
			$1 \leq n \leq 2147483647$

		`SQL TYPE IS BINARY(`*n*`)` *variable-name*`;`[7]	

`BLOB` locator		`SQL TYPE IS`
		`BLOB AS LOCATOR`
		variable-name[8]	

`BINARY` locator		`SQL TYPE IS`
		`BINARY AS LOCATOR`
		variable-name[8]	

Indicator variable	Other than `BLOB`, `BINARY`, `BLOB` locator, or `BINARY` locator	`short` *variable-name*`;`	
	`BLOB`, `BINARY`, `BLOB` locator, or `BINARY` locator	`long` *variable-name*`;`	

SQL statement		`struct {`
		`long` *variable-name-1*`;`
		`char` *variable-name-2*`[`*n*`];`
		`}` *structure-name*`;`
			$1 \leq n \leq 2000000$

Legend:

—: Cannot be coded.

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

[1] The following rules govern data conversion between SQL data types (`CHAR(`*n*`)`, `NCHAR(`*n*`)`, and `MCHAR(`*n*`)`) and C-language data types (`char[`*n*`+1]`, `char[2`*n*`+1]`, and `char[2`*n*`+1]`):

- For input (conversion from `char[`*n*`+1]` to `CHAR(`*n*`)`, conversion from `char[2`*n*`+1]` to `NCHAR(`*n*`)`, or conversion from `char[`*n*`+1]` to `MCHAR(`*n*`)`)

  The length of a fixed-length character string received by HiRDB from a C language-character string is equal to the length from the beginning of the character string to one character before the null character. If no null character is found in *n*`+1` array elements, the length is defined as *n*.

- For output (conversion from `CHAR(`*n*`)` to `char[`*n*`+1]`, conversion from `NCHAR(`*n*`)` to `char[2`*n*`+1]`, or conversion from `MCHAR(`*n*`)` to `char[`*n*`+1]`)

  A null character is appended at the end of the character string; therefore, the length of the character string known to the UAP is the SQL character string length + 1.

[2] When retrieving date data (`DATE`) using a dynamic SQL, the data code for the retrieval item information obtained by the `DESCRIBE` statement must be set as the character data type with a data length of at least 10 bytes. Similarly, when retrieving time data using a dynamic SQL, the data code for the retrieval item information obtained by the `DESCRIBE` statement must be set as the character data type with a data length of at least 8 bytes.

To retrieve time stamp data (`TIMESTAMP`) using a dynamic SQL statement, specify the following:

- Set the data code for the retrieval item information obtained using a `DESCRIBE` statement to the character data type.
- If *p* is 0, set the data size to 19 bytes or greater. If *p* is 2, 4, or 6, set the data size to 20 + *p* bytes or greater.

[3] Operations involving the `ROW` type are allowed only when the HiRDB server and the HiRDB client use the same endian type.

[4] The coding of a BLOB UAP is expanded internally as follows:

```
struct{
      long          variable-name_reserved;          1
      unsigned long variable-name_length;          2
      char          variable-name_data[m];          3
} variable-name
```

1. *variable-name*`_reserved` is not used. In the 64-bit mode, `int` *variable-name*`_reserved;` is used.
2. *variable-name*`_length` indicates the actual `BLOB` size. In the 64-bit mode,

`unsigned int` *variable-name*`_length;` is used.

3. *variable-name*`_data[`*m*`]` is the BLOB data storage area (where *m* denotes the actual data length).

5 The code for a DECIMAL UAP is internally expanded as follows:

`unsigned char` *variable-name*`[`↓*p*`/2`↓`+1];`

One byte of DECIMAL data expresses two numeric digits. A sign is expressed by four low-order bits of the trailing byte. Therefore, for DECIMAL data consisting of an even number of digits, four high-order bits of the leading byte must be padded with 0s. Do not use any number other than 0 for padding purposes.

The following table shows the standard sign representation; for details about the sign for DECIMAL type used with HiRDB, see the manual *HiRDB Version 8 SQL Reference*.

Sign in hexadecimal representation	Description
`X'C'`	Treated as a positive sign. Positive numbers include 0.
`X'D'`	Treated as a negative sign.

Coding examples

`123.4567` (odd number of digits)

```
unsigned char ex1[4]={0x12,0x34,0x56,0x7c};
```

`-123.456` (even number of digits)

```
unsigned char ex2[4]={0x01,0x23,0x45,0x6d};
```

`0` (odd number of digits)

```
unsigned char ex3[1]={0x0c};
```

6 The following internal expansion takes place:

```
struct{
         short   len;
         char    str[n];
}variable-name
```

For `NVARCHAR`, `str[2n]` is used.

7 The following internal expansion takes place:

```
struct{
```

```
        long  len;
        char  str[n];
}variable-name
```

In 64-bit mode, `long len;` is replaced with `int len;`.

[8] The following internal expansion takes place:
`unsigned long` *variable-name*`;`

In 64-bit mode, `unsigned long` *variable-name*`;` is replaced with `unsigned int` *variable-name*`;`.

[9] The following internal expansion takes place:
`char` *variable-name*`[`*n*`+1];`

The character string length is the length from the beginning of the string to the character before the `NULL` character. When a character string in C is accepted, an error occurs if there is no `NULL` character in the n+1-th array element.

Table E-2: SQL data types and C data descriptions when arrays are used

SQL data type	C data description	Remarks
SMALLINT	short *variable-name*[*m*];	—
INTEGER	long *variable-name*[*m*];	—
DECIMAL[(*p*[,*s*])]	SQL TYPE IS DECIMAL(*p*,*s*) *variable-name*[*m*];	$1 \leq p \leq 29, 0 \leq s \leq p$
SMALLFLT, REAL	float *variable-name*[*m*];	—
FLOAT (DOUBLE PRECISION)	double *variable-name*[*m*];	—
CHAR[(*n*)]	char *variable-name*[*m*][*n*+1];	$1 \leq n \leq 30000$
VARCHAR(*n*)	struct { short *variable-name-1*; char *variable-name-2*[*n*]; } *structure-name*[*m*];	$1 \leq n \leq 32000$
	SQL TYPE IS VARCHAR(*n*) *variable-name*[*m*];	—
	VARCHAR *variable-name*[*m*][*n*+1];	
NCHAR[(*n*)]	char *variable-name*[*m*][2*n*+1];	$1 \leq n \leq 15000$

SQL data type	C data description	Remarks
NVARCHAR[(*n*)]	struct { short *variable-name-1*; char *variable-name-2*[2*n*]; } *structure-name*[*m*];	$1 \leq n \leq 16000$
	SQL TYPE IS NVARCHAR(*n*) *variable-name*[*m*];	—
MCHAR(*n*)	char *variable-name*[*m*][*n*+1];	$1 \leq n \leq 30000$
MVARCHAR(*n*)	struct { short *variable-name-1*; char *variable-name-2*[*n*]; } *structure-name*[*m*];	$1 \leq n \leq 32000$
	SQL TYPE IS MVARCHAR(*n*) *variable-name*[*m*];	
DATE	char *variable-name*[*m*][11];	—
TIME	char *variable-name*[*m*][9];	—
TIMESTAMP[(*p*)]	char *variable-name*[*m*][*n* + 1];	If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0) *variable-name*[*m*];	—
INTERVAL HOUR TO SECOND	SQL TYPE IS DECIMAL(6,0) *variable-name*[*m*];	—
ROW	char *variable-name*[*m*][*n*+1];	$1 \leq n \leq 30000$
BLOB	CN	—
BINARY	struct { long *variable-name-1*; char *variable-name-2*[*n*]; } *structure-name*[*m*];	• FETCH that uses an array $4 \leq n \leq 2147483644$ (*n* must be a multiple of 4.) • Other than FETCH that uses an array4 $\leq n \leq 32000$ (*n* must be a multiple of 4.)
	SQL TYPE IS BINARY(*n*) *variable-name*[*m*];	
BLOB locator	—	
BINARY locator	SQL TYPE IS BINARY AS LOCATOR *variable-name*[*m*];	

SQL data type		C data description	Remarks
Indicator variable	Other than BINARY or BINARY locator	`short` *variable-name*[*m*];	—
	BINARY or BINARY locator	`long` *variable-name*[*m*];	—
SQL statement		CN	—

Legend:

CN: Cannot be coded.

m: Number of array elements (1-4096)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

Table E-3: SQL data types and C data descriptions when repetition columns are used

SQL data type	C data description	Remarks
`SMALLINT`	`PD_MV_SINT(`*m*`)` *variable-name*`;`	—
`INTEGER`	`PD_MV_INT(`*m*`)` *variable-name*`;`	—
`DECIMAL`	`PD_MV_DEC(`*m*`,`*p*`,`*s*`)` *variable-name*`;`	$1 \leq p \leq 29$, $0 \leq s \leq p$
`SMALLFLT`, `REAL`	`PD_MV_SFLT(`*m*`)` *variable-name*`;`	—
`FLOAT (DOUBLE PRECISION)`	`PD_MV_FLT(`*m*`)` *variable-name*`;`	—
`CHAR[(`*n*`)]`	`PD_MV_CHAR(`*m*`,`*n*`)` *variable-name*`;`	$1 \leq n \leq 30000$

SQL data type	C data description	Remarks
VARCHAR(*n*)	PD_MV_VCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 32000
	PD_MV_CVCHAR(*m*,n) *variable-name*;	
NCHAR[(*n*)]	PD_MV_NCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 15000
NVARCHAR[(*n*)]	PD_MV_NVCHAR(*m*,*n*) *variable-name*;	1 ≤ *n* ≤ 16000
MCHAR(*n*)	PD_MV_CHAR(*m*, *n*) *variable-name*;	1 ≤ *n* ≤ 30000
MVARCHAR(*n*)	PD_MV_CHAR(*m*, *n*) *variable-name*;	1 ≤ *n* ≤ 32000
DATE	PD_MV_CHAR(*m*,10) *variable-name*;	--
TIME	PD_MV_CHAR(*m*,8) *variable-name*;	--
TIMESTAMP[(*p*)]	PD_MV_CHAR(*m*,*n*) *variable-name*;	If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	PD_MV_DEC(*m*,8,0) *variable-name*;	--
INTERVAL HOUR TO SECOND	PD_MV_DEC(*m*,6,0) *variable-name*;	--
ROW	CN	--
BLOB	CN	--
BINARY	CN	--
Indicator variable (other than BLOB, BINARY, BLOB locator, or BINARY locator)	PD_MV_SINT(*m*) *variable-name*;	--
SQL statement	CN	--

Legend:

CN: Cannot be coded.

m: Maximum number of repetition array elements (2-30000).

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Special macros for referencing or setting embedded variables for each data type are used in the SQL data type and C data description when repetition columns are used. Table E-4 shows the macros for referencing or setting embedded variables.

Table E-4: Macros for referencing or setting embedded variables

SQL data type	Macro name	Data to be referenced or set	Data type
SMALLINT	PD_MV_SINT_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_SINT_DATA (*variable-name, m*)	Each repetition element	short
INTEGER	PD_MV_INT_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_INT_DATA (*variable-name, m*)	Each repetition element	long*
DECIMAL[(*p*[,*s*])]	PD_MV_DEC_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_DEC_DATA (*variable-name*, *m*)	Start address of each repetition element in decimal	unsigned
SMALLFLT, REAL	PD_MV_SFLT_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_SFLT_DATA (*variable-name, m*)	Each repetition element	float
FLOAT (DOUBLE PRECISION)	PD_MV_FLT_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_FLT_DATA (*variable-name, m*)	Each repetition element	double
CHAR[(*n*)]	PD_MV_CHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_CHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]

SQL data type	Macro name	Data to be referenced or set	Data type
VARCHAR(*n*)	PD_MV_VCHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_VCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHAR_STR (*variable-name*)	Address of character string of each repetition element	char[ ]
	PD_MV_CVCHAR_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_CVCHAR_DATA(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
NCHAR[(*n*)]	PD_MV_NCHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_NCHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
NVARCHAR[(*n*)]	PD_MV_NVCHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_NVCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_NVCHAR_STR (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
MCHAR(*n*)	PD_MV_CHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_CHAR_DATA (*variable-name, m*)	Leading address of character string of each repetition element	char[ ]
MVARCHAR(*n*)	PD_MV_VCHAR_CNT (*variable-name*)	Current repetition data element count	long*
	PD_MV_VCHAR_LEN (*variable-name, m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHAR_STR (*variable-name, m*)	Address of character string of each repetition element	char[ ]
DATE	Same as CHAR(10)	—	—
TIME	Same as CHAR(8)	—	—

SQL data type	Macro name	Data to be referenced or set	Data type
TIMESTAMP[(*p*)]	Same as CHAR(*n*) If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.	—	—
INTERVAL YEAR TO DAY	Same as DECIMAL(8,0)	—	—
INTERVAL HOUR TO SECOND	Same as DECIMAL(6,0)	—	—
Indicator variable	PD_MV_SINT_CNT (*variable-name*)	Indicator of the overall repetition column	long*
	PD_MV_SINT_DATA (*variable-name, m*)	Indicator of each repetition column element	short

Legend:

— : Not applicable

m: Number of each repetition column element (0 - *m*-1).

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

* In 64-bit mode, the data type is int.

The following shows an example of macros used for referencing or setting embedded variables in repetition columns:

```
EXEC SQL BEGIN DECLARE SECTION;
char xname[5];
PD_MV_SINT(4) xmscore;
PD_MV_CHAR(4,5) xmsubject;
EXEC SQL END DECLARE SECTION;
     :
strcpy(xname,"SMITH")
PD_MV_SINT_DATA(xmscore,0)=90;
PD_MV_SINT_DATA(xmscore,1)=65;
PD_MV_SINT_DATA(xmscore,2)=85;
PD_MV_SINT_DATA(xmscore,3)=55;
PD_MV_SINT_CNT(xmscore)=4;
strcpy(PD_MV_CHAR_DATA(xmsubject,0),"MATHEMATICS");
strcpy(PD_MV_CHAR_DATA(xmsubject,1),"ENGLISH");
strcpy(PD_MV_CHAR_DATA(xmsubject,2),"SCIENCE");
```

```
strcpy(PD_MV_CHAR_DATA(xmsubject,3),"SOCIAL STUDIES");
PD_MV_CHAR_CNT(xmsubject)=4;
EXEC SQL
  INSERT INTO
SCORE_TABLE(NAME,SUBJECT,SCORE))VALUES(:xname,
:xmsubject;:xmscore);
```

Table E-5 shows pointer variables and the C language data description.

Table E-5: Pointer variables and C language data description

SQL data type	C language data description	Remarks
`SMALLINT`	`short *`*variable-name*`;`	—
`INTEGER`	`long *`*variable-name*`;`	—
`DECIMAL[(`*p*`[,`*s*`])]`	`SQL TYPE IS DECIMAL(`*p*`,`*s*`)` `*`*variable-name*`;`	$1 \leq p \leq 29, 0 \leq s \leq p$
`SMALLFLT`, `REAL`	`float *`*variable-name*`;`	—
`FLOAT` `(DOUBLE PRECISION)`	`double *`*variable-name*`;`	—
`CHAR[(`*n*`)]`	`char *`*variable-name*`;`	$1 \leq n \leq 30000$*
`VARCHAR(`*n*`)`	`struct {` `short` *variable-name-1*`;` `char` *variable-name-2*`[`*n*`];` `} *`*structure-name*`;`	$1 \leq n \leq 32000$
	`SQL TYPE IS` `VARCHAR(`*n*`)` `*`*variable-name*`;`	
	`VARCHAR *`*variable-name*`;`*	
`NCHAR[(`*n*`)]`	`char *`*variable-name*`;`	$1 \leq n \leq 15000$*
`NVARCHAR(`*n*`)`	`struct {` `short` *variable-name-1*`;` `char` *variable-name-2*`[2`*n*`];` `} *`*structure-name*`;`	$1 \leq n \leq 16000$
	`SQL TYPE IS` `NVARCHAR(`*n*`)` `*`*variable-name*`;`	
`MCHAR[(`*n*`)]`	`char *`*variable-name*`;`	$1 \leq n \leq 30000$*

SQL data type	C language data description	Remarks
MVARCHAR(*n*)	struct { short *variable-name-1*; char *variable-name-2*[*n*]; } **structure-name*;	$1 \leq n \leq 32000$
	SQL TYPE IS MVARCHAR(*n*) **variable-name*;	
DATE*	char **variable-name*;	—
TIME*	char **variable-name*;	—
TIMESTAMP*	char **variable-name*;	—
INTERVAL YEAR TO DAY	SQL TYPE IS DECIMAL(8,0) **variable-name*;	—
INTERVAL HOUR TO SECOND	SQL TYPE IS DECIMAL(6,0) **variable-name*;	—
ROW	char **variable-name*;	1 ≤ *total-length* ≤ 30000*
BLOB	SQL TYPE IS BLOB(*n*[{K|M|G}]) **variable-name*;	Default: $1 \leq n \leq 2147483647$ In units of K: $1 \leq n \leq 2097152$ In units of M: $1 \leq n \leq 2048$ In units of G: $1 \leq n \leq 2$
BINARY(*n*)	struct { long *variable-name-1*; char *variable-name-2*[*n*]; } **structure-name*;	$1 \leq n \leq 2147483647$
	SQL TYPE IS BINARY(*n*) **variable-name*;	
BLOB locator	SQL TYPE IS BLOB AS LOCATOR **variable-name*;	
BINARY locator	SQL TYPE IS BINARY AS LOCATOR **variable-name*;	

SQL data type		C language data description	Remarks
Indicator variable	Other than BLOB, BINARY, BLOB locator, or BINARY locator	`short *`*variable-name*`;`	
	BLOB, BINARY, BLOB locator, or BINARY locator	`long *`*variable-name*`;`	

SQL statement		`struct {`
		`long` *variable-name-1*`;`
		`char` *variable-name-2*`[`*n*`];`
		`} *`*structure-name*`;`
			$1 \leq n \leq 2000000$

SMALLINT ARRAY *m*		PD_MV_SINT(*m*) **variable-name*;	—
INTEGER ARRAY *m*		PD_MV_INT(*m*) **variable-name*;	—
DECIMAL[(*p*[,*s*])] ARRAY *m*		PD_MV_DEC(*m*,*p*,*s*) **variable-name*;	$1 \leq p \leq 29, 0 \leq s \leq p$
SMALLFLT ARRAY *m* (REAL)		PD_MV_SFLT(*m*) **variable-name*;	—
FLOAT ARRAY *m* (DOUBLE PRECISION)		PD_MV_FLT(*m*) **variable-name*;	—
CHAR[(*n*)] ARRAY *m* and MCHAR[(*n*)] ARRAY *m*		PD_MV_CHAR(*m*,*n*) **variable-name*;	$1 \leq n \leq 30000$
VARCHAR[(*n*)] ARRAY *m* and MVARCHAR[(*n*)] ARRAY *m*		PD_MV_VCHAR(*m*,*n*) **variable-name*;	$1 \leq n \leq 32000$
		PD_MV_CVCHAR(*m*,*n*) **variable-name*;	
NCHAR[(*n*)] ARRAY *m*		PD_MV_NCHAR(*m*,*n*) **variable-name*;	$1 \leq n \leq 15000$
NVARCHAR[(*n*)] ARRAY *m*		PD_MV_NVCHAR(*m*,*n*) **variable-name*;	$1 \leq n \leq 16000$

SQL data type	C language data description	Remarks
DATE ARRAY *m*	PD_MV_CHAR(*m*,10) **variable-name*;	—
TIME ARRAY *m*	PD_MV_CHAR(*m*,8) **variable-name*;	—
TIMESTAMP ARRAY *m*	PD_MV_CHAR(*m*,*n*) **variable-name*;	If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.
INTERVAL YEAR TO DAY ARRAY *m*	PD_MV_DEC(*m*,8,0) **variable-name*;	—
INTERVAL HOUR TO SECOND ARRAY *m*	PD_MV_DEC(*m*,6,0) **variable-name*;	—
Indicator variable for repetition column	PD_MV_SINT(*m*) **variable-name*;	—

Legend:

— : Not applicable

m: Number (0 - *m*-1) indicating each element in a repetition column

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Note

In 64-bit mode, use `int` instead of `long`.

* The defined length of the area cannot be determined during preprocessing. Therefore, at the time of execution, use `strlen(`*variable-name*`)` to determine the length of the character string stored in the area indicated by the pointer, and use this length in place of the area length. To receive the retrieval result, use a character other than `NULL` character to clear the area indicated by the pointer and enter the `NULL` character at the end.

To reference or set a variable for a pointer-type repetition column, use a dedicated macro. Table E-6 shows the macros for pointer-type repetition columns.

Table E-6: Macros for pointer-type repetition columns

SQL data type	Macro name	Data to be referenced or set	Data type
SMALLINT ARRAY *m*	PD_MV_SINTP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_SINTP_DATA(*variable-name*, *m*)	Each repetition element	short
INTEGER ARRAY m	PD_MV_INTP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_INTP_DATA(*variable-name*,*m*)	Each repetition element	long*
DECIMAL[(*p*[,*s*])] ARRAY *m*	PD_MV_DECP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_DECP_DATA(*variable-name*,*m*)	Start address of each repetition element in decimal	char[ ]
SMALLFLT ARRAY *m* (REAL)	PD_MV_SFLTP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_SFLTP_DATA(*variable-name*, *m*)	Each repetition element	float
FLOAT ARRAY *m* (DOUBLE PRECISION)	PD_MV_FLTP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_FLTP_DATA(*variable-name*,*m*)	Each repetition element	double
CHAR[(*n*)] ARRAY *m*, or MCHAR[(*n*)] ARRAY *m*	PD_MV_CHARP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_CHARP_DATA(*variable-name*, *m*)	Leading address of character string of each repetition element	char[ ]

SQL data type	Macro name	Data to be referenced or set	Data type
VARCHAR(*n*) ARRAY *m*, or MVARCHAR(*n*) ARRAY *m*	PD_MV_VCHARP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_VCHARP_LEN(*variable-name*, *m*)	Actual length of character string of each repetition element	short
	PD_MV_VCHARP_STR(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
	PD_MV_CVCHARP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_CVCHARP_DATA(*variable-name*, *m*)	Address of character string of each repetition element	char[ ]
NCHAR[(*n*)] ARRAY *m*	PD_MV_NCHARP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_NCHARP_DATA(*variable-name*,*m*)	Leading address of character string of each repetition element	char[ ]
NVARCHAR(*n*) ARRAY *m*	PD_MV_NVCHARP_CNT(*variable-name*)	Current repetition data element count	long*
	PD_MV_NVCHARP_LEN(*variable-name*,*m*)	Actual length of character string of each repetition element	short
	PD_MV_NVCHARP_STR(*variable-name*,*m*)	Leading address of character string of each repetition element	char[ ]
DATE ARRAY *m*	Same as CHAR(10)	—	—
TIME ARRAY *m*	Same as CHAR(8)	—	—
TIMESTAMP[(*p*)] ARRAY *m*	Same as CHAR(*n*) If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.	—	—
INTERVAL YEAR TO DAY	Same as DECIMAL(8,0)	—	—
INTERVAL HOUR TO SECOND	Same as DECIMAL(6,0)	—	—

SQL data type	Macro name	Data to be referenced or set	Data type
Indicator variable	`PD_MV_SINTP_CNT(`*variable-name*`)`	Indicator of the overall repetition column	`long`*
	`PD_MV_SINTP_DATA(`*variable-name*`,` *m*`)`	Indicator of each repetition column element	`short`

Legend:

— : Not applicable

m: Number of each repetition column element (0 - *m*-1)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

* In 64-bit mode, the data type is `int`.

Table E-7 shows the structures to be specified in batches.

Table E-7: Structures to be specified in batches

SQL data type	C language data description	Item coding	Remarks
Multiple items	Structure that contains the data types listed in Tables E-1 to E-3 as members	Specifies multiple embedded variables in a batch.	Pointers can be declared.
Indicator variable for multiple items	Structure that contains as members the indicator variables listed in Tables E-1 to E-3	Specifies multiple indicator variables in a batch.	Pointers can be declared.

E.2 SQL data types and COBOL data descriptions

This section provides the correspondence between SQL data types and COBOL data descriptions.

Data can be exchanged between variables of compatible data types and between variables of either convertible or assignable data types.

Table E-8 shows SQL data types and COBOL data descriptions. Table E-9 shows SQL data types and COBOL data descriptions when arrays are used. Table E-10 shows SQL data types and COBOL data descriptions when repetition columns are used. Note that the data descriptions in these tables can also be coded as follows:

```
PICTURE:
    PIC
COMPUTATIONAL:
    COMP
COMPUTATIONAL-n:
    COMP-n
9(n):
    99   9
X(n):
    XX   X
OCCURS n TIMES:
    OCCURS 1 TO n TIMES 0
    OCCURS 1 TO n
    OCCURS n
```

Table E-8: SQL data types and COBOL data descriptions

SQL data type	COBOL data description	Item coding	Remarks

`SMALLINT`	`L1` *elementary-item-name*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
		elementary-item or *independent-item*	—

`INTEGER`	`L1` *elementary-item-name*
	`PICTURE S9(9)`
	`COMPUTATIONAL.`
		elementary-item or *independent-item*	—

`DECIMAL [(`*p*`[,`*s*`])]`	`L1` *elementary-item-name*
	`PICTURE S9(`*p-s*`) [V9(`*s*`)]`
	`COMPUTATIONAL-3.`
		elementary-item or *independent-item*	If $1 \leq p \leq 29$[10], $0 \leq s \leq p$, and $p = s$, then `SV9(`*s*`)`.
			If $s = 0$, then `[V9(`*s*`)]` is omitted.

