HITACHI
Inspire the Next

Hitachi Advanced Database Application Development Guide

3000-6-502-H0(E)

Notices

■ Relevant program products

P-9W62-C411 Hitachi Advanced Data Binder version 05-01 (for Red Hat(R) Enterprise Linux(R) Server 6 (64-bit x86_64) and Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64))

P-9W62-C311 Hitachi Advanced Data Binder Client version 05-01 (for Red Hat(R) Enterprise Linux(R) Server 6 (64-bit x86_64) and Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64))

P-2462-C114 Hitachi Advanced Data Binder Client version 05-01 (for Windows 7, Windows 8.1, Windows 10, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, and Windows Server 2016)

This manual can be used for products other than the products shown above. For details, see the *Release Notes*.

Hitachi Advanced Data Binder is the product name of Hitachi Advanced Database in Japan.

■ Trademarks

HITACHI, HA Monitor, HiRDB, Job Management Partner 1 and JP1 are either trademarks or registered trademarks of Hitachi, Ltd. in Japan and other countries.

Access is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

AMD is a trademark of Advanced Micro Devices, Inc.

Excel is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Intel is a trademark of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

MSDN is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries.

Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.

UNIX is a trademark of The Open Group.

Visual Studio is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Windows is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Windows Server is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Other company and product names mentioned in this document may be the trademarks of their respective owners.

1. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)

2. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

3. This product includes software written by Tim Hudson (tjh@cryptsoft.com).

4. This product uses OpenSSL Toolkit software in accordance with the OpenSSL License and Original SSLeay License, which are described as follows.

LICENSE ISSUES

==========

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
the OpenSSL License and the original SSLeay license apply to the toolkit.
See below for the actual license texts. Actually both licenses are BSD-style
Open Source licenses. In case of any license issues related to OpenSSL
please contact openssl-core@openssl.org.

OpenSSL License

/* ==

* Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

*

* Redistribution and use in source and binary forms, with or without

* modification, are permitted provided that the following conditions

* are met:

*

* 1. Redistributions of source code must retain the above copyright

* notice, this list of conditions and the following disclaimer.

*

* 2. Redistributions in binary form must reproduce the above copyright

* notice, this list of conditions and the following disclaimer in

* the documentation and/or other materials provided with the

* distribution.

*

* 3. All advertising materials mentioning features or use of this

* software must display the following acknowledgment:

* "This product includes software developed by the OpenSSL Project

* for use in the OpenSSL Toolkit. (http://www.openssl.org/)"

*

* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to

* endorse or promote products derived from this software without

* prior written permission. For written permission, please contact

* openssl-core@openssl.org.

*

* 5. Products derived from this software may not be called "OpenSSL"

* nor may "OpenSSL" appear in their names without prior written

* permission of the OpenSSL Project.

*

* 6. Redistributions of any form whatsoever must retain the following

* acknowledgment:

* "This product includes software developed by the OpenSSL Project

* for use in the OpenSSL Toolkit (http://www.openssl.org/)"

*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* ==
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/

Original SSLeay License

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.

■ Microsoft product screen shots

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names:

Abbreviation			Full name or meaning
Windows	Windows 7	Windows 7 x86	Microsoft(R) Windows(R) 7 Professional (32-bit)
			Microsoft(R) Windows(R) 7 Enterprise (32-bit)
			Microsoft(R) Windows(R) 7 Ultimate (32-bit)
		Windows 7 x64	Microsoft(R) Windows(R) 7 Professional (64-bit)
			Microsoft(R) Windows(R) 7 Enterprise (64-bit)
			Microsoft(R) Windows(R) 7 Ultimate (64-bit)
	Windows 8.1	Windows 8.1 x86	Windows(R) 8.1 Pro (32-bit)
			Windows(R) 8.1 Enterprise (32-bit)

Abbreviation			Full name or meaning
		Windows 8.1 x64	Windows(R) 8.1 Pro (64-bit)
			Windows(R) 8.1 Enterprise (64-bit)
	Windows 10	Windows 10 x86	Windows(R) 10 Pro (32-bit)
			Windows(R) 10 Enterprise (32-bit)
		Windows 10 x64	Windows(R) 10 Pro (64-bit)
			Windows(R) 10 Enterprise (64-bit)
	Windows Server 2008 R2		Microsoft(R) Windows Server(R) 2008 R2 Standard
			Microsoft(R) Windows Server(R) 2008 R2 Enterprise
			Microsoft(R) Windows Server(R) 2008 R2 Datacenter
	Windows Server 2012		Microsoft(R) Windows Server(R) 2012 Standard
			Microsoft(R) Windows Server(R) 2012 Datacenter
	Windows Server 2012 R2		Microsoft(R) Windows Server(R) 2012 R2 Standard
			Microsoft(R) Windows Server(R) 2012 R2 Datacenter
	Windows Server 2016		Microsoft(R) Windows Server(R) 2016 Standard
			Microsoft(R) Windows Server(R) 2016 Datacenter

■ 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.

■ Issued

Apr. 2020

■ Copyright

Preface

This manual describes the basic techniques for using Hitachi Advanced Database to develop applications. It also explains how to set up an HADB client environment.

Note that, in this manual, and in the information output by the product (messages, command output results, and so on), *HADB* is often used in place of *Hitachi Advanced Database*.

■ Intended readers

This manual is intended for:

- Application program developers
- HADB client administrators

Readers of this manual must have:

- A basic knowledge of SQL
- A basic knowledge of Java programming and a basic knowledge of JDBC (if you plan to create application programs in Java)
- A basic knowledge of ODBC (if you plan to create ODBC-compliant application programs)
- A basic knowledge of programming in C or C++ (if you plan to create application programs in C or C++)
- A basic knowledge of Linux or Windows system management

■ Organization of this manual

This manual is organized into the following parts, chapters, and appendixes:

PART 1: Environment Setup (Common)

1. Overview of Application Program Development and Execution

   This chapter provides an overview of application program development, explains the prerequisites that you need to know before you begin developing an application program, and shows the application program execution modes.

2. Designing Client Definitions

   This chapter explains the format in which operands for client definitions are to be specified, the content of client definitions, and the syntax rules that apply to client definitions.

3. Setting Up an Environment for the JDBC Driver

   This chapter explains how to set up an environment for the JDBC driver, including how to install the JDBC driver and specify the environment variables.

4. Setting Up an Environment for an HADB Client (If the ODBC Driver and CLI Functions Are Used)

   This chapter explains how to set up an environment for an HADB client, including installation of an HADB client and specification of environment variables.

PART 2: Application Program Creation (Common)

5. Designs Related to Improvement of Application Program Performance

This chapter explains designs related to improving the performance of application programs.

6. Tuning Application Programs

This chapter explains how to use access paths.

PART 3: Application Program Creation (JDBC)

7. Creating Application Programs

This chapter explains how to create application programs that use the JDBC driver.

8. The JDBC 1.2 API

This chapter describes the interfaces and methods available in the JDBC 1.2 API.

9. The JDBC 2.1 Core API

This chapter explains HADB's scope of support for the functions added in the JDBC 2.1 Core API.

10. The JDBC 2.0 Optional Package

This chapter describes the interfaces and methods available in the JDBC 2.0 Optional Package.

11. The JDBC 3.0 API

This chapter describes the interfaces and methods available in the JDBC 3.0 API.

12. The JDBC 4.0 API

This chapter describes the interfaces and methods available in the JDBC 4.0 API.

13. JDBC 4.1 API

This chapter explains HADB's scope of support for the functions added in the JDBC 4.1 API.

14. JDBC 4.2 API

This chapter explains HADB's scope of support for the functions added in the JDBC 4.2 API.

PART 4: Application Program Creation (ODBC)

15. Creating Application Programs

This chapter explains how to set up an environment for the HADB ODBC driver and provides notes about creating application programs that support ODBC.

16. ODBC Functions

This chapter describes the capabilities and syntax of the ODBC functions provided by HADB.

17. Troubleshooting

This chapter explains how to troubleshoot use of the ODBC interfaces.

PART 5: Application Program Creation (CLI Functions)

18. Creating Application Programs

This chapter explains the basic considerations involved in designing and creating application programs in C and C++.

19. CLI Functions

This chapter describes the capabilities and syntax of the CLI functions provided by HADB.

A. Sample Application Program

This appendix provides an overview of the sample application program that is provided, explains the preparations for using the sample program, and explains the sample program execution procedure.

B. Structure of HADB Client Directories

This appendix describes the structures of the client directories of HADB clients (during installation and operation).

C. Estimating the Memory Requirements for an HADB Client

This appendix explains how to estimate the memory requirements for an HADB client.

■ Related publications

This manual is part of a related set of manuals. The manuals in the set are listed below (with the manual numbers):

- *Hitachi Advanced Database Setup and Operation Guide* (3000-6-501(E))
- *Hitachi Advanced Database Command Reference* (3000-6-503(E))
- *Hitachi Advanced Database SQL Reference* (3000-6-504(E))
- *Hitachi Advanced Database Messages* (3000-6-505(E))
- *HA Monitor Cluster Software Guide (for Linux(R) (x86) Systems)* (3000-9-201(E))
- *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 System Design (Work Tasks) Guide* (3021-3-320(E))
- *JP1 Version 11 JP1/Base User's Guide* (3021-3-A01(E))

In references to Hitachi Advanced Database manuals, this manual uses *HADB* in place of *Hitachi Advanced Database*.

Example: *HADB Setup and Operation Guide*

In references to the HA Monitor manual, this manual uses *HA Monitor for Linux(R) (x86)* in place of *HA Monitor Cluster Software Guide (for Linux(R) (x86) Systems)*.

Example: *HA Monitor for Linux(R) (x86)*

In references to the Job Management Partner 1/Automatic Job Management System 3 manual, this manual uses *Job Management Partner 1/Automatic Job Management System 3 System Design (Work Tasks) Guide* in place of *Job Management Partner 1 Version 10 Job Management Partner 1/Automatic Job Management System 3 System Design (Work Tasks) Guide*.

Example: *Job Management Partner 1/Automatic Job Management System 3 System Design (Work Tasks) Guide*

In references to the JP1/Base manual, this manual uses *JP1/Base User's Guide* in place of *JP1 Version 11 JP1/Base User's Guide*.

Example: *JP1/Base User's Guide*

■ Conventions: Abbreviations for product names

This manual uses the following abbreviations for product names:

Abbreviation		Full name or meaning
HADB	HADB server	Hitachi Advanced Database
	HADB client	Hitachi Advanced Database Client
Linux	Linux	Linux(R)

Abbreviation		Full name or meaning
	Red Hat Enterprise Linux Server 6	Red Hat(R) Enterprise Linux(R) Server 6 (64-bit x86_64)
	Red Hat Enterprise Linux Server 6 (64-bit x86_64)	
	Red Hat Enterprise Linux Server 7	Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64)
	Red Hat Enterprise Linux Server 7 (64-bit x86_64)	
HDLM		Hitachi Dynamic Link Manager Software
JP1/AJS3		Job Management Partner 1/Automatic Job Management System 3
JP1/Audit		JP1/Audit Management - Manager
Red Hat Enterprise Linux Server 6 (64-bit x86_64)		Red Hat(R) Enterprise Linux(R) Server 6 (64-bit x86_64)
Red Hat Enterprise Linux Server 7 (64-bit x86_64)		Red Hat(R) Enterprise Linux(R) Server 7 (64-bit x86_64)

■ Conventions: Acronyms

This manual also uses the following acronyms:

Acronym	Full name or meaning
APD	Application Parameter Descriptor
API	Application Programming Interface
ARD	Application Row Descriptor
BI	Business Intelligence
BLOB	Binary Large Object
BNF	Backus-Naur Form
BOM	Byte Order Mark
CLI	Call Level Interface
CLOB	Character Large Object
CPU	Central Processing Unit
CSV	Character-Separated Values
DB	Database
DBMS	Database Management System
DMMP	Device Mapper Multipath
DNS	Domain Name System
ER	Entity Relationship
HBA	Host Bus Adapter
ID	Identification number

Acronym	Full name or meaning
IEF	Integrity Enhancement Facility
IP	Internet Protocol
IPD	Implementation Parameter Descriptor
IRD	Implementation Row Descriptor
JAR	Java Archive File
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JTA	Java Transaction API
LOB	Large Object
LRU	Least Recently Used
LV	Logical Volume
LVM	Logical Volume Manager
MSDN	Microsoft Developer Network
NFS	Network File System
NIC	Network Interface Card
NTP	Network Time Protocol
ODBC	Open Database Connectivity
OS	Operating System
PP	Program Product
RAID	Redundant Array of Independent Disks
RDBMS	Relational Database Management System
TLB	Translation Lookaside Buffer
URL	Uniform Resource Locator
VG	Volume Group
WWN	World Wide Name

■ Conventions: Fonts and symbols

The following table explains the fonts used in this manual:

Font	Convention
Bold	**Bold** type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**.

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

The table below shows the symbols used in this manual for explaining commands and operands, such as the operands used in server definitions.

Note that these symbols are used for explanatory purposes only; do not specify them in the actual operand or command.

Symbol	Meaning	Example
`|`	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR.	`adb_sql_text_out = {Y|N}` In this example, the vertical bar means that you can specify either `Y` or `N`.
`[ ]`	In syntax explanations, square brackets indicate that the enclosed item or items are optional.	`adbsql [-V]` In this example, the square brackets mean that you can specify `adbsql`, or you can specify `adbsql -V`.
`{ }`	In syntax explanations, curly brackets indicate that only one of the enclosed items is to be selected.	`adbcancel {--ALL|-u` *connection-ID*`}` In this example, the curly brackets mean that you can specify either `--ALL` or `-u` *connection-ID*.
`...`	In syntax explanations, an ellipsis (`...`) indicates that the immediately preceding item can be repeated as many times as necessary.	`adbbuff -n` *DB-area-name*`[, DB-area-name] ...` In this example, the ellipsis means that you can specify *DB-area-name* as many times as necessary.
`{{ }}`	In syntax explanations, double curly brackets indicate that the enclosed items can be repeated as a single unit.	`{{adbinitdbarea -n` *data-DB-area-name*`}}` In this example, the double curly brackets mean that you can specify `adbinitdbarea -n` *data-DB-area-name* as many times as necessary.
`X` (underline)	In syntax explanations, underlined characters indicate a default value.	`adb_import_errmsg_lv = {0|1}` In this example, the underline means that the value `0` is assumed by HADB when the operand is omitted.
~	A swung dash indicates that the text following it explains the properties of the specified value.	`adb_sys_max_users =` *maximum-number-of-concurrent-connections* ~ <integer> ((`1` to `1024`)) `<<10>>`
< >	Single angle brackets explain the data type of the specified value.	

Symbol	Meaning	Example
(( ))	Double parentheses indicate the scope of the specified value.	In this example, the text following the swung dash means that you can specify an integer in the range from 1 to 1024. If the operand is not specified, the value 10 is assumed by HADB.
<< >>	Double angle brackets indicate a default value.	

■ Conventions: Method abbreviations

- This manual uses "get*xxx* method" to represent any method whose name begins with get.
- This manual uses the "set*xxx* method" to represent any method whose name begins with set.
- This manual uses "execute*xxx* method" to represent any method whose name begins with execute.

■ Conventions: Path names

- $INSTDIR is used to indicate the server directory path (for installation).
- $ADBDIR is used to indicate the server directory path (for operation).
- $DBDIR is used to indicate the DB directory path.
- %ADBCLTDIR% (for a Windows HADB client) or $ADBCLTDIR (for a Linux HADB client) is used to indicate the client directory path.
- %ADBODBTRCPATH% is used to indicate the folder path where HADB's ODBC driver trace files are stored.

■ Conventions: Symbols used in mathematical formulas

The following table explains special symbols used by this manual in mathematical formulas:

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(3 × 6, 4 + 7) is 18.
MIN	Select the smallest value as the result. Example: The result of MIN(3 × 6, 4 + 7) is 11.

■ Conventions: Syntax elements

Syntax element notation	Meaning
<path name>	The following characters can be used in path names: • In Linux Alphanumeric characters, hash mark (#), hyphen (-), forward slash (/), at mark (@), and underscore (_) • In Windows

Syntax element notation	Meaning
	Alphanumeric characters, hash mark (#), hyphen (-), forward slash (/), at mark (@), underscore (_), backslash (\), and colon (:) Note, however, that the characters that can be used might differ depending on the operating system.
<OS path name>	For an OS path name, all characters that can be used in a path name in the operating system can be used. For details about available characters, see the documentation for the operating system you are using.
<character string>	Any character string can be specified.
<integer suffixed by the unit>	Specify the value in a format consisting of a numeric character (in the range from 0 to 9) followed by a unit (MB (megabyte), GB (gigabyte), or TB (terabyte)). Do not enter a space between the numeric character and the unit. • Examples of correct specification 1024MB 512GB 32TB • Example of specification that causes an error 512 GB

■ Abbreviation of function names

- Functions whose names begin with SQL are referred to generically as SQL*xxx* functions.
- Functions whose names begin with SQL and end with W are referred to generically as SQL*xxx*W functions.

■ Conventions: KB, MB, GB, TB, PB, and EB

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.
- 1 PB (petabyte) is $1{,}024^5$ bytes.
- 1 EB (exabyte) is $1{,}024^6$ 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*.

Contents

Notices 2
Preface 8

Part 1: Environment Setup (Common)

1 Overview of Application Program Development and Execution 39
1.1 Procedure and prerequisites for application program development 40
1.1.1 Programming languages for application programs 40
1.1.2 Character encoding 40
1.1.3 Application program development environment 41
1.2 Application program execution modes 42

2 Designing Client Definitions 44
2.1 Specification formats for operands in the client definition 45
2.2 Contents of operands in the client definition 46
2.2.1 Operands related to system configuration 46
2.2.2 Operands related to application program status monitoring 47
2.2.3 Operands related to performance 48
2.2.4 Operands related to SQL 53
2.3 Operand specification rules 57
2.4 Notes about using the function for centrally managing client definitions 58

3 Setting Up an Environment for the JDBC Driver 59
3.1 Environment setup procedure for the JDBC driver 60
3.1.1 Installing Java Runtime Environment or Java Development Kit 60
3.1.2 Installing the JDBC driver 60
3.1.3 Specifying the CLASSPATH environment variable 61
3.1.4 Checking the value of the TZ environment variable 61
3.1.5 Granting the write permission for the trace file output destination directory 62
3.1.6 Setting system properties 62
3.1.7 Reviewing the scope of scans by antivirus software 65
3.2 Handling unresponsive application programs 66
3.3 Upgrading the JDBC driver (replacing the JAR file) 68
3.4 Replacing the JDBC driver with a revised version 70
3.5 Changing the time of the OS on a machine on which the JDBC driver has been installed 71
3.6 Uninstalling the JDBC driver 72

4 Setting Up an Environment for an HADB Client (If the ODBC Driver and CLI Functions Are Used) 73

4.1 HADB client environment setup procedure 74
4.1.1 HADB client for Windows 74
4.1.2 HADB client for Linux 74
4.2 Installing and uninstalling an HADB client 76
4.2.1 HADB client for Windows 76
4.2.2 HADB client for Linux 77
4.3 Specifying environment variables 82
4.3.1 HADB client for Windows 82
4.3.2 HADB client for Linux 83
4.4 Creating a client definition 85
4.4.1 How to create a client definition 85
4.4.2 Notes about changing a client definition 85
4.4.3 Choosing a client definition 85
4.5 Handling unresponsive application programs 86
4.6 Upgrading an HADB client 88
4.6.1 Preparations before upgrading an HADB client 88
4.6.2 Notes about upgrading 88
4.6.3 How to upgrade an HADB client 89
4.6.4 Tasks to be performed after upgrading 90
4.7 Downgrading an HADB client version (restoring the previous version) 92
4.7.1 Preparations before downgrading an HADB client 92
4.7.2 Notes about downgrading 92
4.7.3 Downgrade procedure 92
4.7.4 Tasks to be performed after downgrading 94
4.8 Replacing HADB client with a revised version 95
4.8.1 Procedure for replacing HADB client with a revised version 95
4.9 Changing the OS time on a client machine 97
4.9.1 Notes (changing the OS time) 97
4.9.2 How to advance the OS time on a client machine 97
4.9.3 How to restore the OS time on a client machine 97

Part 2: Application Program Creation (Common)

5 Designs Related to Improvement of Application Program Performance 99

5.1 How to retrieve tables 100
5.1.1 About table scans 100
5.1.2 About index scans 101
5.1.3 About key scans 102
5.2 B-tree indexes and text indexes used during execution of SQL statements 105

5.2.1 Priority and selection rules for indexes 106
5.2.2 Examples of indexes that are used during retrieval of a table 113
5.2.3 Examples of indexes that are used during retrieval of a table (examples of index priority) 115
5.2.4 Cases where an index is not used 120
5.2.5 How to check the index used during execution of an SQL statement 121
5.2.6 Notes on searching using a text index 122
5.3 Range indexes used during execution of SQL statements 123
5.3.1 Conditions under which range indexes are used during execution of an SQL statement 123
5.3.2 Examples of range indexes used during retrieval 125
5.3.3 How to check the range index used during execution of an SQL statement 130
5.4 How to evaluate the search conditions when indexes are used 131
5.4.1 Evaluation method when B-tree indexes are used 131
5.4.2 Evaluation method when range indexes are used 133
5.5 Table joining methods 137
5.5.1 About nested-loop join 137
5.5.2 About hash join 138
5.5.3 Characteristics of the joining methods 145
5.6 How to process subqueries 146
5.6.1 Methods for processing subqueries that do not contain an external reference column 146
5.6.2 Characteristics of the methods for processing subqueries that do not contain an external reference column 152
5.6.3 Methods for processing subqueries that contain an external reference column 152
5.6.4 Characteristics of the methods for processing subqueries that contain an external reference column 158
5.7 Grouping methods 159
5.7.1 Hash grouping 159
5.7.2 Sort grouping 161
5.7.3 Characteristics of each type of grouping 161
5.8 Methods for processing set operations 163
5.8.1 Hash execution 163
5.8.2 Work table execution 165
5.8.3 Characteristics of the methods for processing set operations 166
5.9 Method for processing SELECT DISTINCT 167
5.9.1 Hash execution 167
5.9.2 Work table execution 169
5.9.3 Characteristics of the methods for processing SELECT DISTINCT 170
5.10 Considerations when executing an SQL statement that creates work tables 171
5.10.1 Types of work tables 171
5.10.2 Work tables created when SQL statements are executed 172
5.10.3 Number of work tables that are created 176
5.11 Equivalent exchange of search conditions 184
5.11.1 Equivalent exchange for OR conditions (removing from the OR conditions) 184

5.11.2 Equivalent exchange for OR conditions (converting to IN conditions) 189
5.11.3 Equivalent exchange for OR conditions (equivalent exchange to derived tables for which the UNION ALL set operation is specified) 191
5.11.4 Equivalent exchange for scalar operations 197
5.11.5 Equivalent exchange for an IN predicate 199
5.11.6 Equivalent exchange for a HAVING clause (converting to the WHERE clauses) 200
5.11.7 Equivalent exchange for search conditions in SQL statements that specify derived queries (transposition to the WHERE clause of a derived query) 201
5.12 Considerations when searching an archivable multi-chunk table 206
5.12.1 Tips for searching an archivable multi-chunk table 206
5.12.2 Using the datetime information of the archive range column to narrow the search range 208
5.12.3 Notes about specifying JOIN (joined table) 213
5.12.4 Equivalent exchange of SQL statements that search archivable multi-chunk tables 215
5.13 Expanding internal derived tables 218
5.14 Improving performance by batch transfer of retrieval results 219
5.15 Batch transfer of dynamic parameter values 221

6 Tuning Application Programs 223
6.1 How to use access paths (how to use SQL statement execution plans) 224
6.1.1 About access paths 224
6.1.2 How to check access paths 226
6.1.3 Examples of access paths 228
6.1.4 Information displayed in the tree view 230
6.1.5 Information displayed in the details view 246
6.1.6 Information output in identification information view (SQL statement identification information) 256
6.1.7 Information displayed for an access path (alphabetical order) 257

Part 3: Application Program Creation (JDBC)

7 Creating Application Programs 262
7.1 JDBC driver provided by HADB 263
7.1.1 Scope of JDBC standards compliance 263
7.1.2 Package name and directory structure of the JAR file 265
7.2 Basic procedure for application program processing 266
7.3 How to connect to the HADB server 267
7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server 267
7.3.2 Using the getConnection method of the DataSource class to connect to the HADB server 277
7.3.3 Connection information priorities 280
7.4 Retrieving data (executing the SELECT statement) 285
7.4.1 How to retrieve data 285
7.4.2 How to use dynamic parameters 287
7.5 Adding, updating, or deleting data (executing the INSERT, UPDATE, or DELETE statement) 289
7.6 Data processing 290

7.6.1 Mapping data types 290
7.6.2 Data conversion process 294
7.6.3 Overflow handling 296
7.6.4 Conversion of character encoding 301
7.7 Troubleshooting 303
7.7.1 JDBC interface method traces 303
7.7.2 Exception trace log 305
7.8 Scalar functions that can be specified in the escape clause 319

8 The JDBC 1.2 API 321
8.1 Driver interface 322
8.1.1 List of the methods in the Driver interface 322
8.1.2 acceptsURL(String url) 323
8.1.3 connect(String url, Properties info) 323
8.1.4 getMajorVersion() 324
8.1.5 getMinorVersion() 325
8.1.6 getPropertyInfo(String url, Properties info) 325
8.1.7 jdbcCompliant() 328
8.1.8 Escape clause 328
8.2 Connection interface 330
8.2.1 List of the methods in the Connection interface 330
8.2.2 clearWarnings() 332
8.2.3 close() 333
8.2.4 commit() 333
8.2.5 createStatement() 334
8.2.6 createStatement(int resultSetType, int resultSetConcurrency) 335
8.2.7 createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) 336
8.2.8 getAutoCommit() 337
8.2.9 getCatalog() 337
8.2.10 getHADBConnectionID() 338
8.2.11 getHADBConnectionSerialNum() 338
8.2.12 getHADBOrderMode() 339
8.2.13 getHADBSQLHashFltSize() 339
8.2.14 getHADBSQLHashTblSize() 340
8.2.15 getHADBSQLMaxRthdNum() 341
8.2.16 getHADBTransactionID() 341
8.2.17 getHoldability() 342
8.2.18 getMetaData() 343
8.2.19 getSchema() 343
8.2.20 getTransactionIsolation() 344
8.2.21 getTypeMap() 344
8.2.22 getWarnings() 345