	`L1` *elementary-item-name*
	`PICTURE S9(`*p–s*`) [V9(`*s*`)]`
	`DISPLAY SIGN`
	`LEADING SEPARATE.`[9]		

`SMALLFLT (REAL)`	`L1` *elementary-item-name* `COMPUTATIONAL-1.`	*elementary-item* or *independent-item*	—
`FLOAT (DOUBLE PRECISION)`	`L1` *elementary-item-name* `COMPUTATIONAL-2.`	*elementary-item* or *independent-item*	—

SQL data type	COBOL data description	Item coding	Remarks
`CHAR[(n)]`	`L1` *elementary-item-name* `PICTURE X(n).`[5]	*elementary-item* or *independent-item*	$1 \leq n \leq 30000$

`VARCHAR(n)`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE X(n).`[5]
		A group item composed of two elementary items
		elementary-item-name-1: *character-string-length*
		elementary-item-name-2: *character-string*
			$1 \leq n \leq 32000$

`NCHAR[(n)]`	`L1` *elementary-item-name* `PICTURE N(n).`	*elementary-item* or *independent-item*	$1 \leq n \leq 15000$

`NVARCHAR(n)`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE N(n)`
		A group item composed of two elementary items
		elementary-item-name-1: *character-string-length*
		elementary-item-name-2: *character-string*
			$1 \leq n \leq 16000$

`MCHAR[(n)]`	`L1` *elementary-item-name* `PICTURE X(n).`[6]	*elementary-item or independent-item*	$1 \leq n \leq 30000$

`MVARCHAR(n)`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE X(n).`[6]
		A group item composed of two elementary items
		elementary-item-name-1: *character-string-length*
		elementary-item-name-2: *character-string*
			$1 \leq n \leq 32000$

`DATE`	`L1` *elementary-item-name* `PICTURE X(10).`[6]	*elementary-item* or *independent-item*	—
`TIME`	`L1` *elementary-item-name* `PICTURE X(8).`[6]	*elementary-item* or *independent-item*	—
`TIMESTAMP[(p)]`	`L1` *elementary-item-name* `PICTURE X(n).`[6]	*elementary-item* or *independent-item*	If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If p = 6, $n = 25$ or 26.

`INTERVAL YEAR TO DAY`	`L1` *elementary-item-name*
	`PICTURE S9(8)`
	`COMPUTATIONAL-3.`
		elementary-item or *independent-item*	—

SQL data type	COBOL data description	Item coding	Remarks
INTERVAL HOUR TO SECOND	L1 *elementary-item-name* PICTURE S9(6) COMPUTATIONAL-3.	*elementary-item* or *independent-item*	—
ROW[3]	Combination of data items and group items in this table[1]	A group item composed of elementary items	1 ≤ *total-length* ≤ 30000
BLOB	L2 *group-item-name*[2] [USAGE [IS]] SQL TYPE IS BLOB(*n*[K |M |G]).[4, 7]	*elementary-item*	Default: 1 ≤ *n* ≤ 2147483647 In units of K: 1 ≤ *n* ≤ 2097152 In units of M: 1 ≤ *n* ≤ 2048 In units of G: 1 ≤ *n* ≤ 2
BINARY(*n*)	L2 *group-item-name*. L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).[5, 7]	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* *elementary-item-name-2*: *character-string* *character-string-length* is the byte count.	1 ≤ *n* ≤ 2147483647
BLOB locator	L1 *elementary-item-name* SQL TYPE IS BLOB AS LOCATOR.[8]	*elementary-item* or *independent-item*	
BINARY locator	L1 *elementary-item-name* SQL TYPE IS BINARY AS LOCATOR.[8]	*elementary-item* or *independent-item*	

SQL data type		COBOL data description	Item coding	Remarks
Indicator variable	Other than BLOB, BINARY, BLOB locator, or BINARY locator	L1 *elementary-item-name* PICTURE S9(4) COMPUTATIONAL.	*elementary-item* or *independent-item*	—
	BLOB, BINARY, BLOB locator, or BINARY locator	L1 *elementary-item-name* PICTURE S9(9) COMPUTATIONAL.		
SQL statement		L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*)	A group item composed of two elementary items *elementary-item-name-1*: *character-string-length* *elementary-item-name-2*: *character-string*	$1 \leq n \leq 2000000$

Legend:

L1: Level number 01-49 or 77

L2: Level number 01-48

L3: Level number 02-49 (L2 < L3)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

[1] The following clauses can be used:

- REDEFINES
- OCCURS
- ADDRESSED BY

[2] A group item name should be coded as no more than 21 characters. However, for COBOL2002, a group item name should be 22 characters or less.

[3] Operations involving the ROW type are allowed only when the HiRDB server and the

HiRDB client use the same endian type.

[4] The coding of a BLOB UAP is expanded internally as follows:

```
L2 group-item-name.
  49 group-item-name_RESERVED PIC S(9) USAGE IS BINARY. 1
  49 group-item-name_LENGTH   PIC S(9) USAGE IS BINARY. 2
  49 group-item-name_DATA     PIC X(m).                 3
```

1. *group-item-name*_RESERVED is not used.
2. *group-item-name*_LENGTH is equal to the BLOB actual length.
3. *group-item-name*_DATA is the BLOB data storage area (where *m* denotes the actual data length).

[5] This item can be defined using 9 in place of X. If 9 is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

[6] Do not use 9 for X during definition, although using 9 does not cause an error during preprocessing.

[7] The maximum value that can be declared depends on the installed COBOL compiler. For details, see the manual for the COBOL compiler to be used.

[8] The following internal expansion takes place:

```
L1 elementary-item-name PICTURE S9(9) COMPUTATIONAL.
```

[9] The data type for the HiRDB server is the DECIMAL type, but it is represented as a signed external decimal item of the numeric type.

[10] The value range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

Table E-9: SQL data types and COBOL data descriptions when arrays are used

SQL data type	COBOL data description	Item coding	Remarks

`SMALLINT`	`L2` *elementary-item-name*
	`PICTURE S9(4)`
	`COMPUTATIONAL`
	`OCCURS` *m* `TIMES.`
		A group item composed of repetitions of data items in which the same data structure is repeated through specification of `OCCURS`	—

`INTEGER`	`L2` *elementary-item-name*
	`PICTURE S9(9)`
	`OCCURS` *m* `TIMES.`
			—

`DECIMAL [(`*p*`[,`*s*`])]`	`L2` *elementary-item-name*
	`PICTURE S9`
	`(`*p*-*s*`)[V9(`*s*`)]`
	`COMPUTATIONAL-3`
	`OCCURS` *m* `TIMES.`
			$1 \leq p \leq 29$[3], $0 \leq s \leq p$
			If $p = s$, SV9(*s*) is used.
			If $s = 0$, [V9(*s*)] is omitted.

	`L2` *elementary-item-name*
	`PICTURE S9(`*p*-*s*`)[V9(`*s*`)]`
	`DISPLAY SIGN LEADING`
	`SEPARATE OCCURS` *m*
	`TIMES.`		

`SMALLFLT (REAL)`	`L2` *elementary-item-name*
	`COMPUTATIONAL-1`
	`OCCURS` *m* `TIMES.`
			—

`FLOAT (DOUBLE PRECISION)`	`L2` *elementary-item-name*
	`COMPUTATIONAL-2`
	`OCCURS` *m* `TIMES.`
			—

`CHAR [(`*n*`)]`	`L2` *elementary-item-name* `PICTURE X(`*n*`)` `OCCURS` *m* `TIMES.`[1]		$1 \leq n \leq 30000$

`VARCHAR (`*n*`)`	`L2` *group-item-name*
	`OCCURS` *m* `TIMES.`
	`L3` *elementary-item-name-1*
	`PICTURE S9(4)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE X(`*n*`).`[*]
			$1 \leq n \leq 32000$

SQL data type	COBOL data description	Item coding	Remarks
NCHAR [(*n*)]	L2 *elementary-item-name* PICTURE N(*n*) OCCURS *m* TIMES.		1 ≤ *n* ≤ 15000
NVARCHAR (*n*)	L2 *group-item-name* OCCURS *m* TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(*n*)		1 ≤ *n* ≤ 16000
MCHAR [(*n*)]	L2 *elementary-item-name* PICTURE X(*n*) OCCURS *m* TIMES.[2]		1 ≤ *n* ≤ 30000
MVARCHAR (*n*)	L2 *group-item-name-2* OCCURS *m* TIMES. L3 *elementary-item-name-1* PICTURE S9(4) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*).[2]		1 ≤ *n* ≤ 32000
DATE	L2 *elementary-item-name* PICTURE X(10) OCCURS *m* TIMES.[2]		—
TIME	L2 *elementary-item-name* PICTURE X(8) OCCURS *m* TIMES.[2]		—
TIMESTAMP(*n*)	L2 *elementary-item-name* PICTURE X(*n*) OCCURS *m* TIMES.[2]		If *p* = 0, *n* = 19. If *p* = 2, *n* = 21 or 22. If *p* = 4, *n* = 23 or 24. If *p* = 6, *n* = 25 or 26.
INTERVAL YEAR TO DAY	L2 *elementary-item-name* PICTURE S9(8) COMPUTATIONAL-3 OCCURS *m* TIMES.		—

SQL data type	COBOL data description	Item coding	Remarks

`INTERVAL HOUR TO SECOND`	`L2` *elementary-item-name*
	`PICTURE S9(6)`
	`COMPUTATIONAL-3`
	`OCCURS` *m* `TIMES.`
			—

`ROW`	`L2` *group-item-name-2* `OCCURS` *m* `TIMES.` Combination of data items and group items in this table		—
`BLOB`	CN	CN	—

`BINARY`	`L2` *group-item-name-2*
	`OCCURS m TIMES.`
	`L3` *elementary-item-name-1*
	`PICTURE S9(9)`
	`COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE X(`*n*`).`[1]
		A group item composed of repetitions of data items in which the same data structure is repeated through specification of `OCCURS`.	• `FETCH` that uses an array $4 \leq n \leq 2147483644$ (n must be a multiple of 4.)
			• Other than `FETCH` that uses an array $4 \leq n \leq 32000$ (n must be a multiple of 4.)

`BLOB` locator	—	—	

`BINARY` locator	`L2` *elementary-item-name*
	`SQL TYPE IS`
	`BINARY AS LOCATOR`
	`OCCURS` *m* `TIMES.`
		Group item consisting of iterative data items that repeat the same data structure according to the `OCCURS` specification	

SQL data type		COBOL data description	Item coding	Remarks
Indicator variable	Other than BINARY or BINARY locator	L2 *elementary-item-name* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.		—
	BINARY or BINARY locator	L2 *elementary-item-name* PICTURE S9(9) COMPUTATIONAL OCCURS *m* TIMES.		—
SQL statement		CN	CN	—

Legend:

CN: Cannot be coded.

L2: Level number 02-49 (L2 < L3). You cannot specify level number 01, 66, 77, or 88 for L2. For details, see the syntax rules for the OCCURS clause in the COBOL manual.

L3: Level number 03-49

m: Number of array elements (1-4,096)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

[1] This item can be defined using 9 in place of X. If 9 is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

[2] Do not use 9 for X during definition, although using 9 does not result in an error during preprocessing.

[3] The range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

Table E-10: SQL data types and COBOL data descriptions when repetition columns are used

SQL data type	COBOL data description	Item coding	Remarks

`SMALLINT`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9(4) COMPUTATIONAL OCCURS` *m* `TIMES.`
		A group item composed of two elementary items	--

`INTEGER`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9(9) COMPUTATIONAL`
	`OCCURS` *m* `TIMES.`
			--

`DECIMAL`
`[(`*p*`[,`*s*`])]`
	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9`
	`(`*p-s*`)[V9(`*s*`)] COMPUTATIONAL-3 OCCURS` *m* `TIMES.`
			1 ≤ *p* ≤ 29[3],
			0 ≤ *s* ≤ *p*
			When *p* = *s*, SV9(*s*) is used.
			When *s* = 0, [V9(*s*)] is omitted.

	`L2` *group-item-name*`.`
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9(`*p–s*`)[V9(`*s*`)]`
	`DISPLAY SIGN LEADING SEPARATE`
	`OCCURS` *m* `TIMES.`		

`SMALLFLT (REAL)`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`COMPUTATIONAL-1 OCCURS` *m* `TIMES.`
			--

`FLOAT (DOUBLE PRECISION)`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`COMPUTATIONAL-2 OCCURS` *m* `TIMES.`
			--

SQL data type	COBOL data description	Item coding	Remarks
CHAR [(*n*)]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*) OCCURS *m* TIMES.[1]		1 ≤ *n* ≤ 30000
VARCHAR (*n*)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* OCCURS *m* TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE X(*n*).[1]	A group item composed of two elementary items and a group item composed of one elementary item.	1 ≤ *n* ≤ 32000
NCHAR [(*n*)]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE N(*n*) OCCURS *m* TIMES.	A group item composed of two elementary items.	1 ≤ *n* ≤ 15000
NVARCHAR (*n*)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* OCCURS *m* TIMES. L4 *elementary-item-name-3* PICTURE S9(4) COMPUTATIONAL. L4 *elementary-item-name-4* PICTURE N(*n*).	A group item composed of two elementary items and a group item composed of one elementary item.	1 ≤ *n* ≤ 16000
MCHAR [(*n*)]	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE X(*n*) OCCURS *m* TIMES.[1]	A group item composed of two elementary items.	1 ≤ *n* ≤ 30000

SQL data type	COBOL data description	Item coding	Remarks

`MVARCHAR` (*n*)	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(9) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`OCCURS` *m* `TIMES.`
	`L4` *elementary-item-name-3*
	`PICTURE S9(4) COMPUTATIONAL.`
	`L4` *elementary-item-name-4*
	`PICTURE X(`*n*`).`[1]
		A group item composed of two elementary items and a group item composed of one elementary item.	$1 \leq n \leq$ 32000

`DATE`	`L2` *group-item-name* `L3` *elementary-item-name-1* `PICTURE S9(9) COMPUTATIONAL.` `L3` *elementary-item-name-2* `PICTURE X(10) OCCURS` *m* `TIMES.`[2]	A group item composed of two elementary items.	—
`TIME`	`L2` *group-item-name* `L3` *elementary-item-name-1* `PICTURE S9(9) COMPUTATIONAL.` `L3` *elementary-item-name-2* `PICTURE X(8 OCCURS` *m* `TIMES.`[2]		—
`TIMESTAMP[(`*n*`)]`	`L2` *group-item-name*. `L3` *elementary-item-name-1* `PICTURE S9(9) COMPUTATIONAL.` `L3` *elementary-item-name-2* `PICTURE X(`*n*`) OCCURS` *m* `TIMES.`[2]		If $p = 0$, $n = 19$. If $p = 2$, $n = 21$ or 22. If $p = 4$, $n = 23$ or 24. If $p = 6$, $n = 25$ or 26.

`INTERVAL YEAR TO DAY`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(8) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9(8) COMPUTATIONAL-3`
	`OCCURS` *m* `TIMES.`
			—

`INTERVAL HOUR TO SECOND`	`L2` *group-item-name*
	`L3` *elementary-item-name-1*
	`PICTURE S9(6) COMPUTATIONAL.`
	`L3` *elementary-item-name-2*
	`PICTURE S9(6) COMPUTATIONAL-3`
	`OCCURS` *m* `TIMES.`
			—

`ROW`	CN	CN	—
`BLOB`	CN	CN	—

SQL data type	COBOL data description	Item coding	Remarks
BINARY	CN	CN	—
BLOB locator	CN	CN	—
BINARY locator	CN	CN	—
Indicator variable (other than BLOB, BINARY, BLOB locator, or BINARY locator)	L2 *group-item-name* L3 *elementary-item-name-1* PICTURE S9(9) COMPUTATIONAL. L3 *elementary-item-name-2* PICTURE S9(4) COMPUTATIONAL OCCURS *m* TIMES.	A group item composed of two elementary items.	—
SQL statement	CN	CN	—

Legend:

CN: Cannot be coded.

L2: Level number 02-49

L3 and L4: Level number 03-49

m: Maximum number of repetition column elements (2-30000)

n: Length (bytes)

p: Precision (total number of digits)

s: Scale (number of digits beyond the decimal point)

Notes

1. The value of *elementary-item-name-1* must be the current element count.
2. The values of *elementary-item-name-2* and *group-item-name-2* must be specified as the value of each repetition element.
3. *elementary-item-name-1* of the indicator variable must be specified as the indicator of the entire repetition column.
4. *elementary-item-name-2* of the indicator variable must be specified as the indicator of each repetition column element.

[1] This item can be defined using 9 in place of X. If 9 is used for definition, the operation when a character string containing a character other than a number is substituted or received as the retrieval result depends on the installed COBOL compiler.

[2] Do not use 9 for X during definition, although using 9 does not result in an error during preprocessing.

[3] The range depends on the specifications of the COBOL compiler. For example, for COBOL85, the range is $1 \leq p \leq 18$.

F. Data Dictionary Table Retrieval

HiRDB data dictionary tables can be referenced in the same way as an ordinary HiRDB database by using operation SQL statements. The authorization identifier of a dictionary table is MASTER.

This appendix provides examples of SQL descriptions for dictionary table retrievals and explains the definition information required for referencing.

Table F-1 lists the data dictionary tables that can be referenced.

Table F-1: Data dictionaries

Number	Table name	Description	Row contents
1	SQL_PHYSICAL_FILES	HiRDB file information (correspondences between HiRDB file system names and RDAREA names)	One HiRDB file
2	SQL_RDAREAS	Information such as the RDAREA names, their definition information, the RDAREA types, the number of stored tables, and number of indexes	One RDAREA
3	SQL_TABLES	Owner name and table name of each table (including dictionary tables) in the database	One table
4	SQL_COLUMNS	Column definition information, such as the column names and their data types	One column
5	SQL_INDEXES	Owner name and index name of each index (including dictionary tables) in the database	One index
6	SQL_USERS	Execution privileges and authorization identifiers of users authorized to access the database	One user
7	SQL_RDAREA_PRIVILEGES	Grants of RDAREA usage privileges	Use of one RDAREA for one authorization identifier
8	SQL_TABLE_PRIVILEGES	Grants of table access privileges	Access to one table for one authorization identifier
9	SQL_VIEW_TABLE_USAGE	Names of base tables used for view tables	One view table

Number	Table name	Description	Row contents
10	SQL_VIEWS	View definition information	One view table
11	SQL_DIV_TABLE	Table partitioning information (partitioning conditions specified in CREATE TABLE and names of RDAREAs that store partitioned tables)	One table (described by *n* rows)
12	SQL_INDEX_COLINF	Names of columns to which indexes are assigned	One index (described by *n* rows)
13	SQL_DIV_INDEX	Index partitioning information (storage RDAREA names)	One index (described by *n* rows)
14	SQL_DIV_COLUMN	BLOB-type column partitioning information (storage RDAREA names specified when CREATE TABLE was executed)	One column (described by *n* rows)
15	SQL_ROUTINES	Routine definition information	One routine (described by one row)
16	SQL_ROUTINE_RESOURCES	Information about resources used in a routine	One routine (described by *n* rows)
17	SQL_ROUTINE_PARAMS	Information about parameter definitions in a routine	One routine (described by *n* rows)
18	SQL_ALIASES	For UNIX: Alias information about tables (table aliases specified when CREATE ALIAS was executed and the three-part name of the target table) For Windows: Used by the system (table is empty)	For UNIX: One alias For Windows: None
19	SQL_TABLE_STATISTICS	Table statistical information	One table
20	SQL_COLUMN_STATISTICS	Column statistical information	One column
21	SQL_INDEX_STATISTICS	Index statistical information	One index
22	SQL_DATATYPES	Information about user-defined types	One user-defined type
23	SQL_DATATYPE_DESCRIPTORS	Information about user-defined type configuration attributes	One attribute

Number	Table name	Description	Row contents
24	SQL_TABLE_RESOURCES	Information about resources used in a table	One resource
25	SQL_PLUGINS	Plug-in information	One plug-in
26	SQL_PLUGIN_ROUTINES	Information about routines in a plug-in	One plug-in routine
27	SQL_PLUGIN_ROUTINE_PARAMS	Information about parameters in a plug-in routine	One set of parameter information
28	SQL_INDEX_TYPES	Information about index types	One index type
29	SQL_INDEX_RESOURCES	Information about resources used in an index	One set of resource information
30	SQL_INDEX_DATATYPE	Information about target items in an index	One set of target item information (for one level)
31	SQL_INDEX_FUNCTION	Information about abstract data type functions used in an index	One set of abstract data function information
32	SQL_TYPE_RESOURCES	Information about resources used in a user-defined type	One set of resource information
33	SQL_INDEX_TYPE_FUNCTION	Information about abstract data type functions used in an index that defines index types	One index type (described by *n* rows)
34	SQL_EXCEPT	Information about exclusion key values in an index	Exclusion key groups in one index (described by *n* rows)
35	SQL_FOREIGN_SERVERS	DBMS information of the foreign server that is accessed by HiRDB using the HiRDB External Data Access facility	One foreign server in one row
36	SQL_USER_MAPPINGS	Mapping information used for accessing a foreign server using the HiRDB External Data Access facility	One piece of mapping information in one row for one user on HiRDB
37	SQL_IOS_GENERATIONS	For UNIX: Generation information in the HiRDB file system areas when the inner replica facility is used For Windows: Used by the system (table is empty)	For UNIX: One row per HiRDB file system area For Windows: None

Number	Table name	Description	Row contents
38	SQL_TRIGGERS	Information on the trigger that is inside the schema	One trigger in one row
39	SQL_TRIGGER_COLUMNS	UPDATE trigger event column list information	One piece of event column information in one row
40	SQL_TRIGGER_DEF_SOURCE	Trigger definition source information	One piece of trigger definition source information in *n* rows
41	SQL_TRIGGER_USAGE	Resource information referenced inside a trigger action condition	One resource name being referenced inside the trigger action condition in one row
42	SQL_PARTKEY	Partitioning key information of a matrix-partitioned table	One piece of partitioning key information in one row
43	SQL_PARTKEY_DIVISION	Partitioning condition value information of a matrix-partitioned table	One piece of partitioning condition value information in one row
44	SQL_AUDITS	Information on the monitoring target	One object or information on one event for one user in one row
45	SQL_REFERENTIAL_CONSTRAINTS	Referential constraint conditions	Information on one constraint in one row
46	SQL_KEYCOLUMN_USAGE	Information on the columns that make up the external keys	Information on one column in one row
47	SQL_TABLE_CONSTRAINTS	Information on the integrity constraints in a schema	Information on one integrity constraint in one row
48	SQL_CHECKS	Check constraint information	Information on one check constraint in one row
49	SQL_CHECK_COLUMNS	Information on columns used by a check constraint	Information on one column using one check constraint in one row

Number	Table name	Description	Row contents
50	SQL_DIV_TYPE	Partitioning key information for matrix partitioning tables that combine key range partitioning and hash partitioning	Information on one partitioning key in one row
51	SQL_SYSPARAMS	Restriction information on the number of consecutive certification failures and the password character string	Information on one setting item in one row, and restriction information on one number of consecutive certification failures or one password character string in *n* rows

F.1 Examples of SQL statements for retrieval

Examples of SQL statements that retrieve data dictionary tables are shown as follows: For details about the SQL statements, see the *HiRDB Version 8 SQL Reference* manual.

The types of information that a particular user can retrieve depend on the setting of the data dictionary referencing authorization. For details about how to set data dictionary referencing authorizations, see the *HiRDB Version 8 System Operation Guide*.

After a dictionary table is retrieved, immediately issue a `COMMIT` statement or specify `WITHOUT LOCK NOWAIT` as shown in the retrieval example.

Example 1

Retrieve the server names for the RDAREAs that exist in the HiRDB system, the HiRDB filenames, and the names of the RDAREAs to which the HiRDB files belong:

```
SELECT X_SERVER_NAME, PHYSICAL_FILE_NAME, X.RDAREA_NAME
  FROM MASTER.SQL_PHYSICAL_FILES X, MASTER.SQL_RDAREAS Y
  WHERE X.RDAREA_NAME=Y.RDAREA_NAME
  ORDER BY SERVER_NAME
  WITHOUT LOCK NOWAIT
```

Example 2

From the column definition information for tables owned by a user, retrieve the names of the tables that contain the columns, the column names, the data types, and the column data lengths:

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH
  FROM MASTER.SQL_COLUMNS
  WHERE TABLE_SCHEMA=USER*
  ORDER BY TABLE_NAME
```

```
  WITHOUT LOCK NOWAIT
```

Example 3

From the index definition information for tables owned by a user, retrieve the names of the tables that contain the index, the index names, and the percentages of unused space per page:

```
SELECT TABLE_NAME, INDEX_NAME, FREE_AREA
  FROM MASTER.SQL_INDEXES
  WHERE TABLE_SCHEMA=USER*
  ORDER BY TABLE_NAME
  WITHOUT LOCK NOWAIT
```

Example 4

Retrieve the tables that a user can access and the types of access privileges to those tables (`SELECT`, `INSERT`, `DELETE`, and `UPDATE` privileges):

```
SELECT TABLE_NAME, SELECT_PRIVILEGE, INSERT_PRIVILEGE,
  DELETE_PRIVILEGE, UPDATE_PRIVILEGE
  FROM MASTER.SQL_TABLE_PRIVILEGES
  WHERE GRANTEE=USER* OR GRANTEE='PUBLIC'
  WITHOUT LOCK NOWAIT
```

Example 5

Retrieve the number of RDAREAs that become targets for group specification by a command (RDAREAs beginning with `RD1`):

```
SELECT COUNT(*) FROM MASTER.SQL_RDAREAS
  WHERE RDAREA_TYPE='U' AND
  RDAREA_NAME LIKE 'RD1%'
  WITHOUT LOCK NOWAIT
```

Example 6

Retrieve the names of RDAREAs that become targets for group specification by a command (RDAREAs beginning with `RD1`):

```
SELECT RDAREA_NAME FROM MASTER.SQL_RDAREAS
  WHERE RDAREA_TYPE='U' AND
  RDAREA_NAME LIKE 'RD1%' ORDER BY RDAREA_NAME
  WITHOUT LOCK NOWAIT
```

Example 7

Retrieve the name of the RDAREA that stores a non-partitioning table owned by a user (table named `T1`):

```
SELECT X.RDAREA_NAME
  FROM MASTER.SQL_RDAREAS X, MASTER.SQL_TABLES Y
  WHERE Y.TABLE_SCHEMA=USER*
    AND Y.TABLE_NAME='T1'
```

```
    AND X.RDAREA_NAME=Y.RDAREA_NAME
```

* USER refers to a variable that stores a value indicating the executing user's authorization identifier. For details about authorization identifiers, see the *HiRDB Version 8 SQL Reference* manual.

Example 8

Retrieve the name of the RDAREA that stores objects for a stored procedure or stored function, to be used during execution to re-initialize a data dictionary LOB RDAREA.

```
SELECT RDAREA_NAME FROM MASTER.SQL_DIV_COLUMN
  WHERE TABLE_SCHEMA='HiRDB'
    AND TABLE_NAME='SQL_ROUTINES'
    AND COLUMN_NAME='ROUTINE_OBJECT'
  WITHOUT LOCK NOWAIT
```

Note

When a data dictionary LOB RDAREA is reinitialized, all its stored SQL objects must be re-created.

Example 9

Retrieve the name of the stored procedure or stored function that has an invalid SQL object or an invalid index:

```
SELECT ROUTINE_SCHEMA,ROUTINE_NAME
    FROM MASTER.SQL_ROUTINES
    WHERE ROUTINE_VALID='N'
      OR INDEX_VALID='N'
    WITHOUT LOCK NOWAIT
```

Example 10

Retrieve the data types of the arguments that are actually used when embedded variables are used in arguments of the user-defined function FUNC1:

```
SELECT PARAMETER_NAME,DATA_TYPE,UDT_OWNER,UDT_NAME,
PARAMETER_NO
      FROM MASTER.SQL_ROUTINE_PARAMS
      WHERE ROUTINE_SCHEMA=USER AND ROUTINE_NAME='FUNC1'
        ORDER BY PARAMETER_NO
      WITHOUT LOCK NOWAIT
```

Example 11

To reorganize all the tables owned by user USERA, retrieve the RDAREAs containing any of those tables (the RDAREAs that need to be placed in shutdown status):

Non-partitioned table:

```
SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_TABLES
     WHERE TABLE_SCHEMA=USERA AND RDAREA_NAME IS NOT NULL
      WITHOUT LOCK NOWAIT
```

Partitioned table:

```
SELECT DISTINCT(RDAREA_NAME) FROM MASTER.SQL_DIV_TABLE
      WHERE TABLE_SCHEMA=USERA
      WITHOUT LOCK NOWAIT
```

Eliminate any duplicated RDAREA names from the result, then place all the resulting RDAREAs in shutdown status.

F.2 Data dictionary table details

The definition information required for referencing of each data dictionary table is shown as follows:

Each dictionary table has a column with the VARCHAR or MVARCHAR data type. This is the dictionary datatype operand for the database initialization utility or database structure modification utility, and must be set to either VARCHAR or MVARCHAR.

(1) SQL_PHYSICAL_FILES table

This table manages HiRDB file information (relationships between HiRDB files and RDAREAs). (Each row describes information on one HiRDB file.)

Table F-2 shows the contents of the SQL_PHYSICAL_FILES table.

Table F-2: SQL_PHYSICAL_FILES table contents

Number	Column name	Data type	Contents
1	SERVER_NAME	CHAR(8)	Server name (back-end server name or dictionary server name)
2	PHYSICAL_FILE_NAME	VARCHAR(167)	HiRDB filename
3	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the RDAREA to which HiRDB files are allocated
4	INITIAL_SIZE	INTEGER	Number of HiRDB file segments
5	PHYSICAL_FILE_ID	INTEGER	Physical file ID

(2) SQL_RDAREAS table

This table manages RDAREA definition information. (Each row describes information on one RDAREA.)

Table F-3 shows the contents of the SQL_RDAREAS table.

Table F-3: SQL_RDAREAS table contents

Number	Column name	Data type	Contents
1	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	RDAREA name
2	SERVER_NAME	CHAR(8)	Server name (back-end server name or dictionary server name)
3	RDAREA_TYPE	CHAR(1)	RDAREA type: M: Master directory RDAREA D: Data directory RDAREA S: Data dictionary RDAREA W: Work RDAREA U: User RDAREA P: Data dictionary LOB RDAREA L: User LOB RDAREA R: Registry RDAREA K: Registry LOB RDAREA A: list RDAREA
4	PAGE_SIZE	INTEGER	Page length (in bytes)
5	SEGMENT_SIZE	INTEGER	Segment size (in pages)
6	FILE_COUNT	INTEGER	Number of HiRDB files
7	N_TABLE	INTEGER	Number of tables stored (defined number) (initial value is 0)
8	N_INDEX	INTEGER	Number of indexes stored (defined number) (initial value is 0)
9	RDAREA_ID	INTEGER	RDAREA ID
10	REBALANCE_TABLE	CHAR(1)	Rebalance table status: Y: A rebalance table is used. Null value: No rebalance table is used.
11	MAX_ENTRIES	INTEGER	Maximum number of entries in the list NULL for any RDAREA other than the list RDAREA or if max entries is not specified
12	EXTENSION	CHAR(1)	Specification of RDAREA expansion: U: Specified. N: Not specified.
13	EXTENSION_SEGMENT_SIZE	INTEGER	Number of extension segments NULL if RDAREA expansion is not specified

Number	Column name	Data type	Contents
14	ORIGINAL_RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	For UNIX: Name of the original RDAREA Null value if the RDAREA is not a replica RDAREA. For Windows: Used by the system (no contents)
15	ORIGINAL_RDAREA_ID	INTEGER	For UNIX: ID of the original RDAREA Null value if the RDAREA is not a replica RDAREA. For Windows: Used by the system (no contents)
16	GENERATION_NUMBER	SMALLINT	For UNIX: Generation number Null value if the RDAREA is not an original RDAREA or replica RDAREA. For Windows: Used by the system (no contents)
17	REPLICA_COUNT	SMALLINT	For UNIX: Replica counter Null value if the RDAREA is not an original RDAREA or if the RDAREA has lost its replica RDAREA. For Windows: Used by the system (no contents)
18	REPLICA_STATUS	CHAR(1)	For UNIX: Replica status C: Current RDAREA S: Sub-RDAREA Null value if the RDAREA is not an original RDAREA or replica RDAREA. For Windows: Used by the system (no contents)
19	SHARED	CHAR(1)	Shared RDAREA S: Shared RDAREA Null value: Unshared RDAREA

(3) SQL_TABLES table

This table manages information of the tables found in schemas. (Each row describes information on one table.)

The rows of the SQL_TABLES table are created during table definition, and row deletion is performed during table deletion.

Table F-4 shows the contents of the SQL_TABLES table.

Table F-4: SQL_TABLES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner or PUBLIC for a public view table
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	TABLE_TYPE	CHAR(16)	Table type BASE TABLE: Base table VIEW: View table READ ONLY VIEW: Read-only view table FOREIGN TABLE: External table.
4	TABLE_ID	INTEGER	Table ID Indicates an internal ID that is unique within the system.
5	N_COLS	SMALLINT	Number of structure columns
6	N_INDEX	SMALLINT	Number of defined indexes (initial value is 0)
7	DCOLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Partitioned column name (column name of the first partitioning key for multiple column partitioning or matrix partitioning) Null value for a non-partitioned table, view tables, and foreign tables
8	VDEFLEN	INTEGER	Length of view analysis information Null value for base tables and foreign tables
9	FREE_AREA	SMALLINT	Percentage of unused space in each page 0 for a view table or a foreign table
10	FREE_PAGE	SMALLINT	Rate (%) of free pages (unused pages) inside a segment 0 for a view table or a foreign table
11	TABLE_COMMENT	VARCHAR(255) or MVARCHAR(255)	Comment (initial value is NULL)
12	CREATE_TIME	CHAR(14)	Table creation date and time (*YYYYMMDDHHMMSS*)

Number	Column name	Data type	Contents
13	ENQ_RESOURCE_SIZE	CHAR(1)	Locked resource unit P: In page units Null value for locking in row units and for view tables, foreign tables
14	DEFAULT_COLUMN	SMALLINT	Number of specified columns with the default value (DEFAULT clause or WITH DEFAULT).[2] Null value for view tables and dictionary tables
15	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of storage RDAREA for non-partitioned table (Null value for partitioned tables, view tables, and foreign tables)
16	DEFINITION_CACHE_SIZE	INTEGER	Table definition cache size (in bytes) (Null value for dictionary tables)
17	STATISTICS_CACHE_SIZE	INTEGER	Statistical information cache size (in bytes) (The initial value is a null value.)
18	N_RDAREA	INTEGER	Number of RDAREAs for storage of table (1-1024) 0 for a view table or a foreign table
19	FIX_TABLE	CHAR(1)	FIX specification F: Specified N: Not specified
20	VIEW_LEVEL	INTEGER	Number of nesting levels in view definition Null value for base tables and foreign tables
21	N_BASETABLE	INTEGER	Number of base tables used for a view table Null value for base tables and foreign tables
22	ROW_LENGTH	INTEGER	Row length of a FIX table Null value for tables that are not FIX tables, view tables, and foreign tables
23	N_NOTNULL	INTEGER	Number of NOT NULL values[2] (Null value for view tables and dictionary tables)
24	COMPRESS_TYPE	VARCHAR(8)	Data compression information: • Compression type (first byte) S: Data compression (SUPPRESS) • Suppressed data type (byte 2 and beyond): D: DECIMAL Null value for tables without SUPPRESS specification, view tables, dictionary tables, and foreign tables

Number	Column name	Data type	Contents
25	DIV_TYPE	CHAR(1)	Partitioning type P: Boundary value partitioning and matrix partitioning H: Flexible hash partitioning F: FIX hash partitioning M: Hash mixed matrix partitioning Null value for non-partitioned tables, key range partitioning tables, view tables, and foreign tables
26	HASH_NAME	VARCHAR(8) or MVARCHAR(8)	Hash function name "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" "HASHA" "HASHB" "HASHC" "HASHD" "HASHE" "HASHF" Null value for tables without a HASH specification, matrix partitioning tables, view tables, dictionary tables, and foreign tables.
27	N_LOB_COLUMN	SMALLINT	Number of columns with BLOB-data type (Null value for view tables and tables without BLOB columns)
28	N_LOB_RDAREA	INTEGER	Number of user LOB RDAREAs for a table Null value for view tables, tables without BLOB columns, tables without abstract data containing BLOB attributes, and foreign tables
29	CHANGE_TIME	CHAR(14)	Time table definition was changed (*YYYYMMDDHHMMSS*) (Null value when a table is initially created.)
30	N_DIV_COLUMN	SMALLINT	Number of partitioning key columns (216) Null value for non-partitioned tables, tables with single column partitioning keys specified, view tables, and foreign tables.

Number	Column name	Data type	Contents
31	COLUMN_SUP_INF	CHAR(1)	Whether or not data suppression is specified for each column: Y: Specified Null value: No specification Null value for tables for which column-by-column data suppression is not specified, view tables, and foreign tables
32	N_ADT_COLUMN	SMALLINT	Number of columns with an abstract data type Null value for tables in which the abstract data type is not defined, view tables, and foreign tables
33	WITHOUT_ROLLBACK	CHAR(1)	Whether or not a WITHOUT ROLLBACK is specified 'Y': Specified Null value: No specification Null value for tables for which WITHOUT ROLLBACK is not defined, view tables, and foreign tables
34	N_EXCEPT_VALUES	INTEGER	Number of exclusion key values in an index (Null value for indexes without exceptional value specifications and for view tables)
35	EXCEPT_VALUES_LEN	INTEGER	Total length of exclusion key values in an index (Null value for indexes without exceptional value specifications and for view tables)
36	REBALANCE	CHAR(1)	Whether or not the rebalancing facility is used: Y: Used. Null value for tables that do not use the rebalancing facility, view tables, and foreign tables
37	INDEXLOCK_OPT	CHAR(1)	Information used by the system
38	N_PK_COLUMNS	SMALLINT	Number of columns for the primary key Null value if no primary key is defined.
39	FOREIGN_SERVER_NAME	VARCHAR(30) or MVARCHAR(30)	External server name Null value for tables that are not foreign tables.
40	FOREIGN_SERVER_ID	INTEGER	External server ID Null value for tables that are not foreign tables.
41	BASE_FOREIGN_TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Authorization identifier or schema name of the user of a base table on a foreign server. Null value for tables that are not foreign tables.

Number	Column name	Data type	Contents
42	BASE_FOREIGN_TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a base table on a foreign server. Null value for tables that are not foreign tables.
43	N_RDAREA_BEFORE_REBALANCE	INTEGER	Number of RDAREAs storing the rebalancing table[1] Null value if rebalancing is started, and for tables that are not rebalancing tables, view tables, and foreign tables.
44	ON_REBALANCE	CHAR(1)	Rebalancing status: Y: Under execution Null value: Execution not ongoing Becomes Y after rebalancing has started, and becomes a null value when rebalancing is normally terminated.
45	SEGMENT_REUSE	CHAR(1)	Whether or not SEGMENT REUSE is specified Y: Specified Null value: Not specified Null value if NO is specified for SEGMENT REUSE (including when its specification is omitted), for view tables, and foreign tables.
46	N_REUSE_SEGMENT	INTEGER	Number of segments that start reusing free areas.[3] Null value if NO is specified for SEGMENT REUSE (including when its specification is omitted), for view tables, and foreign tables.
47	REUSE_SEGMENT_SIZE	CHAR(10)	Specified number of segments that start reusing free areas.[4] Null value if a value other than a segment count is specified for SEGMENT REUSE, for view tables, and foreign tables.
48	REUSE_SEGMENT_SIZE_TYPE	CHAR(1)	Unit for the number of segments that start reusing free areas. K: Specifies K. M: Specifies M. Blank space: Specification omitted Null value if a value other than a segment count is specified for SEGMENT REUSE, for view tables, and foreign tables.

Number	Column name	Data type	Contents
49	INSERT_ONLY	CHAR(1)	Whether or not the falsification prevention facility is specified Y: Specified Null value: Not specified Null value if the falsification prevention facility is not used, for view tables, and foreign tables.
50	DELETE_PROHIBIT_TERM_TYPE	CHAR(1)	Type of deletion prevented duration I: Date interval data Y: Labeled duration (YEAR) M: Labeled duration (MONTH) D: Labeled duration (DAY) Null value: Not specified Null value if the falsification prevention facility is not used, if no deletion prevented duration is specified, for view tables, and foreign tables
51	DELETE_PROHIBIT_TERM	CHAR(10)	Specification value for the deletion prevented duration[5] Null value if the falsification prevention facility is not used, if no deletion prevented duration is specified, for view tables, and foreign tables.
52	SYSGEN_COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the insert history maintenance column Null value if the falsification prevention facility is not used, if no deletion prevented duration is specified, for view tables, and foreign tables.
53	N_TRIGGER	INTEGER	Number of defined triggers Null value if no trigger is defined, and for view tables, foreign tables, and dictionary tables.
54	N_DIV_DIMENSION	SMALLINT	Number of division dimensions Null value for tables that are not matrix-partitioned tables.
55	AUDIT_TABLE_OPTION	CHAR(1)	Value that specifies whether this table is an audit trail table. Y: Audit trail table V: View table based on an audit trail table Null value for tables that are not audit trail tables and not view tables based on an audit trail table
56	N_PARENTS	SMALLINT	Number of foreign keys Null value for tables without a defined referential constraint, view tables, and foreign tables.

Number	Column name	Data type	Contents
57	N_CHILDREN	SMALLINT	Number of foreign keys that reference the main keys of this table Null value for unreferenced tables that are not referenced tables, view tables, and foreign tables.
58	N_FK_COLUMNS	SMALLINT	Total number of foreign key columns Null value for tables without a defined referential constraint, view tables, and foreign tables.
59	CHECK_PEND	CHAR(1)	Type of check pending status for a referential constraint C: Pending status Null value: Non-pending status Null value for view tables, and foreign tables.
60	N_CHECK	INTEGER	Number of defined check constraints Null value for tables without a defined referential constraint, view tables, and foreign tables.
61	N_CHECK_LIMIT	INTEGER	Check constraint limit[6] Null value for tables without a defined referential constraint, view tables, and foreign tables.
62	CHECK_PEND2	CHAR(1)	Type of check pending status for a check constraint C: Pending status Null value: Non-pending status Null value for view tables, and foreign tables.
63	CHK_SOURCE_LEN	INTEGER	Total length of search conditions of a check constraint Null value for tables without a defined referential constraint, view tables, and foreign tables.
64	SHARED	CHAR(1)	Shared table specification S: Shared table Null value: Unshared table
65	CHANGE_TIME_INSERT_ONLY	CHAR(14)	Update date and time of a falsification prevention table (*YYYYMMDDHHMMSS*) Null value when a table is defined and for view tables, and foreign tables.
66	N_UPDATE_COLUMN	SMALLINT	Number of columns for which an updatable column attribute is specified Null value for tables without a specified updatable column attribute, view tables, and foreign tables.

Number	Column name	Data type	Contents
67	TABLE_CREATOR	VARCHAR(30) or MVARCHAR(30)	Creator of a public view table Null value for table that are not public view tables.
68	N_ENCRYPTED_COLUMN	SMALLINT	Used by the system; always the null value.
69	CRYPTO_LIBRARY_TYPE	CHAR(1)	Used by the system; always the null value.

[1] If an RDAREA is added to a rebalancing table using ALTER TABLE ADD RDAREA, the column contains the number of table storage RDAREAs before the RDAREA was added.

[2] If a foreign table is created using the HiRDB External Data Access facility and NO is specified in the NULLABLE column option, NOT NULL WITH DEFAULT is assumed. Therefore, the columns in the DEFAULT_COLUMN column for which WITH DEFAULT is specified are counted, as well as the columns in the N_NOTNULL column that contains non-null values.

[3] When a segment count unit is specified, the following values are stored:

When K is specified: Specified value $\times$ 1024

When M is specified: Specified value $\times$ 1024^2

[4] Values are stored right-justified. Note that the segment count units (K and M) are not included.

[5] The following is stored depending on the type of deletion prevented duration:

When 'I' is specified: +*YYYYMMDD*. character format

When 'Y', 'M', or 'D' is specified: Right-justified character format

[6] The check constraint limit is the sum of the total number of logical operators specified in the search conditions of the check constraints (number of AND and OR specifications, excluding the AND and OR specifications in WHEN search conditions of CASE expressions) and the total number of check constraints.

Example

If a table is defined as follows, the check constraint limit is 4 (the total number of operators (AND and OR) is 2 and the total number of check constraints is 2):

```
CREATE TABLE "STOCK"
  ("GNO" CHAR(5),"GNAME" CHAR(8),"PRICE" INTEGER,
    "QUANTITY" INTEGER,"STOCKING DATE" DATE)
```

```
CHECK("QUANTITY "≥ 100 AND "QUANTITY" ≤ 1000)
CONSTRAINT "QUANTITY RULE"
CHECK("STOCKING DATE"=DATE('1992-08-21')
  OR "STOCKING DATE"=DATE('1992-09-21'))
CONSTRAINT "STOCKING DATE RULE"
```

(4) SQL_COLUMNS table

This table manages column definition information. (Each row describes information on one column.)

Rows of the `SQL_COLUMNS` table are created during table definition, and row deletion (including schema deletion) is performed during table deletion.

Table F-5 shows the contents of the `SQL_COLUMNS` table.

Table F-5: SQL_COLUMNS table contents

Number	Column name	Data type	Contents
1	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Table owner or `PUBLIC` for a public view table
2	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the table that contains the column
3	`COLUMN_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Column name
4	`TABLE_ID`	`INTEGER`	Table ID
5	`COLUMN_ID`	`SMALLINT`	Column ID (integer beginning with 1; values less than 1 are not allowed)
6	`DATA_TYPE`	`CHAR(24)`	Data type[1]
7	`DATA_LENGTH`	`CHAR(7)`	Column data length is stored right justified in character format (blanks are used for leading zeros)
8	`IS_NULLABLE`	`CHAR(3)`	Column null information[5]: `YES`: Null value allowed `NO`: Null values not allowed
9	`DIVIDED_KEY`	`CHAR(1)`	Partitioning key: `Y`: Partitioning key Blank: Not a partitioning key
10	`CLUSTER_KEY`	`CHAR(1)`	Cluster key: `Y`: Column used for cluster key Blank: Not a column used for cluster key

Number	Column name	Data type	Contents
11	COLUMN_COMMENT	VARCHAR(255) or MVARCHAR(255)	Comment (The initial value is a null value.)
12	BASE_TYPE	CHAR(1)	Base column type[8]: C: Column F: Function, operation E: Other Null value for base tables and foreign tables
13	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of base table that contains base column Null value for base tables and foreign tables
14	BASE_TABLE	VARCHAR(30) or MVARCHAR(30)	Name of base table that contains base column Null value for base tables and foreign tables
15	BASE_COLUMN	VARCHAR(30) or MVARCHAR(30)	Base column name Null value for base tables and foreign tables
16	DEFAULT_COLUMN	CHAR(1)	WITH DEFAULT specification[5] Y: Specified N: Not specified Null value for view tables
17	COLUMN_OFFSET	SMALLINT	Column offset Null value for tables that are not FIX tables, view tables, and foreign tables.
18	HASH_KEY	CHAR(1)	Hash key: Y: Hash key Blank: Other than hash key
19	RECOVERY_TYPE	CHAR(1)	RECOVERY specification: A: ALL P: PARTIAL N: NO (Null value if the data type is not BLOB.)
20	LOB_LENGTH	CHAR(20)	Column length specification stored right-justified in character format (blanks are used for leading zeros) Null value if the length is not for BLOB or BINARY.

Number	Column name	Data type	Contents
21	LOB_LENGTH_TYPE	CHAR(1)	Column length type (in column lengths): K: K specified M: M specified G: G specified Blank: Default (Null value if the data type is not BLOB.)
22	DATA_TYPE_CODE	SMALLINT	Data type code[2]
23	DATA_LENGTH_CODE	SMALLINT	Column data length code[3]
24	LOB_LENGTH_CODE	CHAR(8)	BLOB column data length code[4, 6] (Null value if the data type is not BLOB or BINARY.)
25	DIVCOL_ORDER	SMALLINT	Partitioning key specification order (0-16) Unique values within the applicable table, beginning with 1. Partitioning key specification order +1. 0 is specified for a column that is not a partitioning key. Null value for non-partitioned tables, tables with single column partitioning keys specified, view tables, and foreign tables.
26	SUPPRESS_INF	CHAR(1)	Whether or not data suppression is specified: Y: Specified Null value: No specification Null value for tables without data suppression specifications, view tables, and for foreign tables
27	PLUGIN_ DESCRIPTION	VARCHAR(255)	Plug-in option contents Null value if no PLUGIN clause is specified, and for foreign tables.
28	UDT_OWNER	VARCHAR(30)	Owner of a user-defined type Null value if the type is not user-defined, and for foreign tables.
29	UDT_NAME	VARCHAR(30)	Name of the user-defined type Null value if the type is not user-defined, and for foreign tables.
30	UDT_TYPE_ID	INTEGER	User-defined type ID Null value if the type is not user-defined, and for foreign tables.

Number	Column name	Data type	Contents
31	MAX_ELM	SMALLINT	Maximum number of repetition column elements (Null value if the column is not a repetition column.)
32	NO_SPLIT	CHAR(1)	Whether or not NO SPLIT is specified: Y: Specified Null value: No specification Null value for view tables, foreign tables, and if ALTER TABLE CHANGE SPLIT is executed.
33	PRIMARY_KEY	CHAR(1)	Primary key type Y: Primary key Blank: Other than the primary key
34	COLLATING_SEQUENCE	CHAR(1)	Character code and collating sequence for the character string type column of a foreign server and HiRDB External Data Access S: SAME D: DIFFERENT Null value for tables that are not foreign tables and if the data type of a foreign table column is not the character string type.
35	TRAILING_SPACE	CHAR(1)	Whether or not there are trailing spaces in a column of character string type in the external table: Y: There are trailing spaces. N: There are no trailing spaces. Null value for tables that are not foreign tables and if the data type of a foreign table column is not the variable character string type.
36	SYSTEM_GENERATED	CHAR(1)	Whether or not SYSTEM GENERATED is specified Y: Specified Null value: No specification Null value if SYSTEM GENERATED is not specified, for view tables, and foreign tables.
37	DEFAULT_CLAUSE	CHAR(1)	Whether or not the DEFAULT clause is specified Y: Specified Null value: No specification Null value if the DEFAULT clause is not specified, for view tables, and foreign tables.
38	DEFAULT_VALUE	VARCHAR(32000) or MVARCHAR(32000)[7]	Default value (character format) specified for the DEFAULT clause.[9] Null value if the DEFAULT clause is not specified, for view tables, and foreign tables.

Number	Column name	Data type	Contents
39	DEFAULT_VALUE2	VARCHAR(32000) or MVARCHAR(32000)[7]	Default value specified for the DEFAULT clause (stores the 32,001st - 64,000th byte values in the character format when a literal is specified).[9] Null value if a literal is not specified, if the DEFAULT clause is not specified, for view tables, and foreign tables.
40	DEFAULT_VALUE3	VARCHAR(3) or MVARCHAR(3)	Default value specified for the DEFAULT clause (stores the 64,000th byte value and beyond in the character format when a literal is specified).[9] Null value if a literal is not specified, if the DEFAULT clause is not specified, for view tables, and foreign tables.
41	CHECK_COLUMN	CHAR(1)	Check constraint specification Y: Specified Null value for tables in which a check constraint is not defined, view tables, and foreign tables.
42	FOREIGN_KEY	CHAR(1)	Foreign key type Y: Foreign key configuration table Null value: Non-foreign key configuration table
43	UPDATABLE	CHAR(1)	Updatable column attribute U: Can be updated (UPDATE) N: Can be updated only once from a null value to a non-null value (UPDATE ONLY FROM NULL) Null value for tables for which the updatable attribute is not specified, view tables, and foreign tables.
44	CRYPTO_LIBRARY_TYPE	CHAR(1)	Used by the system; always the null value.

[1] The stored value depends on the data type, as follows:

Data type	Value to be stored
INT	INTEGER
INTEGER	
SMALLINT	SMALLINT
DEC	DECIMAL
DECIMAL	

Data type	Value to be stored
FLOAT	FLOAT
DOUBLE PRECISION	
SMALLFLT	SMALLFLT
REAL	
CHAR	CHAR
VARCHAR	VARCHAR
NCHAR	NCHAR
NVARCHAR	NVARCHAR
MCHAR	MCHAR
MVARCHAR	MVARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
INTERVAL YEAR TO DAY	INTERVAL YEAR TO DAY
INTERVAL HOUR TO SECOND	INTERVAL HOUR TO SECOND
BINARY	BINARY
BLOB	BLOB
BINARY LARGE OBJECT	
Abstract data type	ADT
BOOLEAN	BOOLEAN

[2] For the specified data types and the values to be stored, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[3] For the `DECIMAL`, `INTERVAL YEAR TO DAY`, and `INTERVAL HOUR TO SECOND` types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the `NCHAR` and `NVARCHAR` types) is stored in the 2-byte binary format. Note that the value is `0` for the `BLOB`, `BINARY`, and abstract data types.

[4] The specified column length is stored in binary format in 8 bytes divided into 4-byte segments.

[5] If a foreign table is created using the HiRDB External Data Access facility and `YES` is specified in the `NULLABLE` column option, `YES` is assumed in the `IS_NULLABLE` column and `N` for the `DEFAULT_COLUMN` column. If `NO` is specified in the `NULLABLE` column option, `NOT NULL WITH DEFAULT` is assumed, and `NO` is assumed in the `IS_NULLABLE` column and `Y` for the `DEFAULT_COLUMN` column. Additionally, if `NO` is specified in the `NULLABLE` column option, the values in the `DEFAULT_COLUMN` and `N_NOTNULL` columns in the `SQL_TABLES` table are counted.

[6] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

[7] Specifies `NO SPLIT`.

[8] `E` (Other) is set when the selection formula is one of the following:

- Scalar operations (four arithmetic operations, data operation, time operation, `CASE` expression, and scalar functions)
- Literal
- `CAST` specification
- Function invocation (excluding plug-in functions)
- `USER`
- `CURRENT_DATE`
- `CURRENT_TIME`
- `CURRENT_TIMESTAMP`

[9] Table F-6 shows the values that are stored when the `DEFAULT` clause is specified.

Table F-6: Values that are stored when the DEFAULT clause is specified

Default value	Data type[1]	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column[2]	
		Data size (in char format)	Default value (character format)
Omitted	All	Null value	Null value
`NULL`	All	4	`'NULL'`

Default value	Data type[1]	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column[2]	
		Data size (in char format)	Default value (character format)
USER	CHAR and MCHAR	4	'USER'
	VARCHAR and MVARCHAR		
CURRENT DATE	DATE, or CHAR(10)	12	'CURRENT Δ DATE'[3]
CURRENT_DATE		12	'CURRENT_DATE'
CURRENT TIME	TIME or CHAR(8)	12	'CURRENT Δ TIME'[3]
CURRENT_TIME		12	'CURRENT_TIME'
CURRENT TIMESTAMP(*p*) (*p*: decimal seconds precision)	TIMESTAMP, CHAR(19), CHAR(22), CHAR(24), or CHAR(26)	20	'CURRENT Δ TIMESTAMP(*p*)'[3, 7]
CURRENT_TIMESTAMP(*p*) (*p*: decimal seconds precision)		20	'CURRENT_TIMESTAMP(*p*)'[7]

Default value			Data type[1]	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column[2]	
				Data size (in char format)	**Default value (character format)**

Lit	Char string lit	Character string literal
		Example 1:
		`'HiRDB'`
		Example 2:
		`Δ'2002-10-24 10:50:23.1234'`
			`CHAR` or `MCHAR`	*def-val-size* + 2[4]	*specified-default-value-size*[4]
					Example: `''HiRDB''`

			`VARCHAR` or `MVARCHAR`		
			`DATE`, `TIME`, or `TIMESTAMP`	*def-val-size* + 2[4]	*specified-default-value-size*[4] Example: `''2002-10-24Δ10:50:23.1234''`
		Mixed character string literal Example: `M'100 years'`	`CHAR` or `MCHAR`	*def-val-size* + 3[4]	*specified-default-value-size*[4] Example: `'M'100 years''`
			`VARCHAR` or `MVARCHAR`		
		National character string literal Example: `N'software'`	`NCHAR` or `NVARCHAR`	*def-val-size* + 3[4]	*specified-default-value-size*[4] Example: `'N'software''`

		Hexadecimal character string literal
		Example 1:
		`X'48692D43'`
		Example 2:
		`X'2002102410502312'`
			`CHAR`, `VARCHAR`, `MCHAR`, `MVARCHAR`, or `BINARY`	*def-val-size* + 3[4]	Example: `'X'48692D43''`[4, 6]

			`DATE`, `TIME`, or `TIMESTAMP` (*p*)		Example: `'X'2002102410502312''`[4, 6]

Default value			Data type[1]	Value stored in DEFAULT_VALUE column, DEFAULT_VALUE2 column, or DEFAULT_VALUE3 column[2]	
				Data size (in char format)	Default value (character format)
	Num lit	Integer literal Example: `10`	`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, or `SMALLFLT`	*def-val-size*[5]	*specified-default-value*[5] Example: `'10'`
		Floating-point literal Example: `15e + 3`	`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, or `SMALLFLT`	22 or 23	*specified-default-value*[5] Example: `'+1.500000000000000E+04'` (From the left, 1 byte for a sign, 17 bytes for the virtual number portion (decimal literal), 1 byte for 'E', 1 byte for a sign, 2-3 bytes for the exponential part (power of 10))

		Decimal literal
		Example 1: `15.5`
		Example 2: `-010101.`
		Example 3: `00011399.`
			`INTEGER`, `SMALLINT`, `DECIMAL`, `FLOAT`, `SMALLINT`, `INTERVAL YEAR TO DAY`, or `INTERVAL HOUR TO SECOND`	*def-val-size*[5]	*specified-default-value*[5]
					Example 1: `' 15.5'`
					Example 2: `'-010101.'`
					Example 3:
					`'+00020199.'` for `INTERVAL YEAR TO DAY`
					`' 00011399.'` for `INTEGER`
					(For `INTERVAL YEAR TO DAY` and `INTERVAL HOUR TO SECOND`, the value is corrected and a sign is added to the front (the value is blank in all other cases and for a positive value))

The following abbreviations are used in this table:

Num: Numeric

Lit: Literal

Char: Character

def: default

val: value

Δ: 1-byte blank space

[1] Excludes `BLOB`, the abstract data type, and `BINARY` of 32,001 bytes or greater.

[2] If the data size is smaller than 32,001 bytes, the `DEFAULT_VALUE2` column and `DEFAULT_VALUE3` column become null values. If the data size is 32,001-64,000 bytes, the `DEFAULT_VALUE3` column becomes a null value.

[3] Blank spaces between `CURRENT` and `DATE`, `TIME`, or `TIMESTAMP` are edited into a single blank space.

[4] The specified default value is stored as a literal expression in the character format. The data size and default value include the literal expressions `M`, `N`, `X`, and apostrophe (`'`). Therefore, the data size range is 2-32,002 bytes including `' '` for a character string literal, 3-32,003 bytes including `M' '` and `N' '` for a mixed character string literal and a national character string literal, and 3-64,003 bytes including `X' '` for a hexadecimal character string literal.

Bytes 1-32,000 of the specified literal are stored in the `DEFAULT_VALUE` column; bytes 32,001-64,000 are stored in the `DEFAULT_VALUE2` column; and bytes 64,000 and beyond are stored in the `DEFAULT_VALUE3` column.

Example:

When 32,000 bytes worth of a default value is specified for the hexadecimal character string literal (a total of 64,003 bytes including `X` and an apostrophe (`'`))