8.2.23 isClosed() 345
8.2.24 isReadOnly() 346
8.2.25 isValid(int timeout) 346
8.2.26 nativeSQL(String sql) 347
8.2.27 prepareStatement(String sql) 350
8.2.28 prepareStatement(String sql, int resultSetType, int resultSetConcurrency) 351
8.2.29 prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) 352
8.2.30 rollback() 353
8.2.31 setAutoCommit(boolean autoCommit) 354
8.2.32 setCatalog(String catalog) 354
8.2.33 setHADBAuditInfo(int pos,String userinfo) 355
8.2.34 setHADBOrderMode(int mode) 356
8.2.35 setHADBSQLHashFltSize(int areaSize) 358
8.2.36 setHADBSQLHashTblSize(int areaSize) 359
8.2.37 setHADBSQLMaxRthdNum(int rthdNum) 360
8.2.38 setHoldability(int holdability) 363
8.2.39 setReadOnly(boolean readOnly) 363
8.2.40 setSchema(String schema) 364
8.2.41 setTransactionIsolation(int level) 365
8.3 Statement interface 367
8.3.1 List of the methods in the Statement interface 367
8.3.2 addBatch(String sql) 369
8.3.3 cancel() 370
8.3.4 clearBatch() 371
8.3.5 clearWarnings() 371
8.3.6 close() 372
8.3.7 closeOnCompletion() 372
8.3.8 execute(String sql) 373
8.3.9 executeBatch() 374
8.3.10 executeLargeBatch() 375
8.3.11 executeLargeUpdate(String sql) 375
8.3.12 executeQuery(String sql) 376
8.3.13 executeUpdate(String sql) 377
8.3.14 getConnection() 378
8.3.15 getFetchDirection() 378
8.3.16 getFetchSize() 379
8.3.17 getHADBSQLSerialNum() 379
8.3.18 getHADBStatementHandle() 380
8.3.19 getLargeMaxRows() 381
8.3.20 getLargeUpdateCount() 381
8.3.21 getMaxFieldSize() 382

8.3.22 getMaxRows() 383
8.3.23 getMoreResults() 383
8.3.24 getQueryTimeout() 384
8.3.25 getResultSet() 384
8.3.26 getResultSetConcurrency() 385
8.3.27 getResultSetHoldability() 385
8.3.28 getResultSetType() 386
8.3.29 getUpdateCount() 387
8.3.30 getWarnings() 388
8.3.31 isClosed() 388
8.3.32 isCloseOnCompletion() 389
8.3.33 isPoolable() 389
8.3.34 setCursorName(String name) 390
8.3.35 setEscapeProcessing(boolean enable) 391
8.3.36 setFetchDirection(int direction) 391
8.3.37 setFetchSize(int rows) 392
8.3.38 setLargeMaxRows(long max) 393
8.3.39 setMaxFieldSize(int max) 394
8.3.40 setMaxRows(int max) 395
8.3.41 setQueryTimeout(int seconds) 395
8.3.42 Notes about the Statement interface 396
8.4 PreparedStatement interface 398
8.4.1 List of the methods in the PreparedStatement interface 398
8.4.2 addBatch() 400
8.4.3 clearParameters() 400
8.4.4 execute() 401
8.4.5 executeLargeUpdate() 402
8.4.6 executeQuery() 402
8.4.7 executeUpdate() 403
8.4.8 getHADBSQLSerialNum() 404
8.4.9 getHADBStatementHandle() 404
8.4.10 getMetaData() 405
8.4.11 getParameterMetaData() 406
8.4.12 setAsciiStream(int parameterIndex, InputStream x, int length) 406
8.4.13 setBigDecimal(int parameterIndex, BigDecimal x) 407
8.4.14 setBinaryStream(int parameterIndex, InputStream x, int length) 408
8.4.15 setBoolean(int parameterIndex, boolean x) 409
8.4.16 setByte(int parameterIndex, byte x) 409
8.4.17 setBytes(int parameterIndex, byte[] x) 410
8.4.18 setCharacterStream(int parameterIndex, Reader reader, int length) 411
8.4.19 setDate(int parameterIndex, Date x) 411

8.4.20 setDate(int parameterIndex, Date x, Calendar cal) 412
8.4.21 setDouble(int parameterIndex, double x) 413
8.4.22 setFloat(int parameterIndex, float x) 414
8.4.23 setInt(int parameterIndex, int x) 414
8.4.24 setLong(int parameterIndex, long x) 415
8.4.25 setNull(int parameterIndex,int sqlType) 416
8.4.26 setObject(int parameterIndex, Object x) 416
8.4.27 setObject(int parameterIndex, Object x, int targetSqlType) 417
8.4.28 setObject(int parameterIndex, Object x, int targetSqlType, int scale) 418
8.4.29 setShort(int parameterIndex, short x) 419
8.4.30 setString(int parameterIndex, String x) 420
8.4.31 setTime(int parameterIndex, Time x) 420
8.4.32 setTime(int parameterIndex, Time x, Calendar cal) 421
8.4.33 setTimestamp(int parameterIndex, Timestamp x) 422
8.4.34 setTimestamp(int parameterIndex, Timestamp x, Calendar cal) 423
8.4.35 Notes about the PreparedStatement interface 423
8.5 ResultSet interface 426
8.5.1 List of the methods in the ResultSet interface 426
8.5.2 absolute(int row) 429
8.5.3 afterLast() 430
8.5.4 beforeFirst() 431
8.5.5 clearWarnings() 432
8.5.6 close() 432
8.5.7 findColumn(String columnName) 433
8.5.8 first() 434
8.5.9 getAsciiStream(int columnIndex) 434
8.5.10 getAsciiStream(String columnName) 435
8.5.11 getBigDecimal(int columnIndex) 436
8.5.12 getBigDecimal(String columnName) 438
8.5.13 getBinaryStream(int columnIndex) 439
8.5.14 getBinaryStream(String columnName) 440
8.5.15 getBoolean(int columnIndex) 440
8.5.16 getBoolean(String columnName) 442
8.5.17 getByte(int columnIndex) 443
8.5.18 getByte(String columnName) 444
8.5.19 getBytes(int columnIndex) 445
8.5.20 getBytes(String columnName) 446
8.5.21 getCharacterStream(int columnIndex) 447
8.5.22 getCharacterStream(String columnName) 448
8.5.23 getConcurrency() 449
8.5.24 getCursorName() 449

8.5.25 getDate(int columnIndex) 450
8.5.26 getDate(int columnIndex, Calendar cal) 451
8.5.27 getDate(String columnName) 452
8.5.28 getDate(String columnName, Calendar cal) 453
8.5.29 getDouble(int columnIndex) 454
8.5.30 getDouble(String columnName) 456
8.5.31 getFetchDirection() 457
8.5.32 getFetchSize() 457
8.5.33 getFloat(int columnIndex) 458
8.5.34 getFloat(String columnName) 460
8.5.35 getHoldability() 461
8.5.36 getInt(int columnIndex) 462
8.5.37 getInt(String columnName) 463
8.5.38 getLong(int columnIndex) 464
8.5.39 getLong(String columnName) 466
8.5.40 getMetaData() 467
8.5.41 getObject(int columnIndex) 467
8.5.42 getObject(String columnName) 469
8.5.43 getObject(int columnIndex,Class<T> type) 470
8.5.44 getObject(String columnLabel,Class<T> type) 472
8.5.45 getRow() 473
8.5.46 getShort(int columnIndex) 473
8.5.47 getShort(String columnName) 475
8.5.48 getStatement() 476
8.5.49 getString(int columnIndex) 476
8.5.50 getString(String columnName) 478
8.5.51 getTime(int columnIndex) 479
8.5.52 getTime(int columnIndex, Calendar cal) 480
8.5.53 getTime(String columnName) 481
8.5.54 getTime(String columnName, Calendar cal) 482
8.5.55 getTimestamp(int columnIndex) 483
8.5.56 getTimestamp(int columnIndex, Calendar cal) 484
8.5.57 getTimestamp(String columnName) 485
8.5.58 getTimestamp(String columnName, Calendar cal) 486
8.5.59 getType() 486
8.5.60 getWarnings() 487
8.5.61 isAfterLast() 488
8.5.62 isBeforeFirst() 488
8.5.63 isClosed() 489
8.5.64 isFirst() 490
8.5.65 isLast() 490

8.5.66 last() 491
8.5.67 next() 492
8.5.68 previous() 492
8.5.69 relative(int rows) 493
8.5.70 setFetchDirection(int direction) 494
8.5.71 setFetchSize(int rows) 494
8.5.72 wasNull() 495
8.5.73 Fields supported by the ResultSet interface 496
8.5.74 Notes about the ResultSet interface 497
8.6 DatabaseMetaData interface 500
8.6.1 List of the methods in the DatabaseMetaData interface 500
8.6.2 allProceduresAreCallable() 509
8.6.3 allTablesAreSelectable() 509
8.6.4 autoCommitFailureClosesAllResultSets() 510
8.6.5 dataDefinitionCausesTransactionCommit() 510
8.6.6 dataDefinitionIgnoredInTransactions() 511
8.6.7 deletesAreDetected(int type) 511
8.6.8 doesMaxRowSizeIncludeBlobs() 512
8.6.9 generatedKeyAlwaysReturned() 513
8.6.10 getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern) 513
8.6.11 getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable) 514
8.6.12 getCatalogs() 516
8.6.13 getCatalogSeparator() 516
8.6.14 getCatalogTerm() 517
8.6.15 getClientInfoProperties() 517
8.6.16 getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern) 518
8.6.17 getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) 519
8.6.18 getConnection() 521
8.6.19 getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable) 522
8.6.20 getDatabaseMajorVersion() 523
8.6.21 getDatabaseMinorVersion() 524
8.6.22 getDatabaseProductName() 525
8.6.23 getDatabaseProductVersion() 525
8.6.24 getDefaultTransactionIsolation() 526
8.6.25 getDriverMajorVersion() 526
8.6.26 getDriverMinorVersion() 527
8.6.27 getDriverName() 527
8.6.28 getDriverVersion() 528
8.6.29 getExportedKeys(String catalog, String schema, String table) 528

8.6.30 getExtraNameCharacters() 530
8.6.31 getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern) 530
8.6.32 getFunctions(String catalog, String schemaPattern, String functionNamePattern) 531
8.6.33 getIdentifierQuoteString() 532
8.6.34 getImportedKeys(String catalog, String schema, String table) 533
8.6.35 getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate) 534
8.6.36 getJDBCMajorVersion() 536
8.6.37 getJDBCMinorVersion() 537
8.6.38 getMaxBinaryLiteralLength() 537
8.6.39 getMaxCatalogNameLength() 538
8.6.40 getMaxCharLiteralLength() 538
8.6.41 getMaxColumnNameLength() 539
8.6.42 getMaxColumnsInGroupBy() 539
8.6.43 getMaxColumnsInIndex() 540
8.6.44 getMaxColumnsInOrderBy() 540
8.6.45 getMaxColumnsInSelect() 541
8.6.46 getMaxColumnsInTable() 541
8.6.47 getMaxConnections() 542
8.6.48 getMaxCursorNameLength() 542
8.6.49 getMaxIndexLength() 543
8.6.50 getMaxLogicalLobSize() 543
8.6.51 getMaxProcedureNameLength() 544
8.6.52 getMaxRowSize() 544
8.6.53 getMaxSchemaNameLength() 545
8.6.54 getMaxStatementLength() 545
8.6.55 getMaxStatements() 546
8.6.56 getMaxTableNameLength() 546
8.6.57 getMaxTablesInSelect() 547
8.6.58 getMaxUserNameLength() 547
8.6.59 getNumericFunctions() 548
8.6.60 getPrimaryKeys(String catalog, String schema, String table) 548
8.6.61 getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern) 549
8.6.62 getProcedures(String catalog, String schemaPattern, String procedureNamePattern) 551
8.6.63 getProcedureTerm() 552
8.6.64 getPseudoColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern) 552
8.6.65 getResultSetHoldability() 553
8.6.66 getRowIdLifetime() 554
8.6.67 getSchemas() 554
8.6.68 getSchemas(String catalog, String schemaPattern) 555

8.6.69 getSchemaTerm() 556
8.6.70 getSearchStringEscape() 556
8.6.71 getSQLKeywords() 557
8.6.72 getSQLStateType() 557
8.6.73 getStringFunctions() 558
8.6.74 getSuperTables(String catalog, String schemaPattern, String tableNamePattern) 558
8.6.75 getSuperTypes(String catalog, String schemaPattern, String typeNamePattern) 559
8.6.76 getSystemFunctions() 560
8.6.77 getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern) 561
8.6.78 getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) 562
8.6.79 getTableTypes() 564
8.6.80 getTimeDateFunctions() 564
8.6.81 getTypeInfo() 565
8.6.82 getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types) 566
8.6.83 getURL() 567
8.6.84 getUserName() 568
8.6.85 getVersionColumns(String catalog, String schema, String table) 568
8.6.86 insertsAreDetected(int type) 569
8.6.87 isCatalogAtStart() 570
8.6.88 isReadOnly() 570
8.6.89 locatorsUpdateCopy() 571
8.6.90 nullPlusNonNullIsNull() 571
8.6.91 nullsAreSortedAtEnd() 572
8.6.92 nullsAreSortedAtStart() 572
8.6.93 nullsAreSortedHigh() 573
8.6.94 nullsAreSortedLow() 573
8.6.95 othersDeletesAreVisible(int type) 574
8.6.96 othersInsertsAreVisible(int type) 574
8.6.97 othersUpdatesAreVisible(int type) 575
8.6.98 ownDeletesAreVisible(int type) 575
8.6.99 ownInsertsAreVisible(int type) 576
8.6.100 ownUpdatesAreVisible(int type) 577
8.6.101 storesLowerCaseIdentifiers() 577
8.6.102 storesLowerCaseQuotedIdentifiers() 578
8.6.103 storesMixedCaseIdentifiers() 578
8.6.104 storesMixedCaseQuotedIdentifiers() 579
8.6.105 storesUpperCaseIdentifiers() 579
8.6.106 storesUpperCaseQuotedIdentifiers() 580
8.6.107 supportsAlterTableWithAddColumn() 580
8.6.108 supportsAlterTableWithDropColumn() 581
8.6.109 supportsANSI92EntryLevelSQL() 581