```
VARCHAR(32000) DEFAULT X'C1C1C1...C1C1C1'
```

The first 32,000 bytes `X'C1C1C1...` are stored in the `DEFAULT_VALUE` column.

The next 32,000 bytes `C1C1C1...` are stored in the `DEFAULT_VALUE2` column.

The remaining 3 bytes `C1'` are stored in the `DEFAULT_VALUE3` column.

[5] The specified default value is stored as a literal expression in the character format. Size in the character format expression is stored for the data size.

Example:

When a default value is specified for the numeric literal

```
INTEGER DEFAULT 100
```

The first 3 bytes `100` are stored in the `DEFAULT_VALUE` column.

Null values are stored in the `DEFAULT_VALUE2` and `DEFAULT_VALUE3` columns.

[6] The value is all upper-case letters (upper-case letters are stored even when lower-caser letters are specified for the value).

[7] If the decimal precision (p) for the CURRENT_TIMESTAMP value to be specified for the default value is omitted, $p = 0$ is assumed.

(5) SQL_INDEXES table

This table manages index information. (Each row describes information on one index.)

Table F-7 shows the contents of the SQL_INDEXES table.

Table F-7: SQL_INDEXES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains an index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	TABLE_ID	INTEGER	Table ID
6	UNIQUE_TYPE	CHAR(1)	Unique type: U: Unique N: Non-unique
7	COLUMN_COUNT	SMALLINT	Number of columns comprising the index
8	CREATE_TIME	CHAR(14)	Index creation date and time (*YYYYMMDDHHMMSS*)
9	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of storage RDAREA for non-partitioned index Null value for partitioning key indexes and foreign indexes
10	CLUSTER_KEY	CHAR(1)	Index type: Y: Cluster index N: Non-cluster index
11	DIV_INDEX	CHAR(1)	Type of first column of the columns that make up the index: Y: Partitioning key or plug-in index (The same order from the first key of partitioning keys specified in CREATE TABLE for multiple-partitioning keys) N: Not a partitioning key

Number	Column name	Data type	Contents
12	FREE_AREA	SMALLINT	Percentage of unused space in each page (%) 0 for foreign indexes
13	COLUMN_ID_LIST	VARCHAR(64)	List of IDs of columns constituting the index[1] Ascending and descending orders are indicated with + and -. + is set to specify the descending order of single-column indexes (other than cluster key indexes). + is always set for plug-in indexes.
14	SPLIT_OPT	CHAR(1)	Page split option: U: Unbalanced split Null value for indexes for which unbalanced split is not specified, and foreign indexes.
15	ATTR_COUNT	SMALLINT	Number of abstract data type attributes constituting an index Null value for CREATE INDEX (Format 1)
16	INDEX_TYPE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of an index type Null value for CREATE INDEX (Format 1), and foreign indexes
17	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of an index type Null value for CREATE INDEX (Format 1), and foreign indexes
18	INDEX_TYPE_ID	INTEGER	Index type ID Null value for CREATE INDEX (Format 1), and foreign indexes
19	PLUGIN_ DESCRIPTION	VARCHAR(255)	Plug-in option contents Null value if PLUGIN is not specified, and for foreign indexes.
20	N_FUNCTION	INTEGER	Number of applied functions Null value for CREATE INDEX (Format 1), and foreign indexes
21	EXCEPT_VALUES	CHAR(1)	Whether or not exclusion key values are specified: Y: Specified N: Not specified
22	N_EXCEPT_VALUES	SMALLINT	Number of exclusion key values in an index Null value for indexes without exception value specifications

Number	Column name	Data type	Contents
23	ARRAY_TYPE	CHAR(1)	Type of the columns that make up the index: M: Includes repetition columns Null value: The columns that make up the index do not include repetition columns.
24	LOCK_OPT	CHAR(1)	Information used by the system
25	PRIMARY_KEY	CHAR(1)	Index type Y: Primary key index Null value: Not a primary key index
26	DIV_IN_SRV	CHAR(1)	Whether or not a non-partitioning key index is partitioned within the server: Y: Partitioned within the server Null value: Not partitioned within the server Null value for partitioning key indexes as well
27	SHARED	CHAR(1)	Shared index specification S: Shared index Null value: Unshared index

[1] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(6) SQL_USERS table

This table manages information about the execution and DBA (database administration) privileges of users. (Each row describes information on one user.)

This table can be referenced only by owners with the DBA privilege and auditors.

Table F-8 shows the contents of the SQL_USERS table.

Table F-8: SQL_USERS table contents

Number	Column name	Data type	Contents
1	USER_ID	VARCHAR(30) or MVARCHAR(30)	Name of the user with privileges
2	DBA_PRIVILEGE	CHAR(1)	DBA privilege: Y: Has the DBA privilege N: Does not have the DBA privilege
3	SCHEMA_PRIVILEGE	CHAR(1)	Schema definition privilege: Y: Has the schema definition privilege S: Owns a schema N: Does not have the schema definition privilege The initial value is N.

Number	Column name	Data type	Contents
4	CREATE_TIME	CHAR(14)	Schema creation date and time (*YYYYMMDDHHMMSS*) The initial value is a null value; also a null value when DROP SCHEMA is executed.
5	AUDIT_PRIVILEGE	CHAR(1)	Audit privilege status: Y: Granted Null value: Not granted Null value for any user who is not the auditor.
6	AUTH_ERR_COUNT	SMALLINT	Number of consecutive certification failures Null value if the number of consecutive certification failures is not specified, the number of consecutive user certification failures is 0, or the number of continuous certification failures has been cleared.
7	CON_LOCK_TIME	TIMESTAMP(0)	Consecutive certification failure account lock date and time Null value if the number of consecutive certification failures is not specified or if the consecutive certification failure account lock state has not occurred.*
8	PWD_LOCK_TIME	TIMESTAMP(0)	Password-invalid account lock date and time Null value if a password character string limit is not specified or if the password-invalid account lock state has not occurred.
9	PASSWORD_TEST	CHAR(1)	Password limit violation type code L: Minimum number of allowed bytes U: Specification of authentication indicator prohibited S: Specification of single-character type prohibited Null value if the user for whom the password-invalid account lock state occurs has not been prechecked or if there is no violation after the precheck.

* If the consecutive certification failure account lock is set and no connection is established after the specified account lock period has elapsed, a null value is not set even if the consecutive certification failure account lock state has not occurred.

(7) SQL_RDAREA_PRIVILEGES table

This table manages the assignment of RDAREA usage privileges. (Each row describes information on one user of one RDAREA.)

Table F-9 shows the contents of the SQL_RDAREA_PRIVILEGES table.

Table F-9: SQL_RDAREA_PRIVILEGES table contents

Number	Column name	Data type	Contents
1	GRANTEE	VARCHAR(30) or MVARCHAR(30)	Name of the user with the RDAREA usage privilege or PUBLIC
2	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the RDAREA
3	GRANT_TIME	CHAR(14)	Date and time at which the relevant privilege was granted (*YYYYMMDDHHMMSS*)

(8) SQL_TABLE_PRIVILEGES table

This table manages the granting of table access privileges. (Each row describes information on one user.)

Rows of the SQL_TABLE_PRIVILEGES table are created when users are granted table access privileges by GRANT. Rows are deleted when all of a user's privileges are revoked by REVOKE.

Table F-10 shows the contents of the SQL_TABLE_PRIVILEGES table.

Table F-10: SQL_TABLE_PRIVILEGES table contents

Number	Column name	Data type	Contents
1	GRANTOR	VARCHAR(30) or MVARCHAR(30)	Name of the user granting the table access privileges or the definer of the public view table
2	GRANTEE	VARCHAR(30) or MVARCHAR(30)	Name or role name, of the user who receives table access privilege, or PUBLIC
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which access privilege is to be granted. PUBLIC for a public view table.
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which access privileges are to be granted
5	SELECT_PRIVILEGE	CHAR(1)	SELECT privilege status: G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
6	INSERT_PRIVILEGE	CHAR(1)	INSERT privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.

Number	Column name	Data type	Contents
7	DELETE_PRIVILEGE	CHAR(1)	DELETE privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
8	UPDATE_PRIVILEGE	CHAR(1)	UPDATE privilege status G: Granted (for a table owner) Y: Granted N: Not granted The initial value is N.
9	GRANT_TIME	CHAR(14)	Date and time at which the relevant privilege was granted (*YYYYMMDDHHMMSS*)
10	GRANTEE_TYPE	CHAR(1)	Type of table access privilege grantee: G: Role registered in the directory server Null if GRANTEE or the user is PUBLIC.

(9) SQL_VIEW_TABLE_USAGE table

This table manages information of the base tables that serve as the basis for view tables. (Each row describes information on one view table.)

Table F-11 shows the contents of the SQL_VIEW_TABLE_USAGE table.

Table F-11: SQL_VIEW_TABLE_USAGE table contents

Number	Column name	Data type	Contents
1	VIEW_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of a view table or PUBLIC for a public view table
2	VIEW_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a view table
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the base table or the resource to be used or PUBLIC for a public view table
4	BASE_TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the base table or the resource to be used
5	BASE_TYPE	CHAR(1)	Type of the base table or the resource to be used R: Real table V: View table F: External table P: User-defined function (excluding plug-in functions)

(10) SQL_VIEWS table

This table manages view table definition information. (Each row describes information on one view table.)

Table F-12 shows the contents of the `SQL_VIEWS` table.

Table F-12: SQL_VIEWS table contents

Number	Column name	Data type	Contents
1	`VIEW_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of a view table or `PUBLIC` for a public view table
2	`VIEW_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of a view table
3	`SOURCE_ORDER`	`INTEGER`	Order if source is divided and stored in multiple rows (1-*n*)
4	`IS_UPDATABLE`	`CHAR(3)`	Update possibility: `YES`: Possible `NO`: Not possible
5	`VIEW_DEFINITION`	`VARCHAR(32000)` or `MVARCHAR(32000)`	View definition source statements
6	`VIEW_ID`	`INTEGER`	View ID

(11) SQL_DIV_TABLE table

This table manages table partitioning information in the database. (Each row describes information on one table.)

Table F-13 shows the contents of the `SQL_DIV_TABLE` table.

Table F-13: SQL_DIV_TABLE table contents

Number	Column name	Data type	Contents
1	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of a view table
2	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of a view table
3	`DIV_NO`	`INTEGER`	Partitioning condition specification order (unique value beginning with 1 for the corresponding table, which is obtained by adding 1 to the partitioning condition specification order)

Number	Column name	Data type	Contents
4	`TABLE_ID`	`INTEGER`	Table ID
5	`DCOND`	`CHAR(2)`	Partitioning condition code The partitioning storage condition value is stored in character format; the storable values are `=`, `^=`, <, <=, >, and >=; if <> or `!=` is specified, it is stored as `^=`. For a matrix-partitioned table, <= is stored. Blank if no partitioning storage condition is specified or if hash partitioning is specified.
6	`DCVALUES`	`VARCHAR(256)` or `MVARCHAR(256)`	Partitioning condition value (Null value if no partitioning storage condition is specified or if hash partitioning is specified.)
7	`RDAREA_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of storage RDAREA
8	`DCVALUES2`	`VARCHAR(255)` or `MVARCHAR(255)`	Second dimension key partitioning condition value (The storage format is the same as that for `DCVALUES`.) Null value for a table that is not a matrix-partitioned table and for a matrix-partitioned table for which no boundary value is specified.

(12) SQL_INDEX_COLINF table

This table manages index column information. (Each row describes information on one index.)

Table F-14 shows the contents of the `SQL_INDEX_COLINF` table.

Table F-14: SQL_INDEX_COLINF table contents

Number	Column name	Data type	Contents
1	`TABLE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Table owner
2	`TABLE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the table that contains an index
3	`INDEX_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Index name
4	`INDEX_ID`	`INTEGER`	Index ID

Number	Column name	Data type	Contents
5	INDEX_ORDER	INTEGER	Order of columns comprising the index (integer beginning with 1, which identifies the name order of columns comprising the index)
6	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (name of columns comprising the index)
7	ASC_DESC	CHAR(1)	Ascending or descending order: A: Ascending order D: Descending order Blank: (for plug-in indexes) (If descending order is specified for a single-column index, it is stored as ascending order.)

(13) SQL_DIV_INDEX table

This table manages index partitioning information (partitioning conditions and names of storage RDAREAs specified by CREATE TABLE). (Each row describes information on one index.)

Table F-15 shows the contents of the SQL_DIV_INDEX table.

Table F-15: SQL_DIV_INDEX table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains an index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	DIV_NO	INTEGER	RDAREA definition order (unique value beginning with 1 for the corresponding index which is obtained by adding 1 to the RDAREA definition order)[1]
5	INDEX_ID	INTEGER	Index ID
6	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of partitioned storage RDAREA comprising the index)

[1] This value is not related to DIV_NO of SQL_DIV_TABLE.

(14) SQL_DIV_COLUMN table

This table manages BLOB-type column partitioning information (name of storage RDAREA specified by CREATE TABLE). (Each row describes information on one column.)

Table F-16 shows the contents of the SQL_DIV_COLUMN table.

Table F-16: SQL_DIV_COLUMN table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name
4	DIV_NO	INTEGER	Storage order
5	RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user LOB RDAREA
6	STORE_NO	INTEGER	Always 1
7	MASTER_RDAREA_NAME	VARCHAR(30) or MVARCHAR(30)	Name of user RDAREA for the corresponding table
8	N_LEVEL	SMALLINT	Number of levels (Null value for BLOB type columns)
9	COMPONENT_NAME	VARCHAR(30) or MVARCHAR(30)	Component name (Null value for BLOB type columns)
10	LOB_NO	SMALLINT	LOB attribute number (Null value for BLOB type columns)

(15) SQL_ROUTINES table

This table manages routine definition information. (Each row describes information on one routine.)

Table F-17 shows the contents of the SQL_ROUTINES table.

Table F-17: SQL_ROUTINES table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner

Number	Column name	Data type	Contents
2	ROUTINE_NAME	VARCHAR(30) or MVARCHAR(30)	Routine name[10]
3	OBJECT_ID	INTEGER	Object ID
4	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name[2]
5	ROUTINE_TYPE	CHAR(1)	Routine type: P: Procedure F: Function
6	ROUTINE_VALID	CHAR(1)	Validity flag: Y: Validity routine N: Invalidity routine
7	INDEX_VALID	CHAR(1)	Index status change flag: Y: Index status valid[1] N: Index status invalid[1]
8	CREATE_TIME	CHAR(14)	Routine creation date and time (*YYYYMMDDHHMMSS*) SQL analysis time for SQL procedure statements or definition creation time for external routines
9	ALTER_TIME	CHAR(14)	Routine re-creation date and time (*YYYYMMDDHHMMSS*) (The initial value is a null value.)
10	OBJECT_SIZE	INTEGER	Object size (in bytes) 0 for external routines
11	SOURCE_SIZE	INTEGER	Definition source size (bytes) 0 for external routines and registry operation procedures
12	ISOLATION_LEVEL	SMALLINT	Data guarantee level (0-2) Valid for procedures
13	OPTIMIZE_LEVEL	INTEGER	SQL optimization option (converted to decimal format) Specifies the value of OPTIMIZE LEVEL for CREATE PROCEDURE, ALTER PROCEDURE, CREATE TYPE, or ALTER ROUTINE.
14	SQL_LEVEL	SMALLINT	SQL level (0-2) Valid for procedures
15	N_PARAM	INTEGER	Number of parameters

Number	Column name	Data type	Contents
16	N_RESOURCE	INTEGER	Number of resources used in an object
17	PARAM_LOCATION	INTEGER	Start position of a procedure statement in a definition source statement.[8]
18	ROUTINE_ COMMENT	VARCHAR(255) or MVARCHAR(255)	Comment (The initial value is a null value.)
19	DEF_SOURCE	BLOB	Definition source statement (not including compiler options) Null value for foreign routines (excluding Java routines), registry operation procedures, and trigger action procedures.
20	ROUTINE_ADT_OWNER	VARCHAR(30)	Owner of the abstract data type that defined routines (Null value for routines that are not defined inside the abstract data type)
21	ROUTINE_ADT_NAME	VARCHAR(30)	Name of the abstract data type that defined routines (Null value for routines that are not defined inside the abstract data type)
22	ROUTINE_BODY	CHAR(1)	Function routine type: S: SQL procedure E: External routine T: Trigger action procedure Null value for procedures (excluding trigger action procedures) that are not foreign routines.
23	FUNCTION_TYPE	CHAR(1)	Function type: C: System-defined function constructor Blank: User-defined function (Null value for procedures)
24	EXTERNAL_NAME	VARCHAR(255)	External routine name (*library-name* ! *operation-name*) or a Java method name if defined in Java Null value if the name is not for a foreign function.
25	EXTERNAL_LANGUAGE	CHAR(20)	External descriptive language type: C: C language Java: Java language Null value if the language type is not for a foreign function.

Number	Column name	Data type	Contents
26	PARAMETER_STYLE	VARCHAR(20)	Parameter style (external routine type) PLUGIN: Plug-in RDSQL: System-defined scalar function Java: Java Null value if the parameter style is not for a foreign function.
27	ENCAPSULATION_LEVEL	VARCHAR(10)	Encapsulation level (PUBLIC, PRIVATE, or PROTECTED) (Null value for routines that are not defined inside the abstract data type.)
28	RETURN_UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of a return value data type (Null value if the return value is not a user-defined function.)
29	RETURN_UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a return value data type (Null value if the return value is not a user-defined function.)
30	RETURN_UDT_TYPE_ID	INTEGER	ID of a return value data type (Null value if the return value is not a user-defined function.)
31	RETURN_DATA_TYPE	CHAR(24)	Return value data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table. (Null value if the return value data type is not a function.)
32	RETURN_DATA_TYPE_CODE	SMALLINT	Code for a return value data type[3] (Null value if the return value data type is not a function.)
33	RETURN_DATA_LENGTH_CODE	SMALLINT	Code for a return value data length[4] (Null value for procedures)
34	RETURN_DATA_LENGTH	CHAR(7)	Return value data length stored right-justified in character format (blanks are used for leading zeros) (Null value for procedures)
35	RETURN_LOB_LENGTH_CODE	CHAR(8)	Code for a return value BLOB data length[5, 9] (Null value for procedures, or if the return value is not a BLOB or BINARY function.)

Number	Column name	Data type	Contents
36	RETURN_LOB_ LENGTH	CHAR(20)	Specification value of a return value BLOB data length Right-justified in character format (blanks are used for leading zeros) (Null value for procedures, or if the return value is not a BLOB or BINARY function.)
37	RETURN_LOB_ LENGTH_TYPE	CHAR(1)	Type of a return value BLOB data length: K: K specified M: M specified G: G specified Blank: Default (Null value for procedures, or if the return value is not a BLOB or BINARY function.)
38	ADDITIONAL_ OPTIMIZE_LEVEL	INTEGER	Extended SQL optimization option (converted to decimal format) Specifies the value of ADD OPTIMIZE LEVEL for CREATE PROCEDURE, ALTER PROCEDURE, CREATE TYPE, or ALTER ROUTINE. Null value if the routine was created by HiRDB of Version 06-00 or earlier.
39	CLASS_NAME	VARCHAR(255)	*package-name*.*class-name*[6] Null value if the foreign routine is not coded in Java.
40	JAR_NAME	VARCHAR(255)	Java archive file name Null value if the foreign routine is not coded in Java.
41	DYNAMIC_RESULT_ SETS	SMALLINT	Maximum number of result sets to be returned Null value if no maximum number is specified for the result sets.
42	SQL_ SPECIFICATION	CHAR(1)	Data access specification: C: CONTAINS SQL M: MODIFIES SQL N: NO SQL R: Used by the system; always the null value.
43	RETURNS_JAVA_ DATA_TYPE	VARCHAR(255)	Java return value's data type corresponding to return value's data type[7] Null value if the foreign routine is not coded in Java.

Number	Column name	Data type	Contents
44	RETURNS_JAVA_ DATA_TYPE_CODE	INTEGER	Java return value's data type code corresponding to return value's data type[7] Null value if the foreign routine is not coded in Java.
45	RETURN_DATA_ MAX_ELM	SMALLINT	Maximum number of elements for return value's data type Null value if ARRAY is not specified for the return value data type.
46	N_JAVA_RESULT_ SETS	INTEGER	Number of Java.sql.ResultSet[]s specified Null value if Java.sql.ResultSet[] is not specified.
47	FOR_UPDATE_EXCLUSIV E_LOCK	CHAR(1)	Whether ISOLATION LEVEL is a value other than 2 and FOR UPDATE EXCLUSIVE is specified Y: Yes Null value: No Null value for routines created with an HiRDB versions earlier than 07-01, if FOR UPDATE EXCLUSIVE has not been specified, and if the ISOLATION LEVEL value is 2.
48	SUBSTR_LENGTH	SMALLINT	Specification value of SUBSTR LENGTH of the SQL compile option Null value for routines created with HiRDB versions earlier than 08-00, or when the character code type is not Unicode (UTF-8).

[1] Index information in the routine is invalid (the routine cannot be executed). In this case, SQL objects must be re-created by ALTER ROUTINE or ALTER PROCEDURE.

[2] For procedures, this name is the same as the routine name; for functions, the system internally generates a name from the routine name and object ID as follows:

F routine name (up to 19 bytes) object ID (10 bytes)

[3] For details about the specified data type and values to be stored, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[4] For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

[5] The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

[6] The following shows the storage format for *package-name*`.`*class-name*:

- Package name specified

  package-name`.`*class-name*
- Package name not specified

  class-name

[7] The following Java data types are stored as a character string in `RETURN_JAVA_DATA_TYPE`. The Java data types expressed in hexadecimal numbers are stored in `RETURN_JAVA_DATA_TYPE_CODE`.

Java data type	Value in hexadecimal
`byte[]`	`1000`
`byte[][]`	`100A`
`short`	`1002`
`short[]`	`1003`
`int`	`1004`
`int[]`	`1005`
`float`	`1006`
`float[]`	`1007`
`double`	`1008`
`double[]`	`1009`
`java.match.BigDecimal`	`2000`
`java.match.BigDecimal[]`	`2001`
`java.lang.String`	`2002`
`java.lang.String[]`	`2003`
`java.sql.Date`	`2004`
`java.sql.Date[]`	`2005`
`java.sql.Time`	`2006`
`java.sql.Time[]`	`2007`
`java.lang.Double`	`2008`
`java.lang.Double[]`	`2009`

Java data type	Value in hexadecimal
`java.lang.Float`	200A
`java.lang.Float[]`	200B
`java.lang.Integer`	200C
`java.lang.Integer[]`	200D
`java.lang.Short`	200E
`java.lang.Short[]`	200F
`java.sql.Timestamp`	2010
`java.sql.Timestamp[]`	2011
`void`	0000

[8] The location at which the procedure statement starts is counted from the top of the SQL statement, beginning at 1. For an external routine (Java routine), the location at which the external routine specification (`EXTERNAL NAME` clause) begins is counted from the top of the SQL statement. A value of 0 is set for the following:

- External routine (excluding Java routines)
- Registry manipulation procedure
- Trigger action procedure

[9] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

[10] For a trigger action procedure, the following routine name (22 bytes long) is stored:

`'(TRIG`*yyyymmddhhmmssth*`)'`

yyyymmddhhmmssth: Time stamp at the time of trigger definition (units: 1/100 seconds)

(16) SQL_ROUTINE_RESOURCES table

This table manages resource information used in routines. (*n* rows describe information on one routine.)

Table F-18 shows the contents of the `SQL_ROUTINE_RESOURCES` table.

Table F-18: SQL_ROUTINE_RESOURCES table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner
2	ROUTINE_NAME	VARCHAR(30) or MVARCHAR(30)	Routine name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name[1]
4	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Resource owner or PUBLIC for a public view table
5	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	Resource identifier
6	BASE_TYPE	CHAR(1)	Resource type: R: Base table V: View table I: Index D: Data type P: Routine F: External table T: Trigger
7	ROUTINE_TYPE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of abstract data type for routine defined in abstract data type (Null value for routines that are not defined inside the abstract data type.)
8	ROUTINE_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of abstract data type for routine defined in abstract data type (Null value for routines that are not defined inside the abstract data type.)
9	SELECT_OPERATION[2]	CHAR(1)	Retrieval target specification status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]
10	INSERT_OPERATION[2]	CHAR(1)	Data insertion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]

Number	Column name	Data type	Contents
11	UPDATE_OPERATION[2]	CHAR(1)	Data update target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]
12	DELETE_OPERATION[2]	CHAR(1)	Data deletion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]
13	LOCK_OPERATION[2]	CHAR(1)	Data insertion target status: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]
14	PURGE_OPERATION[2]	CHAR(1)	Whether or not a data deletion target is specified in a PURGE TABLE statement: Y: Specified Null value: Not specified Null value if the type of resource used is not R or V.[3]

[1] For procedures, this name is the same as the routine name; for functions, the system internally generates a name from the routine name and object ID as follows:

'F' *routine name* (up to 19 bytes) *object ID* (10 bytes)

[2] If a view table is used as an SQL object, information that merges the operation types of all view tables being used is set in the base table (the highest order base table if the base table is a view table) that is the base for the view table being used as the SQL object.

[3] If the type of resource being used is a view table (V), a null value is set for a view table that is not actually contained in the SQL object.

(17) SQL_ROUTINE_PARAMS table

This table manages parameter information in routines. (*n* rows describe information on one routine.)

Table F-19 shows the contents of the SQL_ROUTINE_PARAMS table.

Table F-19: SQL_ROUTINE_PARAMS table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner
2	ROUTINE_NAME	VARCHAR(30) or MVARCHAR(30)	Routine name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name
4	PARAMETER_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter name[5]
5	PARAMETER_NO	INTEGER	Parameter specification sequence (a unique number within the routine beginning with 1)
6	DATA_TYPE	CHAR(24)	Data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table. (Null value if the data type is BLOB.)
7	DATA_LENGTH	CHAR(7)	Data length stored right-justified in character format (blanks are used for leading zeros) (Null value if the data type is BLOB, BINARY, or a user-defined type.)
8	LOB_LENGTH	CHAR(20)	Column length specification value right-justified in character format (blanks are used for leading zeros) (Null value if the data type is not BLOB or BINARY.)
9	LOB_LENGTH_TYPE	CHAR(1)	Column length type: K: K specified M: M specified G: G specified Blank: Default (Null value if the data type is not BLOB.)
10	PARAMETER_MODE	CHAR(5)	Parameter I/O mode: IN: Input mode NOUT: Output mode INOUT: Input/output mode NONE: Other than above
11	DATA_TYPE_CODE	SMALLINT	Data type code[1] (Null value if the data type is BLOB.)

Number	Column name	Data type	Contents
12	DATA_LENGTH_CODE	SMALLINT	Data length code[2] (Null value if the data type is BLOB, BINARY, or a user-defined type.)
13	LOB_LENGTH_CODE	CHAR(8)	Column length specification value[3, 4] (Null value if the data type is not BLOB or BINARY.)
14	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of a data type parameter (Null value if the parameter is the system-defined type.)
15	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of a data type parameter (Null value if the parameter is the system-defined type.)
16	UDT_TYPE_ID	INTEGER	ID of a data type parameter (Null value if the parameter is the system-defined type.)
17	JAVA_DATA_TYPE	VARCHAR(255)	Data type of the corresponding Java parameter For the storage format, see the RETURNS_JAVA_DATA_TYPE column in the SQL_ROUTINES table. Null value if the foreign routine is not coded in Java.
18	JAVA_DATA_TYPE_CODE	INTEGER	Data type code of the corresponding Java parameter For the storage format, see the RETURNS_JAVA_DATA_TYPE_CODE column in the SQL_ROUTINES table. Null value if the foreign routine is not coded in Java.
19	MAX_ELM	SMALLINT	Maximum number of parameter elements Null value if the number of parameter elements is not specified.

Number	Column name	Data type	Contents
20	TRIGGER_COLUMN	CHAR(1)	Parameter information for the column specified by an old or new values correlation name of the trigger action procedure O: Column referenced by an old values correlation name N: Column referenced by a new values correlation name Null value: Neither of the above Null value if the parameter is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.
21	TRIGGER_TABLE_ID	INTEGER	Table ID that defines the column before it is replaced with a parameter Null value if the ID is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.
22	TRIGGER_COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name before being replaced with a parameter Null value if the name is not for a trigger action procedure or does not correspond to a column specified by an old or new values correlation name.

[1] For the specified data types and the values to be stored, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[2] For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

[3] The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

[4] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

[5] For a trigger action procedure, the following parameter name (27 bytes long) is used:

'(T#*tbl_id*#*col_id*#*nnnnn*)'

tbl_id

Table ID (hexadecimal, 8 digits (If the number of digits is less than 8, the front portion is zero filled.))

col_id

Column ID (hexadecimal, 8 digits (If the number of digits is less than 8, the front portion is zero filled.))

nnnnn

`00001`: Parameter that corresponds to a column modified by an old values correlation name

`00002`: Parameter that corresponds to a column modified by a new values correlation name

(18) SQL_ALIASES table

This table manages table alias information (table alias specified when `CREATE ALIAS` was executed and the three-part name of the target table). (Each row describes information on one alias.) For the Windows version, the `SQL_ALIASES` table is empty.

Table F-20 shows the contents of the `SQL_ALIASES` table.

Table F-20: SQL_ALIASES table contents

Number	Column name	Data type	Contents
1	`ALIAS_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of the alias
2	`ALIAS_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Alias
3	`ALIAS_TYPE`	`CHAR(1)`	Alias type: `T`: Table Blank: Others
4	`RDNODE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	RD node name
5	`BASE_SCHEMA`	`VARCHAR(30)` or `MVARCHAR(30)`	Owner of the alias or `PUBLIC` for a public view table.
6	`BASE_NAME`	`VARCHAR(30)` or `MVARCHAR(30)`	Name of the alias

(19) SQL_TABLE_STATISTICS table

This table manages table statistical information. (Each row describes information on one table.)

If there is no statistical information (for example, immediately following `CREATE TABLE`), the contents of this table are empty.

Table F-21 shows the contents of the SQL_TABLE_STATISTICS table.

Table F-21: SQL_TABLE_STATISTICS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	N_PAGE	FLOAT	Number of pages stored (statistical information) Null value if lvl1 is specified for the -c option of pdgetcst
4	N_ROW	FLOAT	Total number of rows (statistical information)
5	UPDATE_TIME	CHAR(14)	Update date and time (*YYYYMMDDHHMMSS*)

(20) SQL_COLUMN_STATISTICS table

This table manages column statistical information. (Each row describes information on one column.)

If there is no statistical information (for example, immediately after CREATE TABLE), the contents of this table are empty.

Table F-22 shows the contents of the SQL_COLUMN_STATISTICS table.

Table F-22: SQL_COLUMN_STATISTICS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains a column
3	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name
4	N_UNIQUE	FLOAT	Number of unique values (statistical information)
5	N_MAX_DUP_KEY	FLOAT	Maximum number of duplicate key values (statistical information)
6	N_MIN_DUP_KEY	FLOAT	Minimum number of duplicate key values (statistical information)
7	N_NULL	FLOAT	Number of null values
8	UPDATE_TIME	CHAR(14)	Update date and time (*YYYYMMDDHHMMSS*)

Number	Column name	Data type	Contents
9	RANGE_VALUES	VARCHAR(2464)	Column value frequency distribution information (statistical information)[1]

[1] The maximum and minimum column values set in the pdgetcst parameter file are stored in the RANGE_VALUES column after being converted into an internal format. To reference these maximum and minimum values, the SQL described as follows must be executed. The retrieval results are displayed in hexadecimal.

- SQL for retrieving the maximum column value

```
SELECT HEX(SUBSTR("RANGE_VALUE"),33,a)
    FROM "MASTER".SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

For *a*, specify the data length of the column in bytes. If the data is of the character string type, it is truncated to 16 bytes, so a value equal to or less than 16 must be specified.

- SQL for retrieving the minimum column value

```
SELECT HEX(SUBSTR("RANGE_VALUE"),49,a)
    FROM "MASTER".SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

For *a*, specify the data length of the column in bytes. If the data is of the character string type, it is truncated to 16 bytes, so a value equal to or less than 16 must be specified.

Example

Referencing the maximum column value of an INT-type column

```
SELECT HEX(SUBSTR("RANGE_VALUE"),33,4)
    FROM "MASTER".SQL_COLUMN_STATISTICS
    WITHOUT LOCK NOWAIT
```

Output result (when maximum column value is 10)

```
'0000000A'
```

(21) SQL_INDEX_STATISTICS table

This table manages index statistical information. (Each row describes information on one index.)

If there is no statistical information (for example, immediately following CREATE TABLE), the contents of this table are empty.

Table F-23 shows the contents of the SQL_INDEX_STATISTICS table.

Table F-23: SQL_INDEX_STATISTICS table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	N_ENTRY	FLOAT	Number of key entries (statistical information)
5	N_IXPG	FLOAT	Number of leaf pages (statistical information)
6	N_LEVEL	SMALLINT	Number of levels (statistical information)
7	SEQ_RATIO	INTEGER	Sequential level (statistical information)
8	UPDATE_TIME	CHAR(14)	Update date and time (*YYYYMMDDHHMMSS*)

(22) SQL_DATATYPES table

This table manages user-defined type information (each row defines information on one user-defined type).

Table F-24 shows the contents of the SQL_DATATYPES table.

Table F-24: SQL_DATATYPES table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the user-defined type
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user-defined type
3	META_TYPE	CHAR(1)	Type of the user-defined type: A: Abstract data type
4	TYPE_ID	INTEGER	ID of the user-defined type
5	N_ATTR	SMALLINT	Number of attributes
6	CREATE_TIME	CHAR(14)	Creation date and time (*YYYYMMDDHHMMSS*)
7	N_SUBTYPE	INTEGER	Number of subtypes
8	SOURCE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the supertype abstract data type (Null value if there is no supertype abstract data type.)

Number	Column name	Data type	Contents
9	SOURCE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the supertype abstract data type (Null value if there is no supertype abstract data type.)
10	SOURCE_TYPE_ID	INTEGER	ID of the supertype abstract data type (Null value if there is no supertype abstract data type.)
11	ROOT_TYPE_ID	INTEGER	ID of the highest order abstract data type if the supertype abstract data type also has a supertype
12	LEVEL_NO	SMALLINT	Number of generations from highest order supertype abstract data type if the supertype abstract data type also has a supertype
13	TYPE_COMMENT	VARCHAR(255)	Comment (The initial value is a null value; null value is also used if there is no comment.)
14	N_LOB_ATTR	SMALLINT	Number of BLOB-type attributes
15	N_ADT_ATTR	SMALLINT	Number of abstract-data-type attributes
16	N_LARGE_BINARY_ATTR	SMALLINT	Number of attributes for BINARY-type data of 32,001 bytes or more

(23) SQL_DATATYPE_DESCRIPTORS table

This table manages user-defined type attribute information. (Each row describes information on one attribute.)

Table F-25 shows the contents of the SQL_DATATYPE_DESCRIPTORS table.

Table F-25: SQL_DATATYPE_DESCRIPTORS table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the user-defined type
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the user-defined type
3	OBJECT_NAME	VARCHAR(30) or MVARCHAR(30)	Attribute name
4	TYPE_ID	INTEGER	ID of the user-defined type
5	META_TYPE	CHAR(1)	Type of the user-defined type: S: System-defined type A: Abstract data type

Number	Column name	Data type	Contents
6	ORDINAL_POSITION	SMALLINT	Order position
7	ENCAPSULATION_ LEVEL	VARCHAR(10)	Encapsulation level (PUBLIC, PRIVATE, or PROTECTED)
8	IS_NULLABLE	CHAR(3)	Column null value information YES: Null value allowed NO: Null values not allowed
9	DATA_TYPE	CHAR(24)	Data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table.
10	DATA_TYPE_CODE	SMALLINT	Data type code[1]
11	DATA_LENGTH_CODE	SMALLINT	Data length code[2] (Null value if the data type is BLOB, BINARY, or a user-defined type)
12	DATA_LENGTH	CHAR(7)	Data length stored right-justified in character format (blanks are used for leading zeros) (Null value if the data length is for BLOB, BINARY, or a user-defined type.)
13	LOB_LENGTH_CODE	CHAR(8)	BLOB attribute length code[3, 4] (Null value if the code is not for BLOB or BINARY.)
14	LOB_LENGTH	CHAR(20)	BLOB attribute length specification value stored right-justified in character format (blanks are used for leading zeros) Null value if the value is not for BLOB or BINARY.
15	LOB_LENGTH_TYPE	CHAR(1)	BLOB attribute length type (unit): K: K specified M: M specified G: G specified Blank: Default (Null value if the type is not BLOB.)
16	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type for an abstract data type attribute that has another abstract data type (Null value if the owner is for the system definition type.)

Number	Column name	Data type	Contents
17	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type for an abstract data type attribute that has another abstract data type (Null value if the name is for the system definition type.)
18	DATA_COMMENT	VARCHAR(255)	Comment (The initial value is a null value; null value is also used if there is no comment.)
19	NO_SPLIT	CHAR(1)	Whether or not NO SPLIT is specified: Y: Specified Null value: No specification

[1] For the specified data types and the values to be stored, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[2] For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

[3] The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

[4] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(24) SQL_TABLE_RESOURCES table

This table manages resource information used in tables. (Each row describes information on one resource.)

Table F-26 shows the contents of the SQL_TABLE_RESOURCES table.

Table F-26: SQL_TABLE_RESOURCES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used

Number	Column name	Data type	Contents
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	Type of the resource used: A: Abstract data type

(25) SQL_PLUGINS table

This table manages plug-in information. (Each row describes information on one plug-in.)

Table F-27 shows the contents of the SQL_PLUGINS table.

Table F-27: SQL_PLUGINS table contents

Number	Column name	Data type	Contents
1	PLUGIN_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Plug-in owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	PLUGIN_TYPE	CHAR(1)	Plug-in type: D: Data type plug-in I: Index type plug-in
4	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type or index type
5	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type or index type
6	CREATE_TIME	CHAR(14)	Plug-in creation time
7	PLUGIN_LIB_NAME	VARCHAR(255)	Library path name
8	PLUGIN_COMMENT	VARCHAR(255)	Comment (The initial value is a null value; null value is also used if there is no comment.)
9	PLUGIN_VERSION	VARCHAR(10)	Plug-in version (Null value if the plug-in is the initial version.)
10	PLUGIN_EXT_FUNC	VARCHAR(255)	Plug-in extended function code (information used in the system)

(26) SQL_PLUGIN_ROUTINES table

This table manages plug-in routine information. (Each row describes information on one plug-in routine.)

Table F-28 shows the contents of the SQL_PLUGIN_ROUTINES table.

Table F-28: SQL_PLUGIN_ROUTINES table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Routine owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	OPERATION_NAME	VARCHAR(255)	Operation name
4	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name[1]
5	N_PARAM	INTEGER	Number of parameters
6	TIMING_DESCRIPTOR	VARCHAR(30)	Timing descriptor
7	OPERATION_DESCRIPTOR	VARCHAR(255)	Operation modification information

[1] A plug-in routine is named in the following format:

'P' *function-name registration-date-and-time*

P

Code that indicates a function provided by a plug-in

function-name

The leading characters (maximum 15 characters) are truncated so that the specific name is within 30 characters.

registration-date-and-time

Indicates the year, month, hour, minute, and second with 14 characters.

(27) SQL_PLUGIN_ROUTINE_PARAMS table

This table manages plug-in routine parameter information. (Each row describes information on one parameter.)

Table F-29 shows the contents of the SQL_PLUGIN_ROUTINE_PARAMS table.

Table F-29: SQL_PLUGIN_ROUTINE_PARAMS table contents

Number	Column name	Data type	Contents
1	ROUTINE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner
2	PLUGIN_NAME	VARCHAR(30) or MVARCHAR(30)	Plug-in name
3	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name
4	PARAMETER_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter name
5	PARAMETER_MODE	CHAR(7)	Parameter I/O attribute: IN: Input mode OUT: Output mode INOUT: Input/output mode RETURNS: Return value attribute PICKUP: ROWID output attribute
6	PARAMETER_ DESCRIPTOR	VARCHAR(255)	Parameter modification information Parameter modification information specified with the plug-in IDL is held as a character string without changes. (Null value if no parameter modification information is specified)
7	SPECIFIC_BIND_ OPERATION_NAME	VARCHAR(30) or MVARCHAR(30)	Specific bind operation name (Null value if bind operation is not specified.)
8	PARAMETER_TYPE	CHAR(1)	Parameter mode: Blank: normal (data type that can be handled by SQL) I: Indicator N: New data C: Current data D: dbifb K: Index key inf P: Pointer R: rowid U: utlifb T: Pointer These are plug-in specific parameter modes, except normal.
9	PARAMETER_NO	INTEGER	Parameter specification order position for abstract data type functions

Number	Column name	Data type	Contents
10	DATA_TYPE	CHAR(24)	Parameter data type For details about the storage format, see the DATA_TYPE column in the SQL_COLUMNS table. (Null value if the parameter mode is D, K, P, R, U, or T.)
11	DATA_TYPE_CODE	SMALLINT	Parameter data code[1] (Null value if the parameter mode is D, K, P, R, U, or T.)
12	DATA_LENGTH_CODE	SMALLINT	Parameter data type definition length code[2] (Null value if the parameter mode is D, K, P, R, U, or T.)
13	DATA_LENGTH	CHAR(7)	Parameter data definition length stored right-justified in character format (blanks are used for leading zeros) (Null value if the parameter mode is D, K, P, R, U, or T.)
14	LOB_LENGTH_CODE	CHAR(8)	LOB column length code or BINARY column length code[3, 4] Null value if the parameter mode is normal and the data type is not BLOB or BINARY.
15	LOB_LENGTH	CHAR(20)	LOB column length specification value or BINARY column length specification value Stored in the character format, right-justified (higher-order 0s are left as blank spaces). Null value if the parameter mode is normal and the data type is not BLOB or BINARY.
16	LOB_LENGTH_TYPE	CHAR(1)	LOB column length type (unit): K: K specified M: M specified G: G specified Blank: Default (Null value if the parameter mode is normal and the data type is not BLOB or BINARY.)
17	UDT_OWNER	VARCHAR(30) or MVARCHAR(30)	Parameter data type owner (Null value if the data type is not a user-defined type.)
18	UDT_NAME	VARCHAR(30) or MVARCHAR(30)	Parameter data type name (Null value if the data type is not a user-defined type.)

Number	Column name	Data type	Contents
19	UDT_TYPE_ID	INTEGER	Parameter data type ID (Null value if the data type is not a user-defined type.)

[1] For the specified data types and the values to be stored, see *Table B-2 Data codes and data lengths set in the SQL Descriptor Area*.

[2] For the DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND types, precision and scale are each stored in 1 byte. In all other cases, size (number of characters for the NCHAR and NVARCHAR types) is stored in the 2-byte binary format. Note that the value is 0 for the BLOB and abstract data types.

[3] The specified column length is stored in binary format in 8 bytes, divided into 4-byte segments.

[4] SQL results are not subject to endian conversion, even for connection modes with different endians. Therefore, applications must handle the endian.

(28) SQL_INDEX_TYPES table

This table manages index type information. (Each row describes information on one index type.)

Table F-30 shows the contents of the SQL_INDEX_TYPES table.

Table F-30: SQL_INDEX_TYPES table contents

Number	Column name	Data type	Contents
1	INDEX_TYPE_ SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index type owner
2	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Index type name
3	INDEX_TYPE_ID	INTEGER	Index type ID
4	CREATE_TIME	CHAR(14)	Creation time
5	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Abstract data type owner
6	ADT_NAME	VARCHAR(30) or MVARCHAR(30)	Abstract data type name
7	N_FUNCTION	INTEGER	Number of abstract data type functions that can be used in an index-type-defined index

(29) SQL_INDEX_RESOURCES table

This table manages resource information used in indexes. (Each row describes information on one resource.)

Table F-31 shows the contents of the SQL_INDEX_RESOURCES table.

Table F-31: SQL_INDEX_RESOURCES table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the index definition table
2	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	Type of the resource used: I: Index type

(30) SQL_INDEX_DATATYPE table

This table manages target item information in indexes. (Each row describes information on one target item (one level).)

Table F-32 shows the contents of the SQL_INDEX_DATATYPE table.

Table F-32: SQL_INDEX_DATATYPE table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (index column name)
6	N_LEVEL	SMALLINT	Number of levels (number used to identify the name order of attributes constituting an abstract data type)

Number	Column name	Data type	Contents
7	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type
8	ADT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type
9	ADT_ATTR_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type attribute
10	ADT_ATTR_ID	SMALLINT	Attribute position

(31) SQL_INDEX_FUNCTION table

This table manages abstract data type function information used in indexes. (Each row describes information on one abstract data type function.)

Table F-33 shows the contents of the SQL_INDEX_FUNCTION table.

Table F-33: SQL_INDEX_FUNCTION table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name (index column name)
6	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner name of the abstract data type function
7	ADT_FUNCTION_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the abstract data type function (routine name)
8	ADT_FUNCTION_OBJECT_ID	INTEGER	Object ID of the abstract data type function

(32) SQL_TYPE_RESOURCES table

This table manages resource information used in user-defined types. (Each row describes information on one resource.)

Table F-34 shows the contents of the SQL_TYPE_RESOURCES table.

Table F-34: SQL_TYPE_RESOURCES table contents

Number	Column name	Data type	Contents
1	TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	User-defined type owner
2	TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	User-defined type name
3	BASE_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the resource used
4	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the resource used
5	BASE_TYPE	CHAR(1)	ID of the resource used A: Abstract data type

(33) SQL_INDEX_TYPE_FUNCTION table

This table manages abstract data type function information that can be used in an index that defines index types. (Each row describes information on one index type.)

Table F-35 shows the contents of the SQL_INDEX_TYPE_FUNCTION table.

Table F-35: SQL_INDEX_TYPE_FUNCTION table contents

Number	Column name	Data type	Contents
1	INDEX_TYPE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index type owner
2	INDEX_TYPE_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
3	ADT_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the abstract data type function
4	ADT_FUNCTION_NAME	VARCHAR(30) or MVARCHAR(30)	ID of the abstract data type function[1]
5	ADT_FUNCTION_OBJECT_ID	INTEGER	Object ID of the abstract data type function

[1] This is not a specific name.

(34) SQL_EXCEPT table

This table manages index exclusion key value information. (Each row describes information on the exclusion key group for one index.) This table manages one exclusion key value (exclusion value group for multicolumn indexes) in each row.

Table F-36 shows the contents of the `SQL_EXCEPT` table.

Table F-36: SQL_EXCEPT table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Index owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table that contains the index
3	INDEX_NAME	VARCHAR(30) or MVARCHAR(30)	Index name
4	INDEX_ID	INTEGER	Index ID
5	TABLE_ID	INTEGER	Table ID
6	EXCEPT_VALUE	VARCHAR(573) or MVARCHAR(573)	Contents of the exclusion key value The specified values for each column are delimited with a comma in a character format. (The initial value is a null value.)

(35) SQL_FOREIGN_SERVERS table

This table manages server DBMS information. One row is created for the information for one foreign server. (Each row describes information on one foreign server.)

If HiRDB External Data Access is not installed, this table is empty. However, if HiRDB External Data Access is installed and a database is created, and then HiRDB External Data Access is removed afterwards, the data in the table remains.

Table F-37 shows the contents of the `SQL_FOREIGN_SERVERS` table.

Table F-37: SQL_FOREIGN_SERVERS table contents

Number	Column name	Data type	Contents
1	FOREIGN_SERVER_NAME	VARCHAR(30) or MVARCHAR(30)	Foreign server name Null value after DROP SERVER is executed.[3]
2	FOREIGN_SERVER_ID	INTEGER	Foreign server ID
3	FOREIGN_SERVER_TYPE	VARCHAR(30)	Server type[1] HIRDB: HiRDB XDMRD: HiRDB on XDM DB2_UDB_OS390: DB2 Universal Database for OS/390 ORACLE: Oracle Null value after DROP SERVER is executed.[3]

Number	Column name	Data type	Contents
4	FOREIGN_SERVER_VERSION	VARCHAR(30)	Server version[1] Null value after DROP SERVER is executed.[3]
5	AUTHORIZATION_IDENTIFIER	VARCHAR(30) or MVARCHAR(30)	Foreign server owner Null value after DROP SERVER is executed.[3]
6	CREATE_TIME	CHAR(14)	Foreign server creation time (*YYYYMMDDHHMMSS*) Null value after DROP SERVER is executed.[3]
7	CHANGE_TIME	CHAR(14)	Foreign server definition change time (*YYYYMMDDHHMMSS*) Null value when a row is created and after DROP SERVER is executed.[3]
8	N_FOREIGN_TABLE	INTEGER	Number of tables defined in the foreign server
9	USING_BES	CHAR(8)	Name of the back-end server that accesses the foreign server.[2] Null value after DROP SERVER is executed.[3]

[1] The server type and server version of the foreign server accessed by HiRDB are set as follows:

DBMS product name	Server type	Server version
XDM/RD E2	XDMRD	6.0
HiRDB Version 5.0	HIRDB	5.0
HiRDB Version 6	HIRDB	6.0
HiRDB Version 7	HIRDB	6.0
Oracle8.1.5 (for the HP-UX version)	ORACLE	8.1.5
Oracle8.1.7 (for the AIX 5L version)	ORACLE	8.1.5
DB2 Universal Database for OS/390 Version 6	DB2_UDB_OS390	6.0

[2] If the name is less than 8 bytes when left justified, the remaining spaces are filled with blank spaces.

[3] If DROP SERVER is executed to reuse a foreign server ID, the row is not deleted and all columns except the one for a foreign server ID (FOREIGN_SERVER_ID) become null values. Note however that the number of defined tables (N_FOREIGN_TABLE) becomes 0. When CREATE SERVER is subsequently executed, the minimum value

among the unused foreign server IDs is assigned. If there are no unused foreign server IDs, *maximum-value* + 1 is assigned.

(36) SQL_USER_MAPPINGS table

This table manages mapping information between authorization identifiers on HiRDB and user IDs on the external server when an external server is accessed while the HiRDB External Data Access facility is being used (one row is for one mapping for one user on HiRDB).

If HiRDB External Data Access is not installed, this table is empty. However, if HiRDB External Data Access is installed and a database is created, and then HiRDB External Data Access is removed afterwards, the data in the table remains.

Table F-38 shows the contents of the `SQL_USER_MAPPINGS` table.

Table F-38: SQL_USER_MAPPINGS table contents

Number	Column name	Data type	Contents
1	`AUTHORIZATION_IDENTIFIER`	`VARCHAR(30)` or `MVARCHAR(30)`	HiRDB authorization identifier that is the conversion source of mapping (always `PUBLIC`)
2	`FOREIGN_SERVER_NAME`	`VARCHAR(30)` or `VARCHAR(30)`	Name of the external server
3	`FOREIGN_SERVER_ID`	`INTEGER`	External server ID
4	`CREATE_TIME`	`CHAR(14)`	User mapping creation date and time (*YYYYMMDDHHMMSS*)
5	`CHANGE_TIME`	`CHAR(14)`	User mapping definition modification date and time (*YYYYMMDDHHMMSS*)
6	`USER_ID`	`VARCHAR(30)` or `VARCHAR(30)`	User name at the external server

(37) SQL_IOS_GENERATIONS table contents

This table manages the generation information of HiRDB file system areas when the inner replica facility is used. (Each row describes information on one HiRDB file system area.)

If the HiRDB Staticizer Option is not installed, this table is empty. However, if a database is created with HiRDB Staticizer Option installed, and then HiRDB Staticizer Option is removed, any data set in the table remains.

Table F-39 shows the contents of the `SQL_IOS_GENERATIONS` table.

Table F-39: SQL_IOS_GENERATIONS table contents

Number	Column name	Data type	Contents
1	FILE_SYSTEM_NAME	VARCHAR(165)	HiRDB file system area name (absolute path name)
2	GENERATION_NUMBER	SMALLINT	Generation number
3	SERVER_NAME	CHAR(8)	Server name (BES or SDS)*
4	ORIGINAL_FILE_SYSTEM_NAME	VARCHAR(165)	Original HiRDB file system area name (absolute path name)

* Even when a dictionary table of a HiRDB/Parallel Server is used in a HiRDB/Single Server without any modification, the server name is not changed.

If the name is less than 8 characters when left justified, the remaining spaces are filled with blank spaces.

(38) SQL_TRIGGERS table contents

This table manages the information of the triggers that are inside a schema. (Each row describes information on one trigger.)

Table F-40 shows the contents of the SQL_TRIGGERS table.

Table F-40: SQL_TRIGGERS table contents

Number	Column name	Data type	Contents
1	TRIGGER_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Trigger owner