8.6.110 supportsANSI92FullSQL() 582
8.6.111 supportsANSI92IntermediateSQL() 582
8.6.112 supportsBatchUpdates() 583
8.6.113 supportsCatalogsInDataManipulation() 583
8.6.114 supportsCatalogsInIndexDefinitions() 584
8.6.115 supportsCatalogsInPrivilegeDefinitions() 584
8.6.116 supportsCatalogsInProcedureCalls() 585
8.6.117 supportsCatalogsInTableDefinitions() 585
8.6.118 supportsColumnAliasing() 586
8.6.119 supportsConvert() 586
8.6.120 supportsConvert(int fromType, int toType) 587
8.6.121 supportsCoreSQLGrammar() 588
8.6.122 supportsCorrelatedSubqueries() 589
8.6.123 supportsDataDefinitionAndDataManipulationTransactions() 589
8.6.124 supportsDataManipulationTransactionsOnly() 590
8.6.125 supportsDifferentTableCorrelationNames() 590
8.6.126 supportsExpressionsInOrderBy() 591
8.6.127 supportsExtendedSQLGrammar() 591
8.6.128 supportsFullOuterJoins() 592
8.6.129 supportsGetGeneratedKeys() 592
8.6.130 supportsGroupBy() 593
8.6.131 supportsGroupByBeyondSelect() 593
8.6.132 supportsGroupByUnrelated() 594
8.6.133 supportsIntegrityEnhancementFacility() 594
8.6.134 supportsLikeEscapeClause() 595
8.6.135 supportsLimitedOuterJoins() 595
8.6.136 supportsMinimumSQLGrammar() 596
8.6.137 supportsMixedCaseIdentifiers() 596
8.6.138 supportsMixedCaseQuotedIdentifiers() 597
8.6.139 supportsMultipleOpenResults() 597
8.6.140 supportsMultipleResultSets() 598
8.6.141 supportsMultipleTransactions() 598
8.6.142 supportsNamedParameters() 599
8.6.143 supportsNonNullableColumns() 599
8.6.144 supportsOpenCursorsAcrossCommit() 600
8.6.145 supportsOpenCursorsAcrossRollback() 600
8.6.146 supportsOpenStatementsAcrossCommit() 601
8.6.147 supportsOpenStatementsAcrossRollback() 601
8.6.148 supportsOrderByUnrelated() 602
8.6.149 supportsOuterJoins() 602
8.6.150 supportsPositionedDelete() 603

8.6.151 supportsPositionedUpdate() 603
8.6.152 supportsRefCursors() 604
8.6.153 supportsResultSetConcurrency(int type, int concurrency) 604
8.6.154 supportsResultSetHoldability(int holdability) 605
8.6.155 supportsResultSetType(int type) 606
8.6.156 supportsSavepoints() 606
8.6.157 supportsSchemasInDataManipulation() 607
8.6.158 supportsSchemasInIndexDefinitions() 607
8.6.159 supportsSchemasInPrivilegeDefinitions() 608
8.6.160 supportsSchemasInProcedureCalls() 608
8.6.161 supportsSchemasInTableDefinitions() 609
8.6.162 supportsSelectForUpdate() 609
8.6.163 supportsStatementPooling() 610
8.6.164 supportsStoredFunctionsUsingCallSyntax() 610
8.6.165 supportsStoredProcedures() 611
8.6.166 supportsSubqueriesInComparisons() 611
8.6.167 supportsSubqueriesInExists() 612
8.6.168 supportsSubqueriesInIns() 612
8.6.169 supportsSubqueriesInQuantifieds() 613
8.6.170 supportsTableCorrelationNames() 613
8.6.171 supportsTransactionIsolationLevel(int level) 614
8.6.172 supportsTransactions() 615
8.6.173 supportsUnion() 615
8.6.174 supportsUnionAll() 616
8.6.175 updatesAreDetected(int type) 616
8.6.176 usesLocalFilePerTable() 617
8.6.177 usesLocalFiles() 617
8.7 ResultSetMetaData interface 618
8.7.1 List of the methods in the ResultSetMetaData interface 618
8.7.2 getCatalogName(int column) 619
8.7.3 getColumnClassName(int column) 620
8.7.4 getColumnCount() 621
8.7.5 getColumnDisplaySize(int column) 621
8.7.6 getColumnLabel(int column) 623
8.7.7 getColumnName(int column) 623
8.7.8 getColumnType(int column) 624
8.7.9 getColumnTypeName(int column) 624
8.7.10 getPrecision(int column) 625
8.7.11 getScale(int column) 627
8.7.12 getSchemaName(int column) 627
8.7.13 getTableName(int column) 628

8.7.14 isAutoIncrement(int column) 629
8.7.15 isCaseSensitive(int column) 629
8.7.16 isCurrency(int column) 630
8.7.17 isDefinitelyWritable(int column) 630
8.7.18 isNullable(int column) 631
8.7.19 isReadOnly(int column) 631
8.7.20 isSearchable(int column) 632
8.7.21 isSigned(int column) 633
8.7.22 isWritable(int column) 633
8.8 SQLException interface 635
8.9 SQLWarning interface 636
8.9.1 Creating an SQLWarning object 636
8.9.2 Releasing SQLWarning objects 636
8.10 Unsupported interfaces 637

9 The JDBC 2.1 Core API 638
9.1 Scope of support for the result set extended functions 639
9.2 Scope of support for batch update functionality 640
9.2.1 SQL statements that can use the batch update functionality 640
9.2.2 Batch update functionality with the Statement class 640
9.2.3 Batch update functionality with the PreparedStatement class 640
9.2.4 Notes 641
9.3 Added data types 643
9.4 Unsupported interfaces 644

10 The JDBC 2.0 Optional Package 645
10.1 HADB's scope of support for the functions added in the JDBC 2.0 Optional Package 646
10.2 DataSource interface 647
10.2.1 List of the methods in the DataSource interface 647
10.2.2 getConnection() 647
10.2.3 getConnection(String username, String password) 648
10.2.4 getLoginTimeout() 649
10.2.5 getLogWriter() 649
10.2.6 setLoginTimeout(int seconds) 650
10.2.7 setLogWriter(PrintWriter out) 651
10.3 ConnectionPoolDataSource interface 652
10.3.1 List of the methods in the ConnectionPoolDataSource interface 652
10.3.2 getLoginTimeout() 652
10.3.3 getLogWriter() 653
10.3.4 getPooledConnection() 653
10.3.5 getPooledConnection(String user, String password) 654
10.3.6 setLoginTimeout(int seconds) 655

10.3.7 setLogWriter(PrintWriter out) 655
10.4 PooledConnection interface 657
10.4.1 List of the methods in the PooledConnection interface 657
10.4.2 addConnectionEventListener(ConnectionEventListener listener) 657
10.4.3 close() 658
10.4.4 getConnection() 658
10.4.5 removeConnectionEventListener(ConnectionEventListener listener) 659
10.5 Connection information setup and acquisition interface 661
10.5.1 List of the methods in the connection information setup and acquisition interface 661
10.5.2 getApName() 662
10.5.3 getEncodeLang() 662
10.5.4 getInterfaceMethodTrace() 663
10.5.5 getNotErrorOccurred() 663
10.5.6 getPassword() 664
10.5.7 getSQLWarningKeep() 665
10.5.8 getTraceNumber() 665
10.5.9 getUser() 666
10.5.10 getHostName() 666
10.5.11 getPort() 667
10.5.12 setApName(String name) 667
10.5.13 setEncodeLang(String lang) 668
10.5.14 setInterfaceMethodTrace(boolean flag) 669
10.5.15 setNotErrorOccurred(boolean mode) 670
10.5.16 setPassword(String password) 670
10.5.17 setSQLWarningKeep(boolean mode) 671
10.5.18 setTraceNumber(int num) 672
10.5.19 setUser(String user) 672
10.5.20 setHostName(String name) 673
10.5.21 setPort(int port) 673

11 The JDBC 3.0 API 675

11.1 HADB's scope of support for the functions added in the JDBC 3.0 API 676
11.2 ParameterMetaData interface 677
11.2.1 List of the methods in the ParameterMetaData interface 677
11.2.2 getParameterClassName(int param) 678
11.2.3 getParameterCount() 679
11.2.4 getParameterMode(int param) 679
11.2.5 getParameterType(int param) 680
11.2.6 getParameterTypeName(int param) 680
11.2.7 getPrecision(int param) 681
11.2.8 getScale(int param) 682
11.2.9 isNullable(int param) 683

11.2.10 isSigned(int param) 684
11.3 Unsupported interfaces 685

12 The JDBC 4.0 API 686
12.1 HADB's scope of support for the functions added in the JDBC 4.0 API 687
12.1.1 Automatic loading of java.sql.Driver 687
12.1.2 Wrapper pattern 688
12.1.3 SQL exception extension 688
12.1.4 Connection management 689
12.1.5 Added scalar functions 689
12.2 Wrapper interface 690
12.2.1 List of the methods in the Wrapper interface 690
12.2.2 isWrapperFor(Class<?> iface) 691
12.2.3 unwrap(Class<T> iface) 691
12.3 SQL exception extension function 693
12.4 Unsupported interfaces 695

13 The JDBC 4.1 API 696
13.1 HADB's scope of support for the functions added in the JDBC 4.1 API 697
13.1.1 try-with-resources statement 697
13.1.2 Closing Statement objects when their dependent objects close 697

14 The JDBC 4.2 API 698
14.1 HADB's scope of support for the functions added in the JDBC 4.2 API 699
14.1.1 Large update counts 699

Part 4: Application Program Creation (ODBC)

15 Creating Application Programs 700
15.1 ODBC driver provided by HADB 701
15.1.1 ODBC driver version with which the HADB ODBC driver is compliant 701
15.1.2 System configuration 701
15.1.3 About conversion of character encoding 702
15.1.4 About using the ODBC cursor library 704
15.2 HADB ODBC driver environment setup 705
15.2.1 Specifying data sources 705
15.2.2 Registering the registry key 706
15.2.3 Deleting data sources 706
15.3 Correspondence between data types 707
15.3.1 Correspondence between ODBC's SQL data types and HADB's data types 707
15.3.2 Correspondence between ODBC's SQL data types and C data types 708
15.3.3 Notes about data type conversion 710

15.4 Information that is returned in the event of an error 713
15.5 Limitations 714
15.5.1 ROW specification 714
15.5.2 AUTOCOMMIT specifications 714
15.5.3 Notes about the maximum number SQL processing real threads 714
15.6 Notes 715
15.6.1 Effects of update operations on a retrieval using a cursor 715
15.6.2 Notes on using the HADB ODBC driver in ODBC 2.x applications 715

16 ODBC Functions 717
16.1 List of ODBC functions 718
16.2 Notes about SQLxxx and SQLxxxW functions 722
16.3 Connecting to the data source 723
16.3.1 SQLAllocHandle 723
16.3.2 SQLConnect, SQLConnectW 724
16.3.3 SQLDriverConnect, SQLDriverConnectW 727
16.3.4 SQLBrowseConnect, SQLBrowseConnectW 733
16.4 Acquiring driver and data source information 738
16.4.1 SQLDataSources, SQLDataSourcesW 738
16.4.2 SQLDrivers, SQLDriversW 740
16.4.3 SQLGetInfo, SQLGetInfoW 742
16.4.4 SQLGetFunctions 744
16.4.5 SQLGetTypeInfo, SQLGetTypeInfoW 745
16.5 Specifying and obtaining driver options 749
16.5.1 SQLSetConnectAttr, SQLSetConnectAttrW 749
16.5.2 SQLGetConnectAttr, SQLGetConnectAttrW 751
16.5.3 SQLSetEnvAttr 753
16.5.4 SQLGetEnvAttr 754
16.5.5 SQLSetStmtAttr, SQLSetStmtAttrW 756
16.5.6 SQLGetStmtAttr, SQLGetStmtAttrW 757
16.6 Specifying descriptor values 760
16.6.1 SQLGetDescField, SQLGetDescFieldW 760
16.6.2 SQLGetDescRec, SQLGetDescRecW 762
16.6.3 SQLSetDescField, SQLSetDescFieldW 765
16.6.4 SQLSetDescRec 767
16.6.5 SQLCopyDesc 769
16.7 Creating SQL requests 771
16.7.1 SQLPrepare, SQLPrepareW 771
16.7.2 SQLBindParameter 773
16.7.3 SQLGetCursorName, SQLGetCursorNameW 776
16.7.4 SQLSetCursorName, SQLSetCursorNameW 778
16.7.5 SQLDescribeParam 780

16.7.6 SQLNumParams 781
16.8 Executing SQL statements 783
16.8.1 SQLExecute 783
16.8.2 SQLExecDirect, SQLExecDirectW 785
16.8.3 SQLNativeSql, SQLNativeSqlW 789
16.8.4 SQLParamData 793
16.8.5 SQLPutData 795
16.9 Acquiring execution results and execution result information 798
16.9.1 SQLRowCount 798
16.9.2 SQLNumResultCols 799
16.9.3 SQLDescribeCol, SQLDescribeColW 800
16.9.4 SQLColAttribute, SQLColAttributeW 803
16.9.5 SQLBindCol 806
16.9.6 SQLFetch 808
16.9.7 SQLFetchScroll 810
16.9.8 SQLGetData 812
16.9.9 SQLSetPos 815
16.9.10 SQLBulkOperations 817
16.9.11 SQLMoreResults 819
16.9.12 SQLGetDiagField, SQLGetDiagFieldW 820
16.9.13 SQLGetDiagRec, SQLGetDiagRecW 822
16.10 Acquiring system information for the data source 826
16.10.1 SQLColumnPrivileges, SQLColumnPrivilegesW 826
16.10.2 SQLColumns, SQLColumnsW 829
16.10.3 SQLForeignKeys, SQLForeignKeysW 833
16.10.4 SQLPrimaryKeys, SQLPrimaryKeysW 838
16.10.5 SQLProcedureColumns, SQLProcedureColumnsW 840
16.10.6 SQLProcedures, SQLProceduresW 842
16.10.7 SQLSpecialColumns, SQLSpecialColumnsW 844
16.10.8 SQLStatistics, SQLStatisticsW 847
16.10.9 SQLTablePrivileges, SQLTablePrivilegesW 851
16.10.10 SQLTables, SQLTablesW 854
16.11 Terminating execution of SQL statements 858
16.11.1 SQLFreeStmt 858
16.11.2 SQLCloseCursor 859
16.11.3 SQLCancel 860
16.11.4 SQLEndTran 861
16.12 Disconnecting from the data source 864
16.12.1 SQLDisconnect 864
16.12.2 SQLFreeHandle 865
16.13 Information types that can be specified for InfoType in SQLGetInfo and SQLGetInfoW 867

16.14 Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW 887
16.15 Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr 891
16.16 Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW 893
16.17 Attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW 898
16.18 Attributes that can be specified in DiagIdentifier of SQLGetDiagField and SQLGetDiagFieldW 906

17 Troubleshooting 909
17.1 Information used for troubleshooting 910
17.1.1 Messages output by BI tools and ODBC modules 910
17.1.2 ODBC traces 910
17.1.3 HADB ODBC driver trace information 910
17.1.4 Messages output by the HADB server and HADB client 911
17.1.5 SQL trace information 911
17.2 Troubleshooting procedure 912
17.2.1 Handling errors 912
17.2.2 Troubleshooting tips 914
17.3 Settings for outputting HADB ODBC driver trace information 917
17.3.1 Configuration in ODBC Data Source Administrator 917
17.3.2 Configuration using environment variables 921
17.3.3 Relative priority of configuration in ODBC Data Source Administrator and environment variables 922
17.4 Information output as HADB ODBC driver trace information 923
17.4.1 About trace levels 923
17.4.2 Information output when trace level is 1 924
17.4.3 Information output when trace level is 2 929
17.5 Notes about HADB ODBC driver trace information 934

Part 5: Application Program Creation (CLI Functions)

18 Creating Application Programs 935
18.1 Designing application programs 936
18.1.1 Flow of application program processing 936
18.1.2 Transaction control 936
18.1.3 Flow of processing using dynamic parameters 937
18.1.4 Effects of update operations on a retrieval using a cursor 938
18.1.5 Evaluation and handling of SQL statement errors 939
18.2 How to use the CLI functions 941
18.2.1 Connecting to and disconnecting from the HADB server 941
18.2.2 Referencing data 943
18.2.3 Using dynamic parameters 948
18.2.4 Adding, updating, or deleting data 951

18.2.5 Canceling SQL processing that is executing 952
18.2.6 Notes about using the CLI functions 953
18.3 Compiling and linking application programs 955

19 CLI Functions 956

19.1 List of CLI functions and common rules 957
19.1.1 List of CLI functions 957
19.1.2 Common rules 959
19.2 CLI functions for connecting to and disconnecting from the HADB server 961
19.2.1 a_rdb_SQLAllocConnect() (allocate a connection handle) 961
19.2.2 a_rdb_SQLConnect() (establish a connection) 962
19.2.3 a_rdb_SQLSetConnectAttr() (set connection attributes) 963
19.2.4 a_rdb_SQLDisconnect() (close a connection) 965
19.2.5 a_rdb_SQLFreeConnect() (release a connection handle) 966
19.3 CLI functions for controlling transactions 967
19.3.1 a_rdb_SQLCancel() (cancel SQL processing) 967
19.3.2 a_rdb_SQLEndTran() (terminate the transaction) 968
19.4 CLI functions for execution of SQL statements 970
19.4.1 a_rdb_SQLAllocStmt() (allocate a statement handle) 970
19.4.2 a_rdb_SQLBindArrayParams() (bind dynamic parameters in batch mode) 971
19.4.3 a_rdb_SQLBindCols() (associate retrieval result columns) 973
19.4.4 a_rdb_SQLBindParams() (associate dynamic parameters) 974
19.4.5 a_rdb_SQLCloseCursor() (close the cursor) 975
19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns) 976
19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information) 979
19.4.8 a_rdb_SQLExecDirect() (preprocess and execute an SQL statement) 981
19.4.9 a_rdb_SQLExecute() (execute a preprocessed SQL statement) 982
19.4.10 a_rdb_SQLFetch() (fetch a row) 983
19.4.11 a_rdb_SQLFreeStmt() (release a statement handle) 984
19.4.12 a_rdb_SQLNumParams() (acquire the number of dynamic parameters) 985
19.4.13 a_rdb_SQLNumResultCols() (acquire the number of retrieval result columns) 986
19.4.14 a_rdb_SQLPrepare() (preprocess an SQL statement) 987
19.5 CLI functions for data type conversion 989
19.5.1 a_rdb_CNV_charBINARY() (convert to BINARY-type data) 989
19.5.2 a_rdb_CNV_charDATE() (convert to DATE-type data) 991
19.5.3 a_rdb_CNV_charDECIMAL() (convert to DECIMAL-type data) 992
19.5.4 a_rdb_CNV_charTIME() (convert to TIME-type data) 994
19.5.5 a_rdb_CNV_charTIMESTAMP() (convert to TIMESTAMP-type data) 996
19.5.6 a_rdb_CNV_charVARBINARY() (convert to VARBINARY-type data) 998
19.5.7 a_rdb_CNV_BINARYchar() (convert BINARY-type data) 1000
19.5.8 a_rdb_CNV_DATEchar() (convert DATE-type data) 1002
19.5.9 a_rdb_CNV_DECIMALchar() (convert DECIMAL-type data) 1004

19.5.10 a_rdb_CNV_TIMEchar() (convert TIME-type data) 1006
19.5.11 a_rdb_CNV_TIMESTAMPchar() (convert TIMESTAMP-type data) 1007
19.5.12 a_rdb_CNV_VARBINARYchar() (convert VARBINARY-type data) 1009
19.6 Correspondence to the SQL data types 1012
19.6.1 Correspondences among SQL data types, symbolic literals, and values 1012
19.6.2 Correspondences between SQL data types and data descriptions 1012
19.6.3 Correspondence to the VARCHAR type 1013
19.6.4 VARBINARY type 1014
19.7 Data types used in the CLI functions 1015
19.7.1 a_rdb_SQLColumnInfo_t structure (column information) 1015
19.7.2 a_rdb_SQLNameInfo_t structure (name information) 1016
19.7.3 a_rdb_SQLDataType_t structure (data type information) 1017
19.7.4 a_rdb_SQLInd_t (indicator) 1018
19.7.5 a_rdb_SQLParameterInfo_t structure (parameter information) 1018
19.7.6 a_rdb_SQLResultInfo_t structure (SQL results information) 1019
19.8 Return values of the CLI functions 1022

Appendixes 1024

A Sample Application Program 1025
A.1 Overview of sample application program 1025
A.2 Preparations before executing the sample application program 1025
A.3 How to create the SAMPLE table 1025
A.4 Sample application program execution procedure 1027
B Structure of HADB Client Directories 1029
B.1 HADB clients for Windows 1029
B.2 HADB clients for Linux 1036
C Estimating the Memory Requirements for an HADB Client 1039
C.1 Memory required for connecting to the HADB server 1039
C.2 Memory required for communication between an HADB client and the HADB server 1039

Index 1041

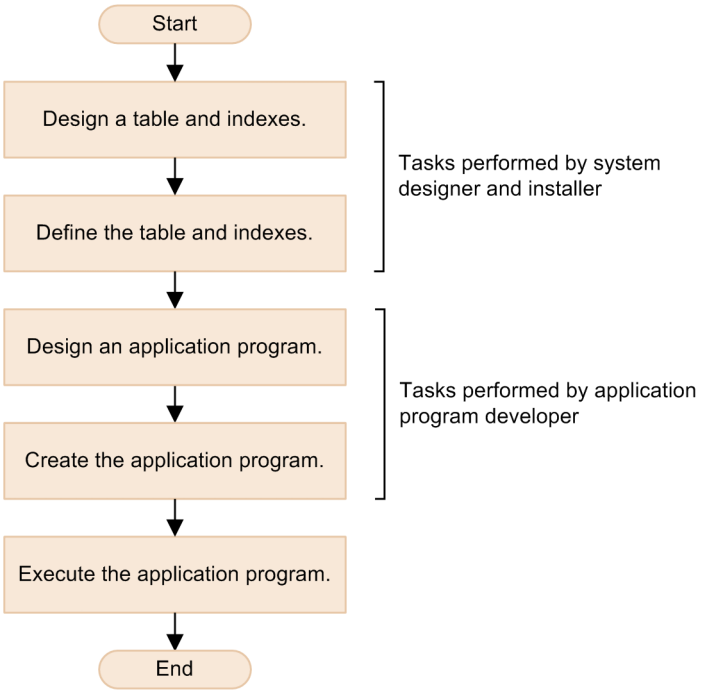
1 Overview of Application Program Development and Execution

This chapter explains the application program development procedure, the prerequisites for application program development, and the application program execution modes.

1.1 Procedure and prerequisites for application program development

The following figure shows the application program development procedure.

Figure 1-1: Application program development procedure



The following subsections explain the prerequisites for developing application programs.

1.1.1 Programming languages for application programs

An application program must be written in one of the following programming languages:

- Java
- C
- C++

In Java, you can use the JDBC API to access the database.

In C or C++, you can use CLI functions (an API provided by HADB that supports C and C++) or ODBC functions to access the database.

You can also use ODBC functions from application programs to access the database.

1.1.2 Character encoding

The following character encodings are supported for HADB servers and HADB clients:

- UTF-8
- Shift-JIS

1.1.3 Application program development environment

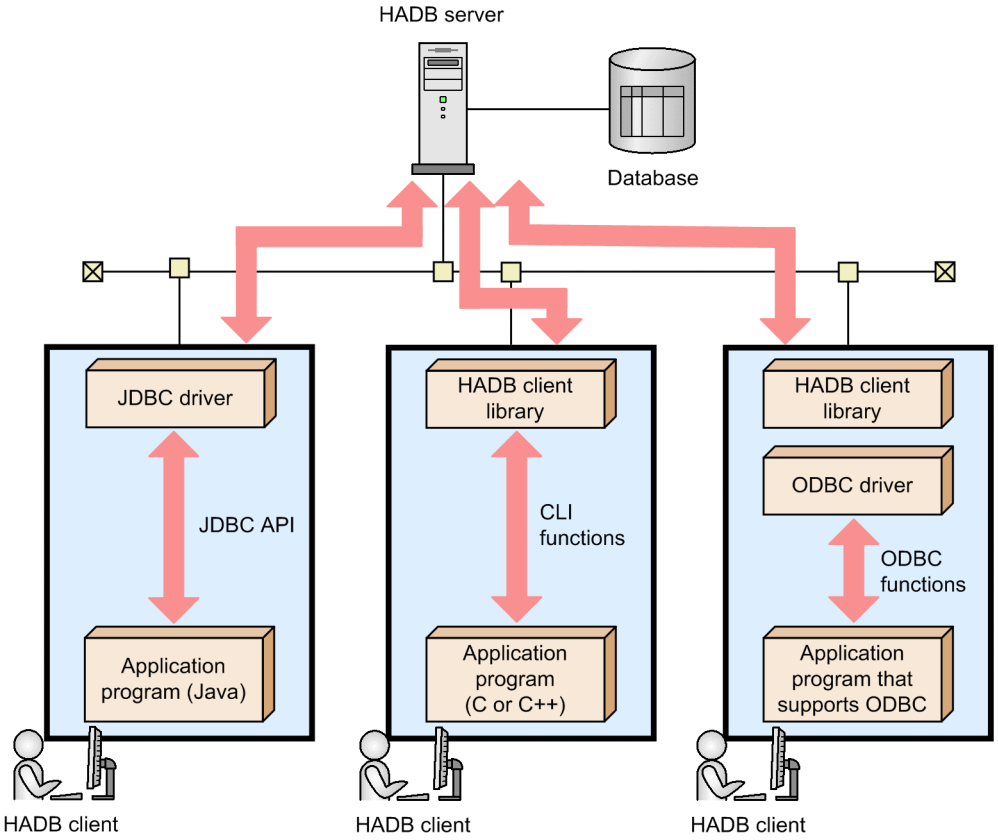
Application programs can be developed on a computer where either HADB server or HADB client is installed.

1.2 Application program execution modes

You can execute application programs on HADB clients and on HADB servers.