2	TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Trigger name
3	OBJECT_ID	INTEGER	Object ID
4	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the trigger is defined.
5	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the trigger is defined.

Number	Column name	Data type	Contents
6	TRIGGER_VALID	CHAR(1)	Trigger-enabling flag Y: Enabled N: Disabled Same value as the ROUTINE_VALID column of the SQL_ROUTINES table for the trigger action procedure
7	INDEX_VALID	CHAR(1)	Index-enabling flag Y: Enabled N: Disabled Same value as the INDEX_VALID column of the SQL_ROUTINES table for the trigger action procedure
8	ACTION_TIME	CHAR(1)	Trigger action timing A: AFTER B: BEFORE
9	EVENT	CHAR(1)	Trigger event type I: INSERT D: DELETE U: UPDATE
10	ACTION_TYPE	CHAR(1)	Trigger action unit R: ROW S: STATEMENT
11	OLD_ROW_NAME	VARCHAR(30) or MVARCHAR(30)	Old values correlation name (correlation name specified in OLD ROW) Null value if OLD ROW is not specified.
12	NEW_ROW_NAME	VARCHAR(30) or MVARCHAR(30)	New values correlation name (correlation name specified in NEW ROW) Null value if NEW ROW is not specified.
13	CREATE_TIME	VARCHAR(16)	Trigger definition creation time
14	ALTER_TIME	CHAR(14)	Trigger SQL object re-creation time Same value as the ALTER_TIME column of the SQL_ROUTINES table for the trigger action procedure Null value if a trigger SQL object is not re-created.
15	DEF_SOURCE_LEN	INTEGER	Trigger definition source length
16	SPECIFIC_NAME	VARCHAR(30) or MVARCHAR(30)	Specific name of the trigger action procedure

Number	Column name	Data type	Contents
17	N_UPDATE_COLUMNS	SMALLINT	Number of trigger event columns Null value for an UPDATE trigger for which no INSERT trigger, DELETE trigger, or trigger event column is specified.
18	REFERENCING_TABLE_ID	INTEGER	Table ID of the referencing table Null value for triggers that are not created by a referential constraint action.
19	REFERENCE_ACTION	CHAR(2)	Referential constraint operation type DC: ON DELETE CASCADE UC: ON UPDATE CASCADE Null value for triggers that are not created by a referential constraint action.
20	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name of referential trigger Null value for triggers that are not created by a referential constraint action.

(39) SQL_TRIGGER_COLUMNS table contents

This table manages the list information of UPDATE trigger event columns. (Each row describes information on one trigger column.)

Table F-41 shows the contents of the SQL_TRIGGER_COLUMNS table.

Table F-41: SQL_TRIGGER_COLUMNS table contents

Number	Column name	Data type	Contents
1	TRIGGER_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Trigger owner
2	TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Trigger name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the trigger is defined.
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the trigger is defined.
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Column name specified for the column list

Number	Column name	Data type	Contents
6	TABLE_ID	INTEGER	ID of the table for which the trigger is defined.

(40) SQL_TRIGGER_DEF_SOURCE table contents

This table manages the source information of trigger definitions. (Each row describes information on one trigger definition source.)

Table F-42 shows the contents of the SQL_TRIGGER_DEF_SOURCE table.

Table F-42: SQL_TRIGGER_DEF_SOURCE table contents

Number	Column name	Data type	Contents
1	TRIGGER_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Trigger owner
2	TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Trigger name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the trigger is defined.
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the trigger is defined.
5	SOURCE_NO	INTEGER	Definition source serial number
6	DEF_SOURCE	VARCHAR(32000) or MVARCHAR(32000)	Definition source (excluding SQL compile options and WITH PROGRAM)

(41) SQL_TRIGGER_USAGE table contents

This table manages the resource information being referenced inside trigger action conditions. (Each row describes information on one resource name being referenced in a trigger action condition.)

Table F-43 shows the contents of the SQL_TRIGGER_USAGE table.

Table F-43: SQL_TRIGGER_USAGE table contents

Number	Column name	Data type	Contents
1	TRIGGER_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Trigger owner
2	TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Trigger name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the trigger is defined.
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the trigger is defined.
5	BASE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the resource being used
6	BASE_TABLE	VARCHAR(30) or MVARCHAR(30)	Table name of the resource being used Null value if the type of the resource being used is F (function).
7	BASE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the resource being used (specific name or column name)
8	BASE_TYPE	CHAR(1)	Type of resource being used F: Function C: Column name
9	TABLE_ID	INTEGER	Table ID Null value if the type of the resource being used is F (function).
10	BASE_ID	INTEGER	ID of the resource being used (object ID or column ID)

(42) SQL_PARTKEY table contents

This table manages the partitioning key information of matrix-partitioned tables. (Each row describes information on one partitioning key.)

If HiRDB Advanced Partitioning Option is not installed, this table is empty. However, if HiRDB Advanced Partitioning Option is installed and a database is created, and then HiRDB Advanced Partitioning Option is removed afterwards, the data in the table remains.

Table F-44 shows the contents of the SQL_PARTKEY table.

Table F-44: SQL_PARTKEY table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	KEY_NO	SMALLINT	Partitioning key number (dimension number 1 or 2)
4	KEY_NAME	VARCHAR(30) or MVARCHAR(30)	Partitioning key column name
5	COLUMN_ID	SMALLINT	Partitioning key column ID
6	N_DIVISION	SMALLINT	Number of divisions inside the key
7	HASH_KEY_NO	SMALLINT	Sequence number in hash key column Null value for dimensions of boundary value partitioning.

(43) SQL_PARTKEY_DIVISION table contents

This table manages the information on the partitioning condition values for a matrix-partitioned table. (Each row describes information on one partitioning condition value.)

If HiRDB Advanced Partitioning Option is not installed, this table is empty. However, if HiRDB Advanced Partitioning Option is installed and a database is created, and then HiRDB Advanced Partitioning Option is removed afterwards, the data in the table remains.

Table F-45 shows the contents of the SQL_PARTKEY_DIVISION table.

Table F-45: SQL_PARTKEY_DIVISION table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner

Number	Column name	Data type	Contents
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	KEY_NO	SMALLINT	Partitioning key number (dimension number 1 or 2)
4	IN_DIM_NO	SMALLINT	Serial number inside a partitioning key
5	DCVALUES	VARCHAR(255) or MVARCHAR(255)	Partitioning condition value (the specified partitioning condition value is stored in the character format). Null value for the last boundary value within a partitioning key and for dimensions of hash partitioning.

(44) SQL_AUDITS table contents

This table manages audit target information. (Each row describes information on one event for one object or user.)

Table F-46 shows the contents of the `SQL_AUDITS` table.

Table F-46: SQL_AUDITS table contents

Number	Column name	Data type	Contents
1	EVENT_TYPE	VARCHAR(30)	Name of the event type[1] specified by the `CREATE AUDIT FOR` operation type or `'ANY'`.
2	EVENT_SUBTYPE	VARCHAR(30)	Event sub-type name [2] or `'ANY'` Null value if `CREATE AUDIT FOR ANY` is specified.
3	OBJECT_TYPE	VARCHAR(30)	Type of object specified by the `CREATE AUDIT` selection option.[3] Null value if no object is specified or if the HiRDB version is earlier than 07-03.
4	OBJECT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of object specified by the `CREATE AUDIT` selection option. Null value if no object is specified or if the HiRDB version is earlier than 07-03.
5	OBJECT_NAME	VARCHAR(30) or MVARCHAR(30)	Name of object specified by the `CREATE AUDIT` selection option. Null value if no object is specified or if the HiRDB version is earlier than 07-03.

Number	Column name	Data type	Contents
6	USER_NAME	VARCHAR(30) or MVARCHAR(30)	Authorization identifier of event executor (null value).
7	ANY_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER ANY is specified: Y: Specified N: Not specified
8	SUCCESSFUL_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER SUCCESSFUL is specified: Y: Specified N: Not specified
9	UNSUCCESSFUL_ANY_VALID	CHAR(1)	Whether or not CREATE AUDIT WHENEVER UNSUCCESSFUL is specified: Y: Specified N: Not specified
10	AUDIT_TYPE	CHAR(1)	Acquisition information type: E: CREATE AUDIT AUDITTYPE EVENT is specified A: CREATE AUDIT AUDITTYPE ANY is specified Null value: CREATE AUDIT AUDITTYPE PRIVILEGE is specified or AUDITTYPE is omitted.

[1] The following event types are available:

SESSION, PRIVILEGE, DEFINITION, ACCESS, and UTILITY

[2] The following event sub-types are available:

CONNECT, AUTHORIZATION, GRANT, REVOKE, CREATE, DROP, ALTER, SELECT, INSERT, UPDATE, DELETE, PURGE, CALL, OPEN, LOCK, PDLOAD, PDRORG, and PDEXP

[3] The following object types are available:

ALIAS, FOREIGN INDEX, FOREIGN TABLE, FUNCTION, INDEX, PROCEDURE, SCHEMA, SERVER, TABLE, TRIGGER, DATA TYPE, USER MAPPING, VIEW, and LIST

(45) SQL_REFERENTIAL_CONSTRAINTS table contents

This table manages the corresponding conditions of referential constraints. (Each row describes information on one constraint.)

Table F-47 shows the contents of the `SQL_REFERENTIAL_CONSTRAINTS` table.

Table F-47: SQL_REFERENTIAL_CONSTRAINTS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
2	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Constraint owner
3	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint is defined
4	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint is defined
5	COLUMN_COUNT	SMALLINT	Number of columns in the foreign key
6	COLUMN_NAME	VARCHAR(527) or MVARCHAR(527)	Column names of the table containing the foreign key Enclose each column in quotation marks and link the columns with commas.
7	COLUMN_NO	VARCHAR(32)	Column IDs (16 IDs) of the table containing the foreign key*
8	R_OWNER	VARCHAR(30) or MVARCHAR(30)	Owner of the table to be referenced
9	R_TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table to be referenced
10	DELETE_RULE	CHAR(11)	Deletion rule (RESTRICT or CASCADE)
11	UPDATE_RULE	CHAR(11)	Update rule (RESTRICT or CASCADE)
12	CONSTRAINT_TIME	CHAR(14)	Date and time when the constraint was defined (*YYYYMMDDHHMMSS*)
13	CHECK_PEND	CHAR(1)	Type of check pending status C: Pending Null value: Non-pending

Number	Column name	Data type	Contents
14	DELETE_TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the trigger created by the action of the ON DELETE referential constraint (DRA*YYYYMMDDHHMMSS*th) Null value if no trigger is created by the action of the ON DELETE referential constraint.
15	UPDATE_TRIGGER_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the trigger created by the action of the ON UPDATE referential constraint (DRA*YYYYMMDDHHMMSS*th) Null value if no trigger is created by the action of the ON UPDATE referential constraint.
16	R_COLUMN_NAME	VARCHAR(527) or MVARCHAR(527)	Column names of the columns that make up the main key Enclose each column in quotation marks and link the columns with commas.
17	R_COLUMN_NO	VARCHAR(32)	Column IDs (16 IDs) of the columns that make up the main key*

* Endian conversion is not performed on the SQL results even if the connection modes have different endians. Therefore, when an application accesses the SQL results, the SQL must consider the endian and convert the endian if necessary.

(46) SQL_KEYCOLUMN_USAGE table contents

This table manages information on the columns that make up foreign keys. (Each row describes information on one column.)

Table F-48 shows the contents of the SQL_KEYCOLUMN_USAGE table.

Table F-48: SQL_KEYCOLUMN_USAGE table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Constraint owner
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined

Number	Column name	Data type	Contents
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the column for which the constraint was defined
6	COLUMN_ORDER	SMALLINT	Position of the column for which the constraint was defined

(47) SQL_TABLE_CONSTRAINTS table contents

This table manages information on integrity constraints found in a schemas. (Each row describes information on one integrity constraint.)

Table F-49 shows the contents of the SQL_TABLE_CONSTRAINTS table.

Table F-49: SQL_TABLE_CONSTRAINTS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Constraint owner
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
3	CONSTRAINT_TYPE	VARCHAR(30)	Constraint type FOREIGN KEY: Foreign key CHECK: Check constraint
4	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined
5	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined

(48) SQL_CHECKS table contents

This table manages information on check constraints. (Each row describes information on one check constraint.)

Table F-50 shows the contents of the SQL_CHECKS table.

Table F-50: SQL_CHECKS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Check constraint owner

Number	Column name	Data type	Contents
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined
5	CHK_SOURCE_LEN	INTEGER	Length of the check constraint search conditions
6	CHK_SOURCE	BINARY(2000000)	Check constraint search conditions
7	CREATE_TIME	CHAR(14)	Date and time when the search constraint was defined (*YYYYMMDDHHMMSS*)
8	CHECK_PEND2	CHAR(1)	Check pending status type C: Pending Null value: Non-pending
9	N_CHK_COLUMN	INTEGER	Number of constraint columns specified in the check constraint definition (number of duplicate exclusion columns)

(49) SQL_CHECK_COLUMNS table contents

This table manages information on the columns used by check constraints. (Each row describes information on one column used by one check constraint.)

Table F-51 shows the contents of the SQL_CHECK_COLUMNS table.

Table F-51: SQL_CHECK_COLUMNS table contents

Number	Column name	Data type	Contents
1	CONSTRAINT_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Check constraint owner
2	CONSTRAINT_NAME	VARCHAR(30) or MVARCHAR(30)	Constraint name
3	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Owner of the table for which the constraint was defined
4	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the table for which the constraint was defined

Number	Column name	Data type	Contents
5	COLUMN_NAME	VARCHAR(30) or MVARCHAR(30)	Name of the column used by the constraint

(50) SQL_DIV_TYPE table contents

This table manages information on partitioning keys in matrix partitioning tables that combine key range partitioning and hash partitioning. (Each row describes information on one partitioning key.)

Table F-52 shows the contents of the SQL_DIV_TYPE table.

Table F-52: SQL_DIV_TYPE table contents

Number	Column name	Data type	Contents
1	TABLE_SCHEMA	VARCHAR(30) or MVARCHAR(30)	Table owner
2	TABLE_NAME	VARCHAR(30) or MVARCHAR(30)	Table name
3	KEY_NO	SMALLINT	Partitioning key number (dimension number)
4	DIV_TYPE	CHAR(1)	Partitioning type in the dimension P: Boundary value partitioning F: FIX hash partitioning H: Flexible hash partitioning
5	HASH_NAME	VARCHAR(30) or MVARCHAR(30)	Hash function name "HASH1" "HASH2" "HASH3" "HASH4" "HASH5" "HASH6" "HASH0" Null value for dimensions without hash partitioning
6	N_DIV_COLUMN	SMALLINT	Number of partitioning columns in the dimension

(51) SQL_SYSPARAMS table contents

This table manages information about limits on the number of consecutive certification failures and password character strings. (Each row describes information on one setting item. *n* rows describes information on one limit on the number of consecutive certification failures or one password character string limit.) The SQL_SYSPARAMS table can be referenced only by owners with the DBA privilege and auditors.

Table F-53 shows the contents of the SQL_SYSPARAMS table.

Table F-53: SQL_SYSPARAMS table contents

Number	Column name	Data type	Contents
1	PARAM_KIND	VARCHAR(20)	Parameter type (CONNECTION_SECURITY)
2	FUNCTION_KEY	VARCHAR(20)	Function name CONNECT: Limit on the number of consecutive certification failures PASSWORD: Password character sting limit
3	PARAM_KEY	VARCHAR(20)	Specification item When the function name is CONNECT, the specification item is one of the following: PERMISSION_COUNT: Permitted number of consecutive certification failures LOCK_MINUTE: Account lock period (minutes) LOCK_MINUTE_CODE: Account lock period code When the function name is PASSWORD, the specification item is one of the following: MIN_LENGTH: Minimum number of allowed bytes USER_IDENTIFIER: Specification of authorization identifier prohibited SIMILAR: Specification of single character type prohibited
4	INT_VALUE	INTEGER	INT-type data value*
5	CHAR_VALUE	VARCHAR(30)	CHAR-type data value*

* The table below shows the values that are stored for the INT-type and CHAR-type data values.

PARAM_KEY setting value	Value specified in SQL	INT_VALUE	CHAR_VALUE
PERMISSION_COUNT	Constant	Constant	Constant
	No specification	2	2

PARAM_KEY setting value	Value specified in SQL	INT_VALUE	CHAR_VALUE
LOCK_MINUTE	Constant	Constant	Constant
	UNLIMITED	Null value	UNLIMITED
	No specification	1440	1440
LOCK_MINUTE_CODE	Constant	Constant	Constant
	UNLIMITED	Null value	UNLIMITED
	No specification	1000000	1000000
MIN_LENGTH	Constant	Constant	Constant
	No specification	8	8
USER_IDENTIFIER	RESTRICT	Null value	RESTRICT
	UNRESTRICT	Null value	UNRESTRICT
	No specification	Null value	RESTRICT
SIMILAR	RESTRICT	Null value	RESTRICT
	UNRESTRICT	Null value	UNRESTRICT
	No specification	Null value	RESTRICT

G. Functions provided by HiRDB

This appendix explains the following functions provided by HiRDB:

- Hash function for table partitioning
- Space conversion facility
- Function for conversion to a `DECIMAL` signed normalized number
- Function that sets the character code classification

Note that the Linux for AP8000 version of a client cannot use the functions provided by HiRDB.

G.1 Hash function for table partitioning

The hash function for table partitioning uses the partitioning key values to obtain the order of partitioning conditions that are specified for partitioning a table. If a UAP is executed using the hash function for table partitioning, the storage RDAREAs can be identified before data is stored in a table, even if the table is a hash-partitioned table. Because this function can identify each storage RDAREA, you can use the function for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when determining the hash function and partitioning key for hash partitioning
- To create an input data file for each RDAREA when loading data to a hash-partitioned table in units of RDAREAs concurrently using the database load utility

(1) Prerequisites for using the hash function for table partitioning

The following describes the prerequisites for using the hash function for table partitioning.

(a) Program language

When the hash function for table partitioning is used to create a UAP, the UAP can be written in either C or C++.

(b) Execution environment

The hash function for table partitioning can be executed on a server machine in which a HiRDB server or HiRDB client has been installed.

However, certain combinations of a HiRDB server operating system and a HiRDB client operating system can produce incorrect results when the function is executed with a HiRDB client.

Table G-1 shows the UAP execution conditions in the HiRDB client.

Table G-1: Execution conditions in the HiRDB client

HiRDB server operating system	HiRDB client operating system	
	HP-UX, Solaris, and AIX 5L	Linux and Windows
HP-UX, Solaris, and AIX 5L	E	—
Linux and Windows	—	E

E: Can be executed.

— : Errors occur in the partitioning condition specification order or the partitioning key sequence numbers because the operating systems use a different byte order.

(2) Creating and executing UAPs that use the hash function for table partitioning

Create and execute a UAP according to the following procedure:

1. Create a source program.
2. Compile and link the source program.
3. Execute the load module.

(a) Creating a source program

Specify function calling of the hash function for table partitioning in the source program written in C or C++. Because the hash function for table partitioning is presented in a shared library format, link the source program to use the function.

When the hash function for table partitioning is used, the distributed header files must be included when the source program is created. Include all header files required by the hash function for table partitioning. For details about the header files required by the hash function for table partitioning, see *(3) Function details*.

(b) Compiling and linking the source program

Compile and link the source program in a server machine that has either the HiRDB server or HiRDB client installed.

If SQL statements are embedded in the source program, preprocessing must be executed before compiling and linking.

For details about compiling, linking, and preprocessing, see *8. Preparation for UAP Execution*.

Compiling and linking in a UNIX environment (HiRDB server)

Specification examples for compiling and linking a source program in the HiRDB server are shown as follows:

Example (C)

When the source filename is `sample.c` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
cc -l $PDDIR/include sample.c -L$PDDIR/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
cc +DD64 -l $PDDIR/include sample.c -L$PDDIR/client/lib
-l sqlauxf64
```

Example (C++)

When the source filename is `sample.C` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
CC -l $PDDIR/include sample.C -L$PDDIR/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
CC +DD64 -l $PDDIR/include sample.C -L$PDDIR/client/lib
-l sqlauxf64
```

Compiling and linking in a UNIX environment (HiRDB client)

Shown below are specification examples for compiling and linking a source program in the HiRDB client.

Example (C)

When the source filename is `sample.c` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
cc -l /HiRDB/include sample.c -L/HiRDB/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
cc +DD64 -l /HiRDB/include sample.c -L/HiRDB/client/lib
-l sqlauxf64
```

Note

The underline indicates the HiRDB client's installation directory.