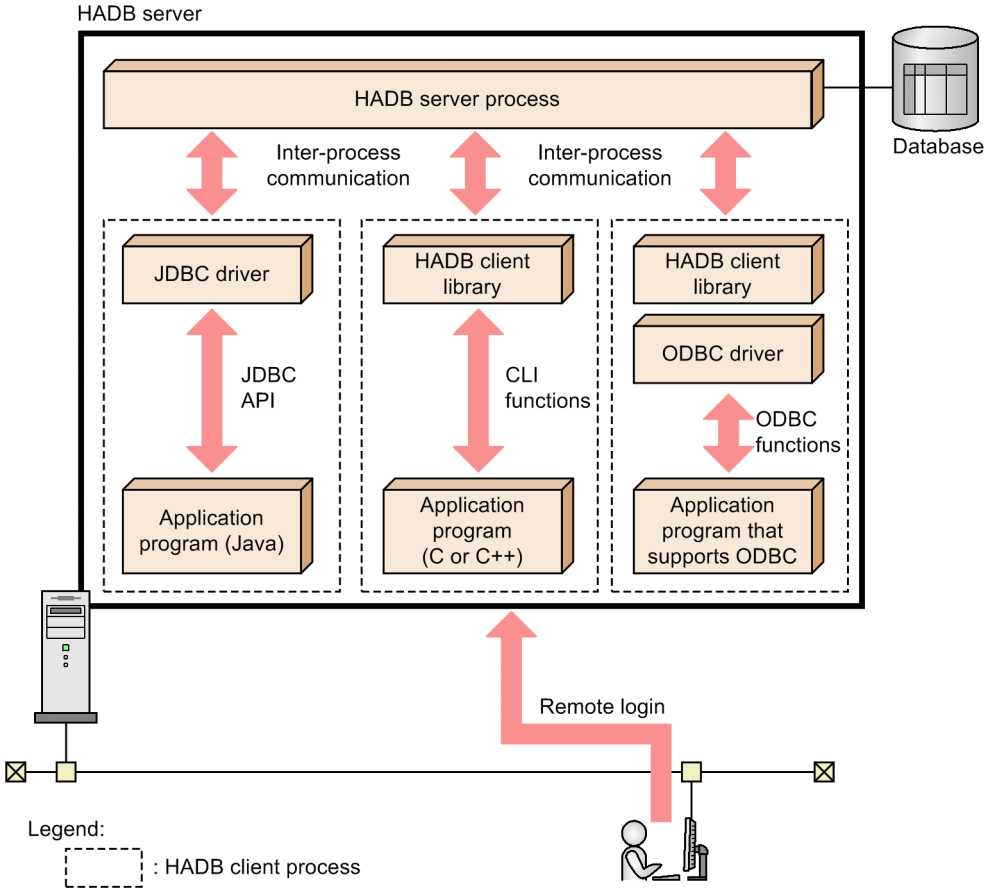
Figure 1-2 shows the mode in which application programs are executed on an HADB client, and Figure 1-3 shows the mode in which application programs are executed on an HADB server.

Figure 1-2: Mode in which application programs are executed on an HADB client



Note: The OS for HADB clients is Linux or Windows.

Figure 1-3: Mode in which application programs are executed on an HADB server



2 Designing Client Definitions

This chapter explains the format in which operands for client definitions are to be specified, the content of client definitions, and the syntax rules that apply to client definitions.

2.1 Specification formats for operands in the client definition

Specify the execution environment of the HADB client in the client definition. The following shows the specification formats for operands in the client definition.

```
Operands related to system configuration
   set adb_clt_rpc_srv_host = HADB-server's-host-name
  [set adb_clt_rpc_srv_port = HADB-server's-port-number]
  [set adb_clt_group_name = client-group-name]

Operands related to application program status monitoring
  [set adb_clt_rpc_con_wait_time = wait-time-until-completion-of-connection-to-HADB-s
erver]
  [set adb_clt_rpc_sql_wait_time = HADB-server's-response-wait-time]
  [set adb_clt_ap_name = application-identifier]

Operands related to performance
  [set adb_clt_fetch_size = number-of-batch-transmission-rows-during-FETCH-processing
]
  [set adb_dbbuff_wrktbl_clt_blk_num = number-of-local-work-table-buffer-pages]
  [set adb_sql_exe_max_rthd_num = maximum-number-of-SQL-processing-real-threads]
  [set adb_sql_exe_hashgrp_area_size = hash-grouping-area-size]
  [set adb_sql_exe_hashtbl_area_size = hash-table-area-size]
  [set adb_sql_exe_hashflt_area_size = hash-filter-area-size]

Operands related to SQL
  [set adb_sql_prep_delrsvd_use_srvdef = {Y|N}]
  [set adb_clt_trn_iso_lv = {READ_COMMITTED|REPEATABLE_READ}]
  [set adb_clt_trn_access_mode = {READ_WRITE|READ_ONLY}]
  [set adb_clt_sql_text_out = {Y|N}]
  [set adb_clt_sql_order_mode = {BYTE|ISO}]
  [set adb_sql_prep_dec_div_rs_prior = {INTEGRAL_PART|FRACTIONAL_PART}]
```

For explanations of operands in the client definition, see 2.2 Contents of operands in the client definition.

For details about how to create and modify the client definition, see 4.4 Creating a client definition.

2.2 Contents of operands in the client definition

This section provides detailed descriptions of the operands in the client definition.

Important

- Operands specified in the client definition are applied when the ODBC driver or CLI functions are used. They are not applied when the JDBC driver is used.
- If you use the JDBC driver, you can use system properties, user properties, or URL connection properties to specify the same information as is specified in the client definition operands.

  For details about system properties, see 3.1.6 Setting system properties.

  For details about the user properties, see (d) Values to be specified in the info argument (specifying the user properties) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

  For details about the URL connection properties, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.
- The operands specified in the client definition are applied when the `adbsql` command is used to execute SQL statements.

2.2.1 Operands related to system configuration

- `adb_clt_rpc_srv_host` = *HADB-server's-host-name*

  ~<character string>((1 to 255 bytes))

  Specify the host name of the HADB server to which the client will be connected. This host name is used for communication between the HADB client and the HADB server.

  Specify the host's host name, domain name, or IP address.

  This operand is mandatory.

  You must specify this operand even when using the function for centrally managing client definitions. However, this operand is ignored when specified in a client definition file that uses the function for centrally managing client definitions.

  In a cold standby configuration, specify the alias IP address used for communication between the HADB server and the HADB client.

  Multi-node function:

  When you use the multi-node function, specify the alias IP address that will be used for communication between the HADB server and the HADB client.

- `adb_clt_rpc_srv_port` = *HADB-server's-port-number*

  ~<integer>((5,001 to 65,535))<<23,650>>

  Specify the port number to be used for communication between the HADB client and the HADB server. Specify the port number specified in the `adb_rpc_port` operand in the server definition.

- `adb_clt_group_name` = *client-group-name*

  ~<character string>((1 to 30 bytes))

Specify the name of the client group to which applications that use this client definition belong. Specify the client group name specified in the adbcltgrp operand in the server definition.

If you omit this operand, applications that use this client definition will not belong to a client group.

If you specify a client group name that is not specified in the adbcltgrp operand, the application will be seen as not belonging to a client group.

For details about client groups, see *Client-group facility* in the *HADB Setup and Operation Guide*.

2.2.2 Operands related to application program status monitoring

- adb_clt_rpc_con_wait_time =*wait-time-until-completion-of-connection-to-HADB-server*

  ~<integer>((1 to 300))<<300>> (seconds)

  Specify the maximum amount of time (in seconds) to wait for HADB server connection processing to be completed. If HADB server connection processing is not completed within the specified amount of time, the connection processing is canceled and control is returned to the application program with an error.

  Normally, there is no need to specify this operand. Specify this operand to reduce the timeout time if it takes a long time to establish a connection when the HADB server is busy.

  ■ When adb_clt_rpc_con_wait_time is specified in the JDBC driver properties

  You can specify 0 in the adb_clt_rpc_con_wait_time operand in the JDBC driver properties. If you specify 0, the default value applies.

- adb_clt_rpc_sql_wait_time =*HADB-server's-response-wait-time*

  ~<integer>((0 to 65,535))<<0>> (seconds)

  Specify the maximum amount of time (in seconds) to wait for a response after a processing request has been issued from the HADB client to the HADB server. If there is no response from the HADB server within the specified time, a timeout error whose SQLCODE is -732 (KFAA30732-E) is returned to the application. When this occurs, processing of the SQL statement is canceled, and the transaction is rolled back. Then, the application is disconnected from the HADB server.

  Note that depending on factors such as the timing with which the processing request was canceled and the nature of the communication error, there are situations in which the transaction might not be rolled back. If this occurs, we recommend that you check the result by viewing relevant information such as the messages output to the message log of the HADB server.

  Specify this operand if you monitor SQL statements that require a long processing time. To determine an appropriate wait time to be specified in this operand, see 4.5 Handling unresponsive application programs.

  If this operand is omitted or 0 is specified, no wait time is set.

  ■ When adb_clt_rpc_sql_wait_time is specified in the JDBC driver properties

  When adb_clt_rpc_sql_wait_time is specified in the JDBC driver properties, HADB also monitors the following wait times:

  - When multiple SELECT statements are executed concurrently in the same connection, the wait time for processing real thread allocation when there are insufficient processing real threads

  If this wait time is exceeded, HADB returns a timeout error whose SQLCODE is -1071570 (KFAA71570-E) to the application. When this happens, processing of the SQL statement is canceled but the transaction is not rolled back. Nor is the application disconnected from the HADB server.

  For details about the purpose of specifying adb_clt_rpc_sql_wait_time, see (4) Note about executing multiple SELECT statements concurrently in the same connection in 7.4.1 How to retrieve data.

- adb_clt_ap_name =*application-identifier*

  ~<character string>((1 to 30 bytes))<<******>>**

Specify the identification information (application identifier) for the application program that is to be connected to the HADB server.

The application identifier you specify is displayed in the output of the command that displays connection statuses (`adbls -d cnct`), in messages, and in SQL trace information. This information is required in order to determine which application program is running.

Because application identifiers are recognized on the basis of the specification of the `ADBCLTLANG` environment variable, we recommend that you use a name consisting of only alphanumeric characters that do not depend on the character encoding.

2.2.3 Operands related to performance

- `adb_clt_fetch_size` = *number-of-batch-transmission-rows-during-FETCH-processing*

  ~<integer>((1 to 65,535))<<1,024>>

  Specify the maximum number of rows that are to be sent as retrieval results from the HADB server to the HADB client by a single `FETCH` process.

  If you specify a large value in this operand, an improvement in performance can be expected because more result rows are sent per `FETCH` process, but more memory is required.

- `adb_dbbuff_wrktbl_clt_blk_num` = *number-of-local-work-table-buffer-pages*

  ~<integer>((5 to 100,000,000))<<value of `adb_dbbuff_wrktbl_clt_blk_num` in the server definition>>

  Specify the number of local work table buffer pages.

  Normally, you do not need to specify this operand. Specify this operand to reduce the execution time of SQL statements that create local work tables. For details, see *Tuning to shorten SQL statement execution time by re-examining the buffers* in *Tuning* in the *HADB Setup and Operation Guide*.

  For details about how to estimate an appropriate value for this operand, see *Estimating the number of pages in the buffer for local work tables* in the *HADB Setup and Operation Guide*.

  The following notes apply to this operand:

  - If this operand and the `adb_dbbuff_wrktbl_clt_blk_num` operand in the server definition are both specified, this operand value takes effect.
  - The buffer specified in this operand is used for an SQL processing real thread only when SQL statements for creating local work tables are executed. Local work tables are specific to real threads, and one local work table is created for each real thread. Therefore, the HADB server allocates for each real thread the amount of local work table buffer space as matches the number of pages specified in this operand. For details about the SQL statements for creating local work tables, see 5.10.2 Work tables created when SQL statements are executed.
  - The `adbmodbuff` command cannot be used to change the number of local work table buffer pages for a connection to which this operand is applied.
  - You can use the `adbls -d lbuf` command to check the number of local work table buffer pages that are applied for each connection.

- `adb_sql_exe_max_rthd_num` =*maximum-number-of-SQL-processing-real-threads*

  ~<integer>((0 to 4,096)) <<value of `adb_sql_exe_max_rthd_num` in the server definition or maximum number of processing real threads usable by the group>>

  Specify the maximum number of processing real threads that are to be used during SQL statement execution.

  Specify this operand if you are changing the maximum number of processing real threads for SQL statement execution that was specified in the server definition's `adb_sql_exe_max_rthd_num` operand.

When you specify this operand, see the explanation of the adb_sql_exe_max_rthd_num operand under the topic *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*.

■ **When not using the client-group facility**

- If you omit this operand, the value specified for the adb_sql_exe_max_rthd_num operand in the server definition is assumed.
- If the value you specify in this operand is greater than the value specified in the adb_sql_exe_max_rthd_num operand in the server definition, the value specified in this operand is ignored. In this case, the value specified for the adb_sql_exe_max_rthd_num operand in the server definition is assumed.

■ **When using the client-group facility**

- If you omit this operand, the maximum number of processing real threads usable by the group is assumed.
- If the value you specify in this operand is greater than the maximum number of processing real threads usable by the group, the value specified in this operand is ignored. In this case, the maximum number of processing real threads usable by the group is assumed.

For details about the client-group facility, see *Client-group facility* in the *HADB Setup and Operation Guide*.

■ **Relationship between this operand and the setHADBSQLMaxRthdNum method of the JDBC driver**

The setHADBSQLMaxRthdNum method can be used to specify the maximum number of SQL processing real threads.

The following table shows which of the values is applied to this operand according to the relationships among them and depending on whether the setHADBSQLMaxRthdNum method is specified.

For details about the setHADBSQLMaxRthdNum method, see 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

Whether the setHADBSQLMaxRthdNum method is specified	Relationship between SV and CGV	Relationships among V, SV, and CGV	Value applied to this operand
Specified	$CGV < SV$	$SV < V$	CGV[#1]
		$CGV < V \leq SV$	
		$V \leq CGV$	V
	$SV \leq CGV$	$V \leq SV$	V
		$SV < V \leq CGV$	SV[#1]
		$CGV < V$	
	CGV not specified[#2]	$V \leq SV$	V
		$SV < V$	SV[#1]
Not specified	--	--	CNV

Legend:

V: Value specified for the setHADBSQLMaxRthdNum method

SV: Value in the server definition[#3]

CGV: Maximum number of processing real threads usable by a group if the client-group facility is used

CNV: Maximum number of SQL processing real threads determined when a connection to the HADB server is established[#4]

--: No condition

#1:

Disables the value specified for the setHADBSQLMaxRthdNum method. At this time, the KFAA41106-W message is output to indicate that the specified value was disabled.

#2:

This is the case where no client-group facility is used.

#3:

The maximum number of SQL processing real threads determined when the HADB server starts. This value is determined by conditions, such as whether the adb_sql_exe_max_rthd_num operand is specified in the server definition, and the magnitude relationship with the number of processing real threads. For details, see the explanation of the adb_sql_exe_max_rthd_num operand in *Operands related to performance (set format)* in the *HADB Setup and Operation Guide*.

#4:

This value is determined by the following conditions: 1) whether the adb_sql_exe_max_rthd_num operand is specified in the client definition, 2) the value specified for the adb_sql_exe_max_rthd_num operand in the server definition, and 3) the magnitude relationship with the maximum number of processing real threads usable by a group of the client-group facility.

Important

The HADB server references the value of this operand when performing the preprocessing of an SQL statement. Therefore, if you use the setHADBSQLMaxRthdNum method to specify the maximum number of SQL processing real threads, make sure that the setHADBSQLMaxRthdNum method is run before the Statement or PreparedStatement object that executes the SQL statement is generated.

- adb_sql_exe_hashgrp_area_size = *hash-grouping-area-size*

  ~<integer>((0, 4 to 1,000,000))<<value of adb_sql_exe_hashgrp_area_size in the server definition>>(kilobytes)

  Specify a size (in kilobytes) for the hash grouping area.

  Specify this operand if you are changing the hash grouping area size that was specified in the server definition's adb_sql_exe_hashgrp_area_size operand.

  When you specify this operand, see the explanation of the adb_sql_exe_hashgrp_area_size operand under the topic *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*.

- adb_sql_exe_hashtbl_area_size = *hash-table-area-size*

  ~<integer>((0 to 4,194,304))<<value of adb_sql_exe_hashtbl_area_size in the server definition>>(megabytes)

  Specify the size (in megabytes) of the hash table area.

  Use this operand to change the size of the hash table area specified in the adb_sql_exe_hashtbl_area_size operand in the server definition.

  When you specify this operand, see the explanation of the adb_sql_exe_hashtbl_area_size operand under the topic *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*.

  If this operand's value is greater than the value of the adb_sql_exe_hashtbl_area_size operand in the server definition, the specification in the server definition is assumed. The specification in the server definition is assumed also when this operand is omitted.

■ **Relationship between this operand and the setHADBSQLHashTblSize method of the JDBC driver**

The `setHADBSQLHashTblSize` method can be used to specify the size of the hash table area.

The following table shows which of the values is applied to this operand according to the relationships among them and depending on whether the `setHADBSQLHashTblSize` method is specified.

For details about the `setHADBSQLHashTblSize` method, see 8.2.36 setHADBSQLHashTblSize(int areaSize).

Whether the `setHADBSQLHashTblSize` method is specified	Relationship between V and SV	Value applied to this operand (in megabytes)
Specified	*SV* < *V*	*SV*[#1]
	V ≤ *SV*	*V*
Not specified	--	*CNV*

Legend:

V: Value specified for the `setHADBSQLHashTblSize` method

SV: Value in the server definition[#2]

CNV: Size of the hash table area determined when a connection to the HADB server is established[#3]

--: No condition

#1:

Disables the value specified for the `setHADBSQLHashTblSize` method. At this time, the `KFAA41106-W` message is output to indicate that the specified value was disabled.

#2:

The size of the hash table area determined when a connection to the HADB server is established. This is the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition. If the `adb_sql_exe_hashtbl_area_size` operand in the server definition is omitted, the default value is applied.

#3:

This value is determined by conditions such as: whether the `adb_sql_exe_hashtbl_area_size` operand is specified in the client definition, and the magnitude relationship between the values specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition and the `adb_sql_exe_hashtbl_area_size` operand in the client definition.

> **Important**
>
> The HADB server references the value of this operand when performing the preprocessing of an SQL statement. Therefore, if you use the `setHADBSQLHashTblSize` method to specify the size of the hash table area, make sure that the `setHADBSQLHashTblSize` method is run before the `Statement` or `PreparedStatement` object that executes the SQL statement is generated.

- `adb_sql_exe_hashflt_area_size =` *hash-filter-area-size*

  ~<integer>((0 to 419,430)) (megabytes)

  Specify the size (in megabytes) of the hash filter area.

  Normally, you will omit this operand. Specify this operand to reduce the execution time of SQL statements to which a hash filter is applied.

  When you specify this operand, see the explanation of the `adb_sql_exe_hashflt_area_size` operand in *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*.

For details about how to tune the value to be specified for this operand, see *Tuning to shorten SQL statement execution time by re-examining the hash filter area size* in the *HADB Setup and Operation Guide*.

The following rules apply to this operand:

1. The value that is assumed when this operand is omitted varies depending on whether the `adb_sql_exe_hashtbl_area_size` operand is specified in the client definition.
   - If the `adb_sql_exe_hashtbl_area_size` operand is specified:
     ↑ Value of the `adb_sql_exe_hashtbl_area_size` operand in the client definition ÷ 10 ↑
   - If the `adb_sql_exe_hashtbl_area_size` operand is not specified:
     Value of the `adb_sql_exe_hashflt_area_size` operand in the server definition

   Note that if the value of the `adb_sql_exe_hashtbl_area_size` operand in the client definition is greater than the value of this operand in the server definition, the value of the server definition is assumed.
2. If the following condition is satisfied, the value of the `adb_sql_exe_hashflt_area_size` operand in the server definition (not in the client definition) is applied:

   ```
   A < B or C
   ```

   A: Value specified for the `adb_sql_exe_hashflt_area_size` operand in the server definition

   B: Value specified for the `adb_sql_exe_hashflt_area_size` operand in the client definition

   C: Value that is assumed if the `adb_sql_exe_hashflt_area_size` operand is omitted in the client definition

■ Relationship between this operand and the setHADBSQLHashFltSize method of the JDBC driver

The `setHADBSQLHashFltSize` method can be used to specify the size of the hash filter area. The following table shows which of the values is applied to this operand according to the relationships among them and depending on whether the `setHADBSQLHashFltSize` method is specified.

For details about the `setHADBSQLHashFltSize` method, see 8.2.35 setHADBSQLHashFltSize(int areaSize).

Whether the `setHADBSQLHashFltSize` method is specified	Relationship between V and SV	Relationship between HTV and SV	Value applied to this operand (in megabytes)
Specified	*SV* < *V*	--	*SV*[#1]
	V ≤ *SV*	--	*V*
Not specified	--	*SV* < ↑*HTV* ÷ 10↑	*SV*
		↑*HTV* ÷ 10↑ ≤ *SV*	↑*HTV* ÷ 10↑
		HTV not specified[#2]	*CNV*

Legend:

V: Value specified for the `setHADBSQLHashFltSize` method

SV: Value in the server definition[#3]

HTV: Size of the hash table area[#4]

CNV: Size of the hash filter area determined when a connection to the HADB server is established[#5]

--: No condition

#1:

Disables the value specified for the `setHADBSQLHashFltSize` method. At this time, the `KFAA41106-W` message is output to indicate that the specified value was disabled.

#2:

This is the case where the `setHADBSQLHashTblSize` method is not specified.

#3:

The size of the hash filter area determined when a connection to the HADB server is established. This value is determined by conditions, such as whether the `adb_sql_exe_hashflt_area_size` operand is specified in the server definition, and the magnitude relationship with the number of processing real threads. For details, see the explanation of the `adb_sql_exe_hashflt_area_size` operand in *Operands related to performance (set format)* in the *HADB Setup and Operation Guide*.

#4:

The size of the hash table area that was last applied if the `setHADBSQLHashTblSize` method is specified.

#5:

This value is determined by conditions such as: whether the `adb_sql_exe_hashflt_area_size` operand is specified in the client definition, and the magnitude relationship between the values specified for the `adb_sql_exe_hashflt_area_size` operand in the server definition and the `adb_sql_exe_hashflt_area_size` operand in the client definition.

Important

The HADB server references the value of this operand when performing the preprocessing of an SQL statement. Therefore, if you use the `setHADBSQLHashFltSize` method to specify the size of the hash filter area, make sure that the `setHADBSQLHashFltSize` method is run before the `Statement` or `PreparedStatement` object that executes the SQL statement is generated.

2.2.4 Operands related to SQL

- `adb_sql_prep_delrsvd_use_srvdef ={Y|N}`

  Specify whether reserved words are to be unregistered if specified as such in the `adb_sql_prep_delrsvd_words` operand in the server definition.

  Check the `adb_sql_prep_delrsvd_words` operand in the server definition for the reserved words that are unregistered:

  `Y`

  Enables the `adb_sql_prep_delrsvd_words` operand in the server definition (reserved words specified in the `adb_sql_prep_delrsvd_words` operand are to be unregistered).

  `N`

  Disables the `adb_sql_prep_delrsvd_words` operand in the server definition (reserved words specified in the `adb_sql_prep_delrsvd_words` operand are not to be unregistered).

  If specification of this operand is omitted, `Y` is assumed.

- `adb_clt_trn_iso_lv = {READ_COMMITTED|REPEATABLE_READ}`

  Specify the transaction isolation level that is to be applied. For details about the transaction isolation levels, see *Transaction isolation levels supported by HADB* in the *HADB Setup and Operation Guide*.

  `READ_COMMITTED`

  Applies `READ COMMITTED` as the transaction isolation level.

REPEATABLE_READ

Applies REPEATABLE READ as the transaction isolation level.

If this operand is omitted, the transaction isolation level specified in the adb_sys_trn_iso_lv operand in the server definition is applied.

Multi-node function

- When both of the following conditions are met, transaction execution processing is also allocated to the slave node:
  - The transaction access mode is read-only mode.
  - The transaction isolation level is READ COMMITTED.

  We recommend that you specify READ_COMMITTED in this operand if you want to utilize the resources of the slave node.
- When transactions that use read/write mode and certain commands are running on the master node, all transactions run on the master node for the duration of that transaction or command. No transactions run on the slave node during this time. For details, see *Nodes on which transactions and commands are executed* in the *HADB Setup and Operation Guide*.

  For details about the commands to which this restriction applies, see *Restrictions on simultaneously executing commands with transactions* in *Nodes on which transactions and commands are executed* in the *HADB Setup and Operation Guide*.

- adb_clt_trn_access_mode = {READ_WRITE|READ_ONLY)

  Specify the transaction access mode. For details about the transaction access mode, see the topic *Transaction access modes* in the *HADB Setup and Operation Guide*.

  READ_WRITE

  Applies read/write as the transaction access mode. In this case, a transaction becomes a read/write transaction and can execute all SQL statements.

  READ_ONLY

  Applies read-only as the transaction access mode. In this case, a transaction becomes a read-only transaction and cannot execute update SQL statements or definition SQL statements.

  You can change the specified transaction access mode by setting the following connection attributes:

  - ODBC functions: SQLSetConnectAttr or SQLSetConnectAttrW
  - CLI functions: a_rdb_SQLSetConnectAttr()

  The transaction access mode is set to READ_WRITE when it is not specified in this operand or with the connection attribute.

  Multi-node function:

  - When both of the following conditions are met, transaction execution processing is also allocated to the slave node:
    - The transaction access mode is read-only mode.
    - The transaction isolation level is READ COMMITTED.

    We recommend that you specify READ_ONLY in this operand if you want to utilize the resources of the slave node.
  - When transactions that use read/write mode and certain commands are running on the master node, all transactions run on the master node for the duration of that transaction or command. No transactions run on the slave node during this time. For details, see *Nodes on which transactions and commands are executed* in the *HADB Setup and Operation Guide*.

For details about the commands to which this restriction applies, see *Restrictions on simultaneously executing commands with transactions* in *Nodes on which transactions and commands are executed* in the *HADB Setup and Operation Guide*.

- `adb_clt_sql_text_out ={Y|N}`

  Specify whether the SQL statements issued by the HADB client are to be output to the client message log files and the server message log files.

  The maximum length of each SQL statement that is output is 2,048 bytes.

  `Y`

  Outputs the SQL statements to the client message log files and the server message log files.

  `N`

  Does not output the SQL statements to either the client message log files or the server message log files.

  The following table shows the relationship between this operand and the `adb_sql_text_out` operand in the server definition.

  Table 2-1: Relationship between this operand and the adb_sql_text_out operand in the server definition

adb_sql_text_out operand in the server definition	adb_clt_sql_text_out operand in the client definition	
	Y	N
Y	B	S
N	B	N

  Legend:

  B: Outputs the SQL statements to both the client message log files and server message log files.

  S: Outputs the SQL statements to the server message log files only.

  N: Does not output the SQL statements to either the client message log files or the server message log files.

  When this operand is specified, the `KFAA81002-I` message indicating normal termination of a transaction is output to the server message log files. This message is not output to the client message log files.

  If specification of this operand is omitted, `N` is assumed.

- `adb_clt_sql_order_mode= {BYTE|ISO}`

  Specify the sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified.

  `BYTE`

  Sort character string data by bytecode.

  `ISO`

  Sort character string data by sort code (ISO/IEC 14651:2011 compliance).

  `ISO` cannot be specified in this operand when `SJIS` is specified in the `ADBCLTLANG` environment variable.

  You can also use the `adb_sql_order_mode` server definition operand and the connection attributes to specify the sort order for character string data. If this sort order is specified by more than one of these methods, the specification to be used is determined in the priority shown below (the smaller the numeric value, the higher the priority).

  Table 2-2: Priority for the sort order of character string data

Priority	Location of specification of the character string data sort order
1	Connection attributes
2	`adb_clt_sql_order_mode` client definition operand

Priority	Location of specification of the character string data sort order
3	adb_sql_order_mode server definition operand

Explanation:

For example, if BYTE is specified in the adb_clt_sql_order_mode client definition operand and ISO is specified in the adb_sql_order_mode server definition operand, BYTE is applied to SELECT statements (in which the ORDER BY clause is specified) that are executed from the application program.

BYTE is assumed when the specification is omitted at all locations shown in the table.

- adb_sql_prep_dec_div_rs_prior = {INTEGRAL_PART|FRACTIONAL_PART}

Specify the minimum scaling value of the result of a division operation (arithmetic operation) specified in an SQL statement when the data type of the result is DECIMAL.

INTEGRAL_PART

The minimum scaling value of the result of he division operation is 0. If you specify INTEGRAL_PART, the number of digits in the integer part has priority. Specify INTEGRAL_PART when the division result might be a large value and you want to avoid overflow errors where possible.

FRACTIONAL_PART

The scaling of the first operand (dividend) of the division operation is used as the minimum scaling value of the division result. When you specify FRACTIONAL_PART, the number of decimal places in the first operand is the minimum number of decimal places in the division result.

When the first operand is DECIMAL(*p1*,*s1*) and the second operand is DECIMAL(*p2*,*s2*), the scaling of the division result is DECIMAL(38,*s*).

- When INTEGRAL_PART is specified

  $s = \mathrm{MAX}\{0, 38 - (p1 - s1 + s2)\}$

- When FRACTIONAL_PART is specified

  $s = \mathrm{MAX}\{s1, 38 - (p1 - s1 + s2)\}$

The following example shows how value specified for this operand affects the division result.

Example:

Table T1

Column C1	Column C2
DECIMAL(38,4)	DECIMAL(10,5)
30.5256	0.05223

Suppose that the following SELECT statement is executed:

```
SELECT "C1"/"C2" AS "division result" FROM "T1"
```

- Division result when INTEGRAL_PART is specified

  584.

- Division result when FRACTIONAL_PART is specified

  584.4457

For an example of the scaling of the division result when retrieving data from a viewed table, see *Notes applying when the data type of the division result is DECIMAL* in the manual *HADB SQL Reference*.

If you omit this operand, the value specified for the adb_sql_prep_dec_div_rs_prior operand in the server definition applies.

2.3 Operand specification rules

The specification rules for operands in the client definition are the same as those for server definition operands. For details about the syntax rules for the server definition operands, see the topic *Syntax rules for the server definition* in the *HADB Setup and Operation Guide*.

2.4 Notes about using the function for centrally managing client definitions

The function for centrally managing client definitions does not support the following client definition operands. Specify these operands directly in the client definition of each HADB client.

- `adb_clt_rpc_srv_host`
- `adb_clt_rpc_srv_port`
- `adb_clt_rpc_con_wait_time`
- `adb_clt_rpc_sql_wait_time`

For details about the function for centrally managing client definitions, see *Centralized management of client definitions* in the *HADB Setup and Operation Guide*.

3

Setting Up an Environment for the JDBC Driver

This chapter explains how to set up an environment for the JDBC driver, including installation of the JDBC driver and specification of environment variables.

3.1 Environment setup procedure for the JDBC driver

This section describes how to set up an environment for the JDBC driver. Set up an environment for the JDBC driver, following the description in 3.1.1 Installing Java Runtime Environment or Java Development Kit and the subsequent subsections.

3.1.1 Installing Java Runtime Environment or Java Development Kit

Install either of the following products on a machine on which application programs are to be executed or developed:

- Java Runtime Environment (JRE) version 8 or later
- Java Development Kit (JDK) version 8 or later

JRE is required to execute application programs. JDK is required to develop and execute application programs.

> **Note**
>
> To execute application programs on an HADB server, install JRE on the HADB server machine. If you use the HADB server to develop and run applications, JDK must be installed on the computer where the HADB server is installed.

■ JDBC driver provided by HADB

The following table shows the JDBC driver provided by HADB.

Table 3-1: JDBC driver provided by HADB

JRE or JDK version	Types of JDBC drivers provided by HADB	JAR file name	Corresponding JDBC standard
JRE 8 or later, or JDK 8 or later	JDBC driver for JRE 8	`adbjdbc8.jar`	JDBC 4.2

3.1.2 Installing the JDBC driver

The following shows the procedure for installing the JDBC driver.

Procedure

1. Copy a compressed file from the HADB client installation CD-ROM to a folder of your choice.
   The file to be copied is as follows:
   - In the 64-bit edition of Windows
     `hitachi_advanced_data_binder_client.zip`
   - In the 32-bit edition of Windows
     `hitachi_advanced_data_binder_client32.zip`
   - In Linux
     `hitachi_advanced_data_binder_client-$VER.tar.gz`
     The `$VER` portion is replaced by an HADB version and release number.
2. Expand the compressed file that you copied.

The following table describes the location in which the JAR file will be stored when the compressed file is expanded.

Table 3-2: Location of the JAR file after the compressed file is expanded

Compressed file to be expanded	Location where the JAR file will be stored
Either of the following compressed files: • `hitachi_advanced_data_binder_client.zip` • `hitachi_advanced_data_binder_client32.zip`	`%INSTCLTDIR%\HADBCL\client\lib\adbjdbc8.jar` `%INSTCLTDIR%` indicates the folder in which the compressed file is expanded.
`hitachi_advanced_data_binder_client-$VER.tar.gz` is expanded	`$INSTCLTDIR/hitachi_advanced_data_binder_client_$VER/client/lib/adbjdbc8.jar` `$INSTCLTDIR` indicates the directory in which the compressed file is expanded.

3. Copy the JAR file to a folder of your choice.
4. Delete the following folders and files:
   - The compressed file you copied in step 1.
   - The folders and files you expanded in step 2.

3.1.3 Specifying the CLASSPATH environment variable

For `CLASSPATH`, specify the absolute path of the JAR file.

Important

- If the OS is Windows, set `CLASSPATH` as a system environment variable.
- If you change the location of the JAR file, also change the specification of `CLASSPATH` accordingly.

3.1.4 Checking the value of the TZ environment variable

Confirm that the correct time zone is set for the `TZ` environment variable.

Important

- Do not specify a time zone that uses leap seconds.
- If the OS is Windows, set `TZ` as a system environment variable.

3.1.5 Granting the write permission for the trace file output destination directory

Be careful when the JDBC interface method trace or Exception trace log, which is a troubleshooting function for the JDBC driver, is to be used. In this case, give the write permission for the trace file output destination folder (directory) to the user who uses the JDBC driver.

For details about these troubleshooting functions, see 7.7 Troubleshooting.

3.1.6 Setting system properties

Use system properties to set up an execution environment for application programs. The following table lists the properties that can be specified as system properties.

Important

- In the following table, properties from 1 to 18 are functionally the same as the operands in the client definition.
- Property 19 and subsequent properties specify the settings related to the Exception trace log.

Table 3-3: System properties that can be specified

No.	Property name	Description
1	adb_clt_rpc_srv_host	Specifies the host name of the HADB server at the connection destination. Functionally, this property is the same as the adb_clt_rpc_srv_host operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_srv_host operand in the client definition.
2	adb_clt_rpc_srv_port	Specifies the port number of the HADB server that is used for communication between the HADB client and the HADB server. Functionally, this property is the same as the adb_clt_rpc_srv_port operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_srv_port operand in the client definition.
3	adb_clt_rpc_con_wait_time	Specifies the maximum amount of time to wait for HADB server connection processing to be completed. Functionally, this property is the same as the adb_clt_rpc_con_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_con_wait_time operand in the client definition.
4	adb_clt_rpc_sql_wait_time	Specifies the following wait times: • How long a HADB client waits for the HADB server to respond to a processing request. • How long to wait to secure processing real threads if a shortage occurs when multiple SELECT statements are executed concurrently in the same connection. For details about the wait time monitoring performed if this property is specified, see 3.2 Handling unresponsive application programs. Functionally, this property is the same as the adb_clt_rpc_sql_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_sql_wait_time operand in the client definition.

No.	Property name	Description
5	adb_clt_ap_name	Specifies the identification information (application identifier) for the application program that is to be connected to the HADB server. Functionally, this property is the same as the adb_clt_ap_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_ap_name operand in the client definition.
6	adb_clt_group_name	Specifies the name of the client group to which the application belongs. Functionally, this property is the same as the adb_clt_group_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_group_name operand in the client definition.
7	adb_clt_fetch_size	Specifies the maximum number of rows that are to be sent as retrieval results from the HADB server to the HADB client by a single FETCH process. Functionally, this property is the same as the adb_clt_fetch_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_fetch_size operand in the client definition.
8	adb_dbbuff_wrktbl_clt_blk_num	Specifies the number of local work table buffer pages. Functionally, this property is the same as the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition.
9	adb_sql_exe_max_rthd_num	Specifies the maximum number of SQL processing real threads. Functionally, this property is the same as the adb_sql_exe_max_rthd_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_max_rthd_num operand in the client definition.
10	adb_sql_exe_hashgrp_area_size	Specifies the size (in kilobytes) of the hash grouping area. Functionally, this property is the same as the adb_sql_exe_hashgrp_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashgrp_area_size operand in the client definition.
11	adb_sql_exe_hashtbl_area_size	Specifies the size (in megabytes) of the hash table area. Functionally, this property is the same as the adb_sql_exe_hashtbl_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashtbl_area_size operand in the client definition.
12	adb_sql_exe_hashflt_area_size	Specifies the size (in megabytes) of the hash filter area. Functionally, this property is the same as the adb_sql_exe_hashflt_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashflt_area_size operand in the client definition.
13	adb_sql_prep_delrsvd_use_srvdef	Specifies whether reserved words are to be unregistered if specified as such in the adb_sql_prep_delrsvd_words operand in the server definition. Functionally, this property is the same as the adb_sql_prep_delrsvd_use_srvdef operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_prep_delrsvd_use_srvdef operand in the client definition.
14	adb_clt_trn_iso_lv	Specifies the transaction isolation level. Functionally, this property is the same as the adb_clt_trn_iso_lv operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_iso_lv operand in the client definition.
15	adb_clt_trn_access_mode	Specifies the transaction access mode.

No.	Property name	Description	
		Functionally, this property is the same as the adb_clt_trn_access_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_access_mode operand in the client definition.	
16	adb_clt_sql_text_out	Specifies whether SQL statements issued by the HADB client are to be output to the client message log files and the server message log files. Functionally, this property is the same as the adb_clt_sql_text_out operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_text_out operand in the client definition.	
17	adb_clt_sql_order_mode	Specifies the sort order for character string data in a SELECT statement in which the ORDER BY clause is specified. Functionally, this property is the same as the adb_clt_sql_order_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_order_mode operand in the client definition.	
18	adb_sql_prep_dec_div_rs_prior	Specifies the minimum scaling value of the result of a division operation (arithmetic operation) specified in an SQL statement when the data type of the result is DECIMAL. Functionally, this property is the same as the adb_sql_prep_dec_div_rs_prior operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_prep_dec_div_rs_prior operand in the client definition.	
19	adb_jdbc_exc_trc_out_path	Specifies the absolute path of the directory to which exception trace logs are to be output.	For details about these properties and their permitted values, see (b) Setup for acquisition of the exception trace log (setting properties) in (1) Methods to be acquired and setup for log acquisition in 7.7.2 Exception trace log.
20	adb_jdbc_info_max	Specifies the maximum number of information items to be output to one file.	
21	adb_jdbc_cache_info_max	Specifies the maximum number of information items to be stored in memory.	
22	adb_jdbc_trc_out_lv	Specifies the trace acquisition level.	

Important

The values of the properties listed in the preceding table can also be specified by using user properties or the properties of the URL to be used for connection.

For details about the user properties, see (d) Values to be specified in the info argument (specifying the user properties) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

For details about the properties of the URL to be used for connection, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

For details about the priority of each specification, see 7.3.3 Connection information priorities.

Note

The property names of system properties were changed in HADB 03-00, as shown below. The previous property names are still supported, but if you have upgraded your HADB to version 03-00 or later, we recommend that you change the property names.

No.	Property name before change (property name used in HADB versions earlier than 03-00)	Property name after change (property name used in HADB version 03-00 or later)
1	adb_jdbc_ap_name	adb_clt_ap_name
2	adb_jdbc_dbbuff_wrktbl_blk_num	adb_dbbuff_wrktbl_clt_blk_num
3	adb_jdbc_fetch_size	adb_clt_fetch_size
4	adb_jdbc_rpc_sql_wait