Example (C++)

When the source filename is `sample.C` and the name of the executable file is not specified

- Creating a UAP that is run in 32-bit mode:

```
CC -l /HiRDB/include sample.C -L/HiRDB/client/lib -l
sqlauxf
```

- Creating a UAP that is run in 64-bit mode:

```
CC +DD64 -l /HiRDB/include sample.C -L/HiRDB/client/lib
-l sqlauxf64
```

Note

The underline indicates the HiRDB client's installation directory.

Compiling and linking in a Windows environment (HiRDB server)

For a source program written in C, use an ANSI-C-compliant compiler to compile the program. For a source program written in C++, use a C++-compliant compiler to compile the program.

If you are using Microsoft Visual C++ Version 1.0 to compile and link the source program, select **Set Project** from the **Options** menu for the option settings.

If you are using Microsoft Visual C++ Version 2.0 to compile and link the source program, select **Set** from the **Project** menu for the option settings.

Table G-2 shows the items to be set in the HiRDB server with **Set Project** or **Set**.

Table G-2: Items to be set in the HiRDB server with Set Project or Set

Item	Category	Category setting	Setting value
Compiler	Code generation	Structure member alignment	8 bytes
		Run time library to be used	Multi-thread
	Processor	Include file path	`\HiRDB\client\include`
Linker	Input	Library	`\HiRDB\client\lib\pdsqlauxf.lib`

Note

The underline indicates the HiRDB client's installation directory.

Compiling and linking in a Windows environment (HiRDB client)

For a source program written in C, use an ANSI-C-compliant compiler to compile the program. For a source program written in C++, use a C++-compliant compiler to compile the program.

If you are using Microsoft Visual C++ Version 1.0 to compile and link the source program, select **Set Project** from the **Options** menu for the option settings.

If you are using Microsoft Visual C++ Version 2.0 to compile and link the source program, select **Set** from the **Project** menu for the option settings.

Table G-3 shows the items in the HiRDB client to be set with **Set Project** or **Set**.

Table G-3: Items to be set in the HiRDB client with Set Project or Set

Item	Category	Category setting	Setting value
Compiler	Code generation	Structure member alignment	8 bytes
		Run time library to be used	Multi-thread
	Processor	Include file path	`\`*HiRDB*`\include`
Linker	Input	Library	`\`*HiRDB*`\lib\pdsqlauxf.lib`

Note

The underline indicates the HiRDB client's installation directory.

(3) Function details

(a) Required input information

To call the hash function for table partitioning, obtain the information for items 1 through 8, described as follows, and set the information to arguments.

1. Hash function name specified for partitioning
2. Number of columns specified in partitioning keys
3. Specification order of partitioning keys, data type codes, and data length codes
4. Number of table partitions
5. Data values stored in partitioning keys
6. Double-byte space character for each national character code type used in the HiRDB server.
7. Space conversion level
8. Whether or not to use the facility for conversion to a DECIMAL signed normalized number

Items 1 through 4 correspond to the following sections in the `CREATE TABLE` statement:

```
CREATE TABLE TABLE1 ( C1  CHAR(10) NOT NULL,
                                 3
                      C2  NVARCHAR(4) NOT NULL,
                                 3
                      C3  DEC(5,2) NOT NULL,
                                 3
                     C4  INT,

                      C5  SMALLINT NOT NULL )
                                 3
       FIX HASH  HASH6  BY  C1,C3,C5,C2
                   1           2,3
           in ( RU01,RU02,RU03,RU04,RU05,RU06 )
                             4
```

If the table is already defined, information for items 1 through 4 can be obtained by retrieving the dictionary table. For examples of dictionary table retrieval, see *(6) Retrieval from dictionary tables (for hash partitioning)*.

For details about the space conversion level (space conversion facility) and the facility for conversion to a DECIMAL signed normalized number, see the *HiRDB Version 8 System Operation Guide*.

(b) Specification configuration

Details about the hash function for table partitioning are explained as follows:

Description

Provides an overview of the function.

Header files

Explains the headers that are necessary for using the hash function for table partitioning.

Format

Explains the actual specification format.

Arguments

Explains the arguments specified in the format.

Return value

Explains the return value types (specified as data types) of the hash function for table partitioning.

(c) p_rdb_dbhash hash function for table partitioning

Description

This function obtains the partitioning condition specification order (1 to number of table partitions) in which the partitioning keys are stored, or the partitioning key sequence numbers. If the function does not terminate normally, an incorrect value is obtained for the partitioning condition specification order.

If the partitioning condition specification order is obtained from multiple rows, the partitioning key data must be changed for each of those rows before the hash function for table partitioning is called. In this case, only those arguments that contain data values for partitioning keys must be changed; all other arguments do not need to be changed.

For details about how to determine the partitioning condition specification order from the partitioning key sequence numbers, see *(7) Retrieval from dictionary tables (for matrix partitioning)*.

Header files

`#include<pddbhash.h>`

This header file must be specified when the hash function for table partitioning is used.

`#include<pdbsqlda.h>`

This header file should be specified when a macro that begins with `PDSQL_` is used to set data type codes for the partitioning keys. This header file can be omitted when the data type codes to be set are retrieved from the dictionary table.

Format

```
int p_rdb_dbhash(short           hashcode,
                 short           ncol,
                 p_rdb_collst_t  *collst,
                 p_rdb_dadlst_t  *dadlst,
                 unsigned int    ndiv,
                 unsigned char   ncspace[2],
                 int             flags,
                 int             *rdno);
```

Arguments

`hashcode` (input)

Specifies a hash function code that corresponds to a hash function name. For details on hash function codes, see *(4)(a) Hash function codes*.

`ncol` (input)

Specifies the number of columns that were specified as partitioning keys when the table was defined.

`collst` (input)

Specifies a pointer to a partitioning key list. A partitioning key list is a structure that consists of the data type code and data size code of a partitioning key, and is an area in which partitioning keys are continuously listed. For details on the partitioning key list, see *(4)(b) Partitioning key list*.

You can obtain the data type code and data size code of a partitioning key by retrieving them from a dictionary table. For examples of dictionary table retrieval, see *(6) Retrieval from dictionary tables (for hash partitioning)*.

`dadlst` (input)

Specifies the pointer to the data address list. The data address list is a structure composed of the addresses to the data storage areas for partitioning keys and is allocated as a contiguous area for all partitioning keys. For details, see *(4)(c) Data address list*.

`ndiv` (input)

Specifies the number of partitions in hash partitioning.

`ncspace` (input)

Specifies the double-byte space character for each national character code type used in the HiRDB server. The character is specified in a two-byte area. When the data type of a partitioning key is `NVARCHAR`, this argument is used to remove spaces that follow character strings before hashing is executed. This argument is also used for space conversion for the partitioning key value (`NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`) when space conversion level 1 or 3 is specified in the `flags` argument.

An error results if the area specified in the `ncspace` argument contains no space character in the following cases:

- Partitioning key is `NVARCHAR`.
- Space conversion level 1 or 3 is specified in `flags` and the partitioning key is `NCHAR`, `NVARCHAR`, `MCHAR`, or `MVARCHAR`.

Table G-4 lists the double-byte space characters that can be specified in `ncspace`.

Table G-4: Double-byte space characters specified in ncspace

Character code type specified in pdsetup[2]	**ncspace**	
	ncspace[0]	**ncspace[1]**
`sjis` (shift JIS kanji code)	`0x81`	`0x40`
`Chinese` (EUC Chinese kanji code)	`0xA1`	`0xA1`

Character code type specified in pdsetup[2]	ncspace	
	ncspace[0]	ncspace[1]
ujis (EUC Japanese kanji code)	0xA1	0xA1
lang-c (single-byte character code)[1]	0x00	0x00
Unicode (UTF-8)[3]	0x00	0x00
Default value (sjis for HP-UX)	0x81	0x40
Default value (sjis for AIX 5L)	0x81	0x40
Default value (ujis for Solaris)	0xA1	0xA1
Default value (ujis for Linux)	0xA1	0xA1
Default value (sjis for Windows)	0x81	0x40

[1] If the character code type is lang-c, NCHAR, NVARCHAR, MCHAR, or MVARCHAR cannot be used for the column data type.

[2] For a Windows environment, specify the space character code of the character code type that was specified in the pdntenv command.

[3] NCHAR or NVARCHAR cannot be used for the column data type.

flags (input)

Specifies the flag according to the space conversion level and the facility for conversion to a DECIMAL signed normalized number (this argument is required even if these facilities are not used). For details about the space conversion level (space conversion facility) and the facility for conversion to a DECIMAL signed normalized number, see the *HiRDB Version 8 System Operation Guide*.

The following table shows the values for flags:

HiRDB operating environment		Value of flags
Space conversion level*	Omitted	p_rdb_FLG_SPLVL_0
	0	
	1	p_rdb_FLG_SPLVL_1
	3	p_rdb_FLG_SPLVL_3

HiRDB operating environment		Value of flags
Facility for conversion to a DECIMAL signed normalized number	Omitted	p_rdb_FLG_DECNRM_N
	N	
	Y	p_rdb_FLG_DECNRM_Y

* If the character code for the HiRDB server is Unicode (UTF-8), spaces must be converted before this function is executed. Therefore, specify only p_rdb_FLG_SPLVL_0 for the flags value.

rdno (output)

Sets the partitioning condition specification order (1 to number of table partitions) or the partitioning key sequence numbers.

Return values

data type: int

p_rdb_RC_RTRN(0)

Normal termination.

p_rdb_RC_ERRHASH(-1)

Invalid hash function code (p_rdb_HASH1 to p_rdb_HASH6, p_rdb_HASH0, p_rdb_HASHA to p_rdb_HASHF).

p_rdb_RC_ERRNCOL(-2)

Partitioning key count error (1 to p_rdb_MXDCL).

p_rdb_RC_ERRCLST(-3)

Area error for partitioning key data type or data length.

p_rdb_RC_ERRCTYP(-31)

Invalid data type for partitioning key.

p_rdb_RC_ERRCLEN(-32)

Invalid data type for partitioning key.

p_rdb_RC_ERRDLST(-4)

Area error for data address.

p_rdb_RC_ERRDADR(-41)

Data address not set.

p_rdb_RC_ERRDLEN(-42)

Actual data length is shorter than character length limit for hash function.

p_rdb_RC_ERRNDIV(-5)

Table partition count error (1 to p_rdb_MNCND)

p_rdb_RC_ERRRADR(-6)

Storage area for partitioning condition specification order or partitioning key sequence numbers is not set.

p_rdb_RC_ERRNCSC(-7)

Area for double-byte space character is not set.

Notes

1. If the partitioning key is NCHAR, NVARCHAR, MCHAR, or MVARCHAR, the value of rdno may be invalid unless an appropriate value corresponding to the space conversion level is specified in flags.
2. If the partitioning key is NCHAR, NVARCHAR, MCHAR, or MVARCHAR, and 1 or 3 is specified for the space conversion level, perform one of the following:
   - Use the setlocale function to set an appropriate locale for the LC_CTYPE category or the LC_ALL category.
   - Call the p_rdb_set_lang function.

   Operation is not guaranteed if the character code type of the key value and the locale specified by the setlocale function or p_rdb_set_lang function contradict each other. If this function is called from a Windows UAP, Linux UAP with SJIS character codes type, or UAP with CHINESE character codes type, the p_rdb_set_lang function is used instead of the setlocale function. For details about the p_rdb_set_lang function, see *G.4 Character code type specification function*.
3. If the partitioning key value is DECIMAL, INTERVAL YEAR TO DAY, or INTERVAL HOUR TO SECOND, the value of rdno may be invalid unless an appropriate value is specified in flags which corresponds to the facility for conversion to a DECIMAL signed normalized number.
4. When using the hash function for table partitioning with a HiRDB client version earlier than 05-05, you cannot use the space conversion level in flags or the facility for conversion to a DECIMAL signed normalized number. In this case, the function ignores flags and assumes that the facility for conversion to a DECIMAL signed normalized number is omitted. To use the flags specification, either execute the function at the HiRDB server or at a HiRDB client with version 05-05 or later.
5. If the character code for the HiRDB server is Unicode (UTF-8), this function does not convert spaces. The partitioning key value to be specified for

`dadlst` must be converted beforehand using the space conversion function `p_rdb_conv_space_utf8`.

(4) Data types and macros

(a) Hash function codes

Table G-5 shows the hash function codes that correspond to the hash functions specified in `CREATE TABLE` or `ALTER TABLE`.

Table G-5: Hash function codes for hash functions

Hash function name	Hash function code (value)
`HASH1`(when hash function name is omitted)	`p_rdb_HASH1(1)`
`HASH2`	`p_rdb_HASH2(2)`
`HASH3`	`p_rdb_HASH3(3)`
`HASH4`	`p_rdb_HASH4(4)`
`HASH5`	`p_rdb_HASH5(5)`
`HASH6`	`p_rdb_HASH6(6)`
`HASH0`	`p_rdb_HASH0(100)`
`HASHA`	`p_rdb_HASHA(101)`
`HASHB`	`p_rdb_HASHB(102)`
`HASHC`	`p_rdb_HASHC(103)`
`HASHD`	`p_rdb_HASHD(104)`
`HASHE`	`p_rdb_HASHE(105)`
`HASHF`	`p_rdb_HASHF(106)`

(b) Partitioning key list

The partitioning key list is a structure composed of data type codes and data length codes for partitioning keys, and is allocated a contiguous area for all partitioning keys. Table G-6 shows the area for setting partitioning keys. If there are multiple partitioning keys, the area must be specified as an array consisting of all columns specified as partitioning keys.

Table G-7 lists the data type codes and the data length codes.

Table G-6: Area for setting partitioning keys

Data type	Data type details	Explanation

p_rdb_collst_t	struct p_rdb_TG_collst {
	    unsigned short datatype ;
	    short datalen ;
	} p_rdb_collst_t ;
		Data type code
		Data length code

Table G-7: Data type codes and data length codes

Data type	Data type code	Data length code
INTERVAL YEAR TO DAY	PDSQL_YEARTODAY	8 × 256
INTERVAL HOUR TO SECOND	PDSQL_HOURTOSEC	6 × 256
DATE	PDSQL_DATE	4
TIME	PDSQL_TIME	3
TIMESTAMP[(*p*)]	PDSQL_TIMESTAMP	7 + ↑ *p*/2 ↑ (0 is assumed if *p* is omitted.)
MVARCHAR(*n*)	PDSQL_MVARCHAR	*n*
MCHAR[(*n*)]	PDSQL_MCHAR	*n* (default value is 1)
NVARCHAR(*n*)	PDSQL_NVARCHAR	*n*
NCHAR[(*n*)]	PDSQL_NCHAR	*n* (default value is 1)
VARCHAR(*n*)	PDSQL_VARCHAR	*n*
CHAR[(*n*)]	PDSQL_CHAR	*n* (default value is 1)
FLOAT	PDSQL_FLOAT	8
SMALLFLT	PDSQL_SMALLFLT	4
DECIMAL[(*p*[,*q*])]	PDSQL_DECIMAL	*p* × 256 + *q* (default values are 15 for *p* and 0 for *q*)
INTEGER	PDSQL_INTEGER	4
SMALLINT	PDSQL_SMALLINT	2

(c) Data address list

The data address list is a structure composed of the addresses to the data storage areas for partitioning keys, and is allocated as a contiguous area for all partitioning keys. Table G-8 shows the area for setting the data address of a partitioning key. If there are multiple partitioning keys, the area must be specified as an array consisting of all columns specified as partitioning keys.

Specify the area in binary format. For details about the binary format, see the *HiRDB Version 8 Command Reference* manual.

Table G-8: Area for setting the data address of a partitioning key

Data type	Data type details	Explanation

`p_rdb_dadlst_t`	`struct p_rdb_TG_dadlst {`
	`    unsigned char * dataaddr ;`
	`}  p_rdb_dadlst_t ;`
		Address to data area

Notes common to all data types

- Convert the real data in the data address list to the data type format defined in the column.
- The boundaries of the real data area for the data address list do not have to be adjusted.

Notes about data types DECIMAL, INTERVAL YEAR TO DAY, and INTERVAL HOUR TO SECOND

- For positive values, use `C` or `F` in the sign section of the real data in the data address list. If `Y` is specified for the facility for conversion to a DECIMAL signed normalized number, `A` and `E` are also available.
- For negative values, use `D` in the sign section of the real data in the data address list. If `Y` is specified for the facility for conversion to a DECIMAL signed normalized number, `B` is also available.

Notes about data types CHAR, NCHAR, and MCHAR

- For `CHAR` and `MCHAR`, pad the data area of the data address list with single-byte space characters up to the defined length.
- For `NCHAR`, pad the data area of the data address list with double-byte space characters up to the defined length. The double-byte space characters must be of the character code that was specified at HiRDB server setup.
- The data area of the data address list must be specified with the character codes used by the HiRDB server.

Notes about data types VARCHAR, NVARCHAR, and MVARCHAR

- For the real length section in the data area of the data address list, use bytes instead of character string length to indicate the data length.
- If the real length of the data area for the data address list is less than the defined length of the partitioning key list, do not pad the character string that

follows.

- Specify character codes used by the HiRDB server in the data area of the data address list.

(d) Macros for maximum values

Table G-9 lists the macros for maximum values.

Table G-9: Macros for maximum values

Macro name	Description (value)
p_rdb_MXDCL	Maximum number of partitioning key columns (16)
p_rdb_MNCND	Maximum number of table partitions (1024)

(5) Coding examples

A partial coding example that uses C to describe hash partitioning is shown below. Use this coding example by customizing it to suit the user needs. However, because this example does not include error handling during SQL statement execution, code error handling as needed. For details about error handling, see *3.6 SQL error identification and corrective measures*.

(a) Declaration section

```
/
***************************************************************
*/
/* ALL RIGHTS RESERVED. COPYRIGHT (C) 1999,2000, HITACH, LTD.  */
/* LICENSED MATERIAL OF HITACHI,LTD.                           */
/* Sample Program that Uses the Hash Function for Table
Partitioning                                                   */
/
***************************************************************
*/
#include <stdio.h>
#include <string.h>
#include <pdbsqlda.h>
#include <pddbhash.h>

union data_area {                      /* Data storage area */
    short data_smallint ;
    int data_int ;
    unsigned char data_dec[15] ;
    float data_smallflt ;
    double data_float ;
    unsigned char data_char[255] ;
    struct {
        short length ;
```

```
        unsigned char data[255] ;
    } data_varchar ;
    unsigned char data_date[4] ;
    unsigned char data_time[3] ;
    unsigned char data_timestamp[10] ;
    unsigned char data_iytd[5] ;
    unsigned char data_ihts[4] ;
} ;

void print_data(short , p_rdb_collst_t * , union data_area *) ;


/
***************************************************************
*/
/* Main Function                                               */
/
***************************************************************
*/
int main(int argc , char *argv[])
{

  short           hashcode ;        /* Hash function code */
  short           ncol ;            /* Number of partitioning key columns */
  p_rdb_collst_t  collst[p_rdb_MXDCL] ;/* Partitioning key list      */
  p_rdb_dadlst_t  dadlst[p_rdb_MXDCL] ;/* Data address list          */
  union data_area data[p_rdb_MXDCL] ;  /* Data storage area          */
  unsigned int    ndiv ;               /* Number of storage RDAREAs  */
  unsigned char   ncspace[2] ;      /* Space code for each national character
code
                                       type */
  int             flags ;           /* Enhancement flag */
  int           rdno ;          /* Partitioning condition specification order */
  int             rc ;              /* Return value */
  short           i, j, k ;         /* Counter variables */
  struct rdarea {                   /* RDAREA list */
  int             rdareaid ;
  char            rdareaname[31] ;
  } rdarealst [p_rdb_MNCND] ;

  EXEC SQL BEGIN DECLARE SECTION ;
    struct {                   /* Embedded variable for hash function name */
      short length ;
      char data[9] ;
    } xhashname ;
    short xncol ;          /* Embedded variable for number of partitioning key
columns */
```

```
    short xndiv ;          /* Embedded variable for number of table partitions */
    short xdatatype ;      /* Embedded variable for data type code */
    short xdatalen ;       /* Embedded variable for data length code */
    struct {               /* Embedded variable for storage RDAREA name */
      short length ;
      char data[31] ;
    } xrdname ;
  EXEC SQL END DECLARE SECTION ;

  EXEC SQL CONNECT ;
```

(b) Settings for the data storage area and space code for national character codes

```
for  (k = 0 ; k < p_rdb_MXDCL ; k ++) {
dadlst[k].dataaddr = (unsigned char *)&data[k] ;
}
ncspace[0] = 0x81 ;                  /* Space code */
ncspace[1] = 0x40 ;                  /* Example of shift JIS kanji code */
flags = 0 ;
```

(c) Settings for flags

```
/
*************************************************************
**/
/* (a) Specifying explicitly                                   */
/*   1 specified for space conversion level and Y for facility */
/* for conversion to a DECIMAL signed normalized number        */
/
*************************************************************
**/
flags=p_rdb_FLG_SPLVL_1+p_rdb_FLG_DECNRM_Y;
```

(d) Settings for the hash function name, number of partitioning key columns, and number of storage RDAREAs

```
/*********************************************************/
/*(a)Setting values with codes                           */
/*********************************************************/
hashcode = p_rdb_HASH6 ;            /* When HASH6 is specified */
ncol = 4 ;                          /* For partitioning with 4 columns */
ndiv = 6 ;                          /* For 6 partitions */

/*********************************************************/
/* (b) Retrieving values from the dictionary table       */
/*********************************************************/
  EXEC SQL
      select  HASH_NAME,
```

```
          value(N_DIV_COLUMN,1) ,
          N_RDAREA
        into  :xhashname , :xncol, :xndiv
        from MASTER.SQL_TABLES
        where TABLE_SCHEMA=USER
          and TABLE_NAME='TABLE1' ;

xhashname.data[xhashname.length] = '\0' ;
if (strcmp(xhashname.data,"HASH1") == 0) {
    hashcode=p_rdb_HASH1 ;                        /* HASH1 setting */
} else if (strcmp(xhashname.data,"HASH2") == 0) {
    hashcode=p_rdb_HASH2 ;                        /* HASH2 setting */
} else if (strcmp(xhashname.data,"HASH3") == 0) {
    hashcode=p_rdb_HASH3 ;                        /* HASH3 setting */
} else if (strcmp(xhashname.data,"HASH4") == 0) {
    hashcode=p_rdb_HASH4 ;                        /* HASH4 setting */
} else if (strcmp(xhashname.data,"HASH5") == 0) {
    hashcode=p_rdb_HASH5 ;                        /* HASH5 setting */
} else if (strcmp(xhashname.data,"HASH6") == 0) {
    hashcode=p_rdb_HASH6 ;                        /* HASH6 setting */
} else if (strcmp(xhashname.data,"HASHA") == 0) {
    hashcode=p_rdb_HASH0 ;                        /* HASH0 setting */
} else if (strcmp(xhashname.data,"HASHA") == 0) {
    hashcode=p_rdb_HASHA ;                        /* HASHA setting */
} else if (strcmp(xhashname.data,"HASHB") == 0) {
    hashcode=p_rdb_HASHB ;                        /* HASHB setting */
} else if (strcmp(xhashname.data,"HASHC") == 0) {
    hashcode=p_rdb_HASHC ;                        /* HASHC setting  */
} else if (strcmp(xhashname.data,"HASHD") == 0) {
    hashcode=p_rdb_HASHD ;                        /* HASHD setting  */
} else if (strcmp(xhashname.data,"HASHE") == 0) {
    hashcode=p_rdb_HASHE ;                        /* HASHE setting  */
} else if (strcmp(xhashname.data,"HASHF") == 0) {
    hashcode=p_rdb_HASHF ;                        /* HASHF setting  */
} else {
    /* Add when a hash function is added in the future. */
}
ncol = xncol ;
ndiv = xndiv ;

/****************************************************/
/* Displaying table definition information          */
/****************************************************/
printf("Hash function code:%d\n",hashcode);
printf("Number of partitioning key columns:%d\n",ncol);
printf("Number of table partitions:%d\n",ndiv);
printf("\n") ;
```

(e) Settings for the partitioning key specification order, data type code, and data length code

```
/*******************************************************/
/* (a) Setting values with codes                       */
/*******************************************************/
collst[0].datatype=PDSQL_CHAR ;          /* CHAR(10)*/
collst[0].datalen=10 ;
collst[1].datatype=PDSQL_DECIMAL ;       /* DEC(5,2) */
collst[1].datalen=5*256+2 ;
collst[2].datatype=PDSQL_SMALLINT ;      /* SMALLINT */
collst[2].datalen=2 ;
collst[3].datatype=PDSQL_NVARCHAR ;      /* NVARCHAR(4) */
collst[3].datalen=4 ;

/*******************************************************/
/* (b) Retrieving values from the dictionary table     */
/*******************************************************/

    EXEC SQL
    declare cr1 cursor for
        select  value(DIVCOL_ORDER,1) ,
                DATA_TYPE_CODE,
                DATA_LENGTH_CODE
        from MASTER.SQL_COLUMNS
        where TABLE_SCHEMA=USER
          and TABLE_NAME='TABLE1'
          and DIVIDED_KEY='Y'
        order by 1 asc ;

EXEC SQL open cr1 ;
EXEC SQL whenever not found goto fetch_end1 ;

for  (i = 0 ; ; i++) {
   EXEC SQL fetch cr1 into :xncol , : xdatatype , : xdatalen ;
    collst[i].datatype = xdatatype ;
    collst[i].datalen = xdatalen ;
}

fetch_end1 :
EXEC SQL close cr1 ;
```

(f) Settings for storage RDAREA name

```
/*******************************************************/
/* Retrieving values from the dictionary table         */
/*******************************************************/
EXEC SQL
    declare cr2 cursor for
        select RDAREA_NAME
```

```
                from MASTER.SQL_DIV_TABLE
                where TABLE_SCHEMA=USER
                  and TABLE_NAME='TABLE1'
                order by DIV_NO asc ;

  EXEC SQL open cr2 ;
  EXEC SQL whenever not found goto fetch_end2 ;

  for (j = 0 ; ; j++) {
      EXEC SQL fetch cr2 into :xrdname ;
      strncpy(rdarealst[j].rdareaname,
              xrdname.data,
              xrdname.length) ;
      rdarealst[j].rdareaname[xrdname.length] = '\0' ;
  }

  fetch_end2 :
  EXEC SQL close cr2 ;

  EXEC SQL DISCONNECT ;


  /*****************************************************/
  /* Displaying RDAREA information                     */
  /*****************************************************/
  printf("RDAREA-name[")  ;
  for (j = 0 ; j<ndiv ; j++) {
      printf("%s",rdarealst[j].rdareaname) ;
      if (j != ndiv-1) {
               printf(",") ;
      } else ;
  }
  printf("]\n") ;
  printf("\n") ;
```

(g) Data setting to be stored in partitioning keys

```
/
**************************************************************
*/
/* Assigning data in binary format.                         */
/* Setting data and call hash function for each line.       */
/
**************************************************************
*/
memcpy((char *)data[0].data_char,"abcdefg  ",10) ;/*"abcdefg "
*/

data[1].data_dec[0] = 0x04 ;
```

```
data[1].data_dec[1] = 0x32 ;
data[1].data_dec[2] = 0x1D ;                /* -43.21 */

data[2].data_smallint = 12345 ;             /* 12345 */

/* NCHAR and NVARCHAR specify character codes used in the HiRDB server.*/
data[3].data_varchar.length = 6 ;
data[3].data_varchar.data[0] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[1] = 0xa0 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[2] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[3] = 0xa2 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[4] = 0x82 ; /* Example of shift JIS kanji code */
data[3].data_varchar.data[5] = 0xa4 ; /* Example of shift JIS kanji code */

/
***************************************************************
*/
/*Displaying data type code, data length code, and data area   */
/
***************************************************************
*/
print_data(ncol , collst , data) ;

/
***************************************************************
*/
/* Hash function call                                          */
/
***************************************************************
*/
rc =
p_rdb_dbhash(hashcode,ncol,collst,dadlst,ndiv,ncspace,flags,&r
dno);

switch (rc) {
case p_rdb_RC_RTRN :
  /
**************************************************************/
  /* Normal processing                                         */
  /
**************************************************************/
  printf("Partitioning condition specification order : %d ->
%s\n",

     rdno,rdarealst[rdno-1].rdareaname) ;
  break ;
default :
/
```

```
*************************************************************
*/
/* Adding error processing                                    */
/
*************************************************************
*/
  printf("RETURN CODE=%d\n",rc) ;
  break ;
}

return ;
}

/
*************************************************************
*/
/* Display function for data type code, data length code,     */
/*and data area                                                */
/
*************************************************************
*/
void print_data( short            ncol ,
                 p_rdb_collst_t  *pcollst ,
                 union data_area *pdata )
{
  int i , j ;                               /* Counter variables  */
  int len;
  p_rdb_collst_t   *ccollst ;
  union data_area  *cdata ;

  printf("Partitioning key specification order Data type code
Data length code Binary-format data value\n") ;
  for  (i = 0 , ccollst = pcollst , cdata = pdata ;
        i<ncol ;
        i++ , ccollst++ , cdata++) {
      printf("            %2d         %#.4x         %#.4x  ",
              i+1,ccollst->datatype, ccollst->datalen) ;

    switch (ccollst->datatype) {
      case PDSQL_CHAR :
      case PDSQL_MCHAR :
      case PDSQL_INTEGER :
      case PDSQL_SMALLFLT :
      case PDSQL_FLOAT :
      case PDSQL_SMALLINT :
      case PDSQL_DATE :
      case PDSQL_TIME :
      case PDSQL_TIMESTAMP :
```

```
            len=ccollst->datalen ;
            break ;
        case PDSQL_VARCHAR :
        case PDSQL_MVARCHAR :
        case PDSQL_NVARCHAR :
            len=cdata->data_varchar.length+2 ;
            break ;
        case PDSQL_NCHAR :
            len=ccollst->datalen*2 ;
            break ;
        case PDSQL_DECIMAL :
        case PDSQL_YEARTODAY :
        case PDSQL_HOURTOSEC :
            len=ccollst->datalen/256/2+1 ;
            break ;
        default :
            break ;
        }
        for=(j=0 ; j<len ;j++){
            printf("%.2X",cdata->data_char[j]) ;
        }
        printf("\n") ;
    }
    printf("\n") ;
  return;
  }
```

(h) Execution results for HP-UX and shift JIS kanji codes

```
Hash function code: 6
Number of partitioning key columns: 4
Number of table partitions: 6
RDAREA names: [RU01, RU02, RU03, RU04, RU05, RU06]
```

Partitioning key specification order	Data type code	Data length code	Binary format data value
1	0x00c4	0x000a	61626364656667202020
2	0x00e4	0x0502	04321D
3	0x00f4	0x0002	3039
4	0x00b0	0x0004	000682A082A282A4

```
Partitioning condition specification order: 1 > RU01
```

(6) Retrieval from dictionary tables (for hash partitioning)

Examples of retrieval from dictionary tables for hash partitioning are shown below.

(a) Obtaining the hash function name, number of partitioning key columns, and number of table partitions for a hash-partitioned table

```
SELECT HASH_NAME,                /* Hash function name */
     VALUE(N_DIV_COLUMN,1),      /* Number of partitioning key columns */
     N_RDAREA                    /* Number of table partitioned */
  FROM MASTER.SQL_TABLES
    WHERE TABLE_SCHEMA = authorization-identifier
                       /* Item with matching authorization identifier */
      AND TABLE_NAME = table-identifier
                       /* Item with matching table identifier */
```

(b) Obtaining the partitioning key specification order, the data type code, and the data length code

```
SELECT VALUE(DIVCOL_ORDER,1), /* Partitioning key specification order */
     DATA_TYPE_CODE,          /* Data type code */
     DATA_LENGTH_CODE         /* Data length code */
  FROM MASTER.SQL_COLUMNS
    WHERE TABLE_SCHEMA = authorization-identifier
                       /* Item with matching authorization identifier */
      AND TABLE_NAME= table-identifier
                       /* Item with matching table identifier */
      AND DIVIDED_KEY='Y'     /* Item that is a partitioning key*/
  ORDER BY 1 ASC
```

(c) Obtaining the storage RDAREA name

```
SELECT  DIV_NO,            /* Partitioning condition specification order */
        RDAREA_NAME        /* Storage RDAREA name */
  FROM MASTER.SQL_DIV_TABLE
    WHERE TABLE_SCHEMA = authorization-identifier
                       /* Item with matching authorization identifier */
      AND TABLE_NAME = table-identifier
                       /* Item with matching table identifier */
  ORDER BY 1 ASC
```

(7) Retrieval from dictionary tables (for matrix partitioning)

Examples of retrieval from dictionary tables for matrix partitioning are shown below.

(a) Obtaining the hash function name, number of partitioning key columns, and number of table partitions for a hash-partitioned table

- Obtaining the hash function name and the number of partitioning key columns

```
select HASH_NAME,                    /* Hash function name */
     value(N_DIV_COLUMN,1),          /* Number of partitioning key columns */
```

```
      KEY_NO                         /* Partitioning key number */
 from MASTER.SQL_DIV_TYPE
     where TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
      and TABLE_NAME=table-identifier        /* Item with matching table identifier */
```

- Obtaining the number of partitions in the key

```
select distinct N_DIVISION           /* Number of partitions in key */
from MASTER.SQL_PARTKEY
     where TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
      and TABLE_NAME=table-identifier        /* Item with matching table identifier */
      and KEY_NO=partitioning-key-number          /* Set partitioning key numbers */
                                     /* for hash partitioning */
```

(b) Obtaining the partitioning key specification order, the data type code, and the data length code

```
select DIVCOL_ORDER,                          /* Number of partitions in key */
       DATA_TYPE_CODE,                         /* Data type code */
       DATA_LENGTH_CODE                        /* Data length code */
 from MASTER.SQL_COLUMNS X,
     MASTER.SQL_PARTKEY Y
     where X.TABLE_SCHEMA=Y.TABLE_SCHEMA
       and X.TABLE_NAME=Y.TABLE_NAME
       and X.COLUMN_ID=Y.COLUMN_ID
      and Y.TABLE_SCHEMA=authorization-identifier         /* Item with matching
authorization identifier */
       and Y.TABLE_NAME=table-identifier            /* Item with matching table
identifier */
       and Y.KEY_NO=partitioning-key-number              /* Set partitioning key
number */
                                               /* for hash partitioning */
  order by DIVCOL_ORDER asc
```

(c) Obtaining the storage RDAREA name

```
select DIV_NO,                  /* Partitioning condition specification order */
      RDAREA_NAME                        /* Storage RDAREA name */
 from MASTER.SQL_DIV_TABLE
     where TABLE_SCHEMA=authorization-identifier      /* Item with matching
authorization identifier */
      and TABLE_NAME=table-identifier        /* Item with matching table identifier */
       order by 1 asc
```

Note

The partitioning condition specification order is determined from the partitioning key sequence numbers. The expression follows:

$N \times m - (N - n)$

N: Number of partitions in second dimension

m: Partitioning key sequence number of first partitioning key

n: Partitioning key sequence number of second partitioning key

G.2 Space conversion function

The space conversion function converts single-byte spaces in a character string to double-byte spaces, and vice versa. Because this function lets you know the conversion result without having to store character string data in a database, you can use the function for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when determining the partitioning key for partitioning a table by the key range
- To create an input data file for each RDAREA when loading data to a key-range-partitioned table in units of RDAREAs concurrently using the database load utility

Prerequisites for using the space conversion function

The prerequisites for using the space conversion function are the same as for the hash function for table partitioning. For details, see *G.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the space conversion function

The prerequisites for creating and executing a UAP using the space conversion function are the same as for the hash function for table partitioning. For details, see *G.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the space conversion function

(a) Specification configuration

For details about the specification configuration, see *G.1(3)(b) Specification configuration*.

(b) Space conversion function (p_rdb_conv_space)

Function

The function converts spaces according to the specified conversion type as follows:

Single-byte space → double-byte space:

Converts two consecutive single-byte spaces in a character string to one double-byte space.

Double-byte space → single-byte space:

Converts each double-byte space in a character string to two single-byte spaces.

The function converts spaces in the character string indicated by `srcp` and stores the conversion result in `destp`. The following table shows the combination of `stype` and `flags` arguments and the conversion type:

stype (data type)	flags (conversion type)	
	Single-byte space → double-byte space	Double-byte space → single-byte space
`NCHAR`	Checks two bytes at a time from the top and converts any two consecutive single-byte spaces to a double-byte space.* The function does not convert any isolated single-byte space.	Checks two bytes at a time from the top and converts any double-byte space* to two single-byte spaces.
`NVARCHAR`		
`MCHAR`	Results in an error.	Checks each character code from the top and converts any double-byte space* to two single-byte spaces.
`MVARCHAR`		

* The function treats the value specified in the `ncspace` argument as the character code for double-byte space.

Header files

`#include<pdauxcnv.h>`

This header file is required to use the space conversion function.

`#include<pdbsqlda.h>`

This header file lets you use macros (with a name beginning with `PDSQL_`) to specify a data type code. If the data type code is to be retrieved from a data dictionary table, this header file is not necessary.

Format

```
int p_rdb_conv_space(char           *srcp,
                     unsigned char  stype,
                     unsigned int   srcl,
                     char           *destp,
                     unsigned char  ncspace[2],
                     int            flags);
```

Arguments

`srcp (input)`

Specifies the start address of the character string storage area.

`stype (input)`

Specifies the data type before conversion. Specifiable data types are as follows:

Macro name	Data type
PDSQL_NCHAR	NCHAR
PDSQL_NVARCHAR	NVARCHAR
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

`srcl (input)`

Specifies the length of the character string specified in srcp. For a variable-length character string, specify the length of the actual character string (in bytes) that is stored in the area indicated by srcp.

`destp (output)`

Sets the start address of the character string storage area after conversion. Allocate this area indicated by destp on the side that calls the space conversion function.

`ncspace (input)`

Specifies a two-byte area that contains the double-byte space character for the national character code used in the HiRDB server. For the double-byte space characters that can be specified in ncspace, see *Table G-4 Double-byte space characters specified in ncspace*.

`flags (input)`

Specifies the conversion type. Available conversion types are as follows:

Macro name	Conversion type
p_rdb_HALF_TO_FULL_SPACE	Single-byte space → double-byte space
p_rdb_FULL_TO_HALF_SPACE	Double-byte space → single-byte space

Return values

data type: int

p_rdb_RC_RTRN(0)

Normal termination.

p_rdb_RC_ERRINVF(-8)

Invalid flags argument

p_rdb_RC_ERRTYPC(-9)

Invalid data type

Notes

1. The National Language Support (NLS) facility provided by the OS is used to convert double-byte spaces into single-byte spaces. Therefore, before invoking the space conversion function, you must use the setlocale function to set an appropriate locale to the LC_CTYPE or LC_ALL category. Additionally, for a Windows UAP, a Linux UAP with a character code type of SJIS, or a UAP with a character code type of CHINESE, you must invoke the p_rdb_set_lang function before invoking the space conversion function. For details about the p_rdb_set_lang function, see *G.4 Character code type specification function*.

   If the character code type of the character string indicated by the srcp argument contradicts the locale specified by the setlocale or p_rdb_set_lang function, the operation cannot be guaranteed.
2. Operation is guaranteed if the data input area is the same as the data output area, or if the output area is located before the input area and the latter half of the output area overlaps the first half of the input area.
3. Be sure to specify an appropriate value in srcl because the function does not check the length of a character string for any error.
4. The function uses 0x20 as the character code for a single-byte space and the character code specified in the ncspace argument as the character code for a double-byte space.
5. The data types that can be specified for input are NCHAR, NVARCHAR, MCHAR,

and MVARCHAR.

6. For a variable-length character string, the function references srcl to determine the length of character string to be converted. Specify the length without the real length section in the srcl argument.
7. The real length section of a variable-length character string remains unchanged after space conversion.
8. If the character code type is Unicode, the operation of this function cannot be guaranteed. If the character code type is Unicode, use the p_rdb_conv_space_utf8 function. For details on this function, see *(c) Space conversion function (p_rdb_conv_space_utf8)*.

(c) Space conversion function (p_rdb_conv_space_utf8)

Function

This function converts double-byte spaces into single-byte spaces when the character code is Unicode (UTF-8). It converts each double-byte space inside a character string into two single-byte spaces.

This function applies space conversion to the space characters inside the character string indicated by srcp. The conversion result is stored in destp and the character string length following the conversion is stored in destl.

The following table shows the combinations of the stype and flags arguments, along with conversion details.

stype (data type)	flags (conversion type)	
	Single-byte spaces -> Double-byte spaces	Double-byte spaces -> Single-byte spaces
MCHAR	An error occurs.	Character codes are checked from the beginning, and any double-byte spaces* found are converted into single-byte spaces.
MVARCHAR		

* 0xE38080 is treated as a double-byte space character code.

Header files

```
#include <pdauxcnv.h>
```

This header file is required to use the space conversion function.

```
#include <pdbsqlda.h>
```

This header file lets you use macros (with a name beginning with PDSQL_) to specify a data type code. If the data type code is to be retrieved from a data dictionary table, this header file is not necessary.

Format

```
int p_rdb_conv_space_utf8(char          *srcp,
                 unsigned char          stype,
                 unsigned int           srcl,
                 char                   *destp,
                 unsigned int           *destl,
                 int                    flags) ;
```

Arguments

srcp (input)

Specifies the start address of the character string storage area.

stype (input)

Specifies the data type before conversion. The following table shows the data types that can be specified:

Macro name	Data type
PDSQL_MCHAR	MCHAR
PDSQL_MVARCHAR	MVARCHAR

srcl (input)

Specifies the length of the character string specified by srcp. For a variable-length character string, this argument specifies the length (units: bytes) of the character string actually stored in the area indicated by srcp.

destp (output)

Specifies the start address of the character string storage area after conversion. The area indicated by destp must be allocated in the system that invokes the space conversion function.

destl (output)

Specifies the length of the character string specified by destp. For a variable-length character string, this argument specifies the length (units: bytes) of the character string actually stored in the area indicated by destp.

flags (input)

Specifies a conversion type. The following table shows the conversion types:

Macro name	Conversion type
p_rdb_FULL_TO_HALF_SPACE	Double-byte spaces -> Single-byte spaces

Return values

Data type: `int`

`p_rdb_RC_RTRN(0)`

Normal termination

`p_rdb_RC_ERRINVF(-8)`

Invalid `flags` argument

`p_rdb_RC_ERRTYPC(-9)`

Invalid data type

Notes

1. This space conversion function is used only for Unicode (`UTF-8`).
2. Before invoking this function, you must set `UTF8` for the `lang` argument and invoke the `p_rdb_set_lang` function. For details about the `p_rdb_set_lang` function, see *G.4 Character code type specification function*.
3. If the data input area is the same as the data output area, or if the output area is located before the input area and the second half of the output area overlaps with the first half of the input area, the correct operation of the function is guaranteed.
4. Because errors related to character string length are not checked, you must enter an appropriate value in `srcl`.
5. The single- and double-byte space codes use `0x20` and `0xE38080`, respectively.
6. The data types that can be set for the input are `MCHAR` and `MVARCHAR`.
7. When a character string has a variable length, `srcl` is referenced for the length of the character string to be converted. Specify for `srcl` a length that excludes the effective-length portion.
8. Because space conversion converts each double-byte space (3 bytes) into two single-byte spaces (2 bytes), the length of the character string following conversion is shorter than that before the conversion.
9. When a character string has a variable length, the effective-length portion of the area for storing character strings following conversion stores the data length following the conversion.
10. The data inside the area for storing character strings following conversion is guaranteed only for the length specified in `destl`.
11. If this function is invoked by a character code other than Unicode (`UTF-8`), the operation of this function cannot be guaranteed.

G.3 Function for conversion to a DECIMAL signed normalized number

The function for conversion to a DECIMAL signed normalized number sets the sign for `DECIMAL` data to either `X'C'` or `X'D'` (for a value of `0`, the sign is `X'C'`). Because this function lets you obtain the normalized sign without having to store `DECIMAL` data in a database, you can use it for the following purposes:

- To evaluate whether the data to be stored will be partitioned equally when `pd_dec_sign_normalize=Y` is specified in the system definition and the key for partitioning a table is determined by the key range
- To create an input data file for each RDAREA when `pd_dec_sign_normalize=Y` is specified in the system definition and data is loaded to a key-range-partitioned table in units of RDAREAs concurrently using the database load utility

Prerequisites for using the function for conversion to a DECIMAL signed normalized number

The prerequisites are the same as those for the hash function for table partitioning. For details, see *G.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the function for conversion to a DECIMAL signed normalized number

The prerequisites are the same as those for the hash function for table partitioning. For details, see *G.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the function for conversion to a DECIMAL signed normalized number

(a) Specification configuration

For details about the specification configuration, see *G.1(3)(b) Specification configuration*.

(b) Function for conversion to a DECIMAL signed normalized number (p_rdb_dec_sign_norm)

Function

The function normalizes the sign of `DECIMAL` data indicated by `srcp` as follows:

Before normalization	After normalization
X'A'	X'C'
X'B'	X'D'*

Before normalization	After normalization
X'C'	X'C'
X'D'	X'D'*
X'E'	X'C'
X'F'	X'C'
X'0' to X'9'	Error

* If the absolute value of data is 0, the sign part is set to X'C'.

Header file

#include<pdauxcnv.h>

This header file is required to use the function for conversion to a DECIMAL signed normalized number.

Format

```
int p_rdb_dec_sign_norm(unsigned char   *srcp,
                        short           srcl,
                        unsigned char   *destp);
```

Arguments

srcp (input)

Specifies the start address of the DECIMAL data to be normalized.

srcl (input)

Specifies the length code of the DECIMAL data indicated by the srcp argument. Specifiable data length codes are as follows:

Data type	Data length code
INTERVAL YEAR TO DAY	8 × 256
INTERVAL HOUR TO SECOND	6 × 256
DECIMAL[(*p*[,*q*])]	*p* × 256 + *q* (If *p* is omitted, 15 is assumed; if *q* is omitted, 0 is assumed.)

destp (output)

Sets the normalized DECIMAL data. Allocate this area indicated by destp on the side that calls the function for conversion to a DECIMAL signed normalized number.

Return values

data type: `int`

`p_rdb_RC_RTRN(0)`

Normal termination.

`p_rdb_RC_ERRDFRM(-12)`

Invalid sign part for data.

Notes

1. The function does not check anything for error other than the sign part of `DECIMAL` data. Operation is not guaranteed if `DECIMAL` data is invalid or the data length code specified by the `srcl` argument contradicts the `DECIMAL` data.
2. Operation is guaranteed if the data input area is the same as the data output area, or if the output area is located before the input area and the latter half of the output area overlaps the first half of the input area.

G.4 Character code type specification function

The character code type specification function is used to pass the type of the character code from a UAP to a hash function for table partitioning or a space conversion function.

By using this function to specify the type of the character code, you can execute processing that depends on the type of character code, such as the hash function for table partitioning and the space conversion function.

Prerequisites for using the character code type specification function

The prerequisites are the same as those for the hash function for table partitioning. For details, see *G.1(1) Prerequisites for using the hash function for table partitioning*.

Prerequisites for creating and executing a UAP using the character code type specification function

The prerequisites are the same as those for the hash function for table partitioning. For details, see *G.1(2) Creating and executing UAPs that use the hash function for table partitioning*.

(1) Details about the character code type specification function

(a) Specification configuration

For details about the specification configuration, see *G.1(3)(b) Specification configuration*.

(b) Character code type specification function (p_rdb_set_lang)

Function

The character code type specification function specifies the type of character code to be handled by the hash function for table partitioning and the space conversion function.

Header file

`#include<pdauxcnv.h>`

This header file is required to use the character code type specification function.

Format

```
int p_rdb_set_lang(char *lang);
```

Arguments

`lang (input)`

Specifies the type of character encoding to be handled by the hash function for table partitioning and the space conversion function. Specifiable character encodings are as follows:

Type of character codes	Value of lang argument
Shift JIS kanji codes[1]	"SJIS"
EUC Chinese kanji codes	"CHINESE"
Single-byte character codes[2]	"C"
Unicodes (UTF-8)	"UTF8"

[1] Can be specified for Linux and Windows.

[2] Can be specified for Windows.

If a blank character string (for example, `p_rdb_set_lang ("")`) is specified, the operation is as follows:

UNIX environment

The `setlocale` function executed before this function sets the character code type corresponding to the locale that was set to the `LC_ALL` category. If the `setlocale` function has not been executed, the character code type corresponding to the default locale of the `LC_ALL` category is set.

Windows environment

The default character code type of the OS is set. However, if the default character code type is set to a type that is not listed in the above table, the operation cannot be guaranteed.

Return values

data type: `int`

`p_rdb_RC_RTRN(0)`

Normal termination.

`p_rdb_RC_ERRIVLG(-10)`

Invalid character encoding type.

Notes

1. You must execute `p_rdb_set_lang` if any of the following conditions is applicable:

   - When setting a character code type from a UAP in a Windows environment
   - When invoking `p_rdb_conv_space_utf8` from a UAP in a UNIX environment*
   - When setting the character code type to `SJIS` from a UAP in a Linux environment
   - When setting the character code type to `CHINESE` from a UAP in a UNIX environment

   * Before invoking `p_rdb_conv_space_utf8`, execute `p_rdb_set_lang`. When invoking the space conversion function `p_rdb_conv_space`, use the `setlocale` function provided by the OS instead of `p_rdb_set_lang`.

2. In an UNIX environment, after setting a character code type using this function, to use a character code type that cannot be used to use another function, issue `p_rdb_set_lang("")` first and then invoke the `setlocale` function to reset the character code type to an appropriate one.

H. Maximum and Minimum HiRDB Values

The HiRDB system defines a specific range of acceptable values for each item. This appendix lists the maximum and minimum values allowed.

Table H-1 lists the maximum and minimum values for the HiRDB system.

Table H-1: HiRDB maximum and minimum values

Classification	Item	Minimum value	Maximum value	Unit
Database manipulation	Number of retrieval items	1	30,000	Tables
	Number of update columns	1	30,000	
	Number of sort columns	1	255	
	Number of grouped columns	0 or 1[1]	255	
	Number of duplicate locked columns	1	255	
	Number of nested logical operations	0	255	
	Number of value expressions of IN predicate	1	255	
	Number of nested scalar functions	0	255	
	Length of character string literal in SQL	0	255	Bytes
	Length of national character string literal in SQL	0	127	Characters
	Length of mixed character string literal in SQL	0	255	Bytes
	Length of one SQL statement	1	2,000	Kilobytes
	Number of tables that can be specified in one SQL statement	1	64	Tables
	Number of correlation names that can be specified in one SQL statement	0	65	
	Number of locked base tables in `LOCK` statement	1	64	
	Number of arguments in `CALL` statement	0	30,000	
	Row length of work table[2]	6	32,720	Bytes

Classification	Item	Minimum value	Maximum value	Unit
UAP	Number of SQL statements in one UAP	1	4,095	Tables
	Number of cursors in one UAP	0	1,023	
	Number of `?` parameters in SQL statement	0	30,000	
	Number of embedded variables in SQL statement	0	30,000	

[1] If the `GROUP BY` clause is specified, the minimum value is `1`. If the `HAVING` clause is specified without the `GROUP BY` clause, or if a set function is specified in the `SELECT` clause, the minimum value is `0`.

[2] Some SQL statements require a work table file. For details, see the *HiRDB Version 8 Installation and Design Guide*.

Index

C

D

E

F

H

Q

R

T

U

V

W

X

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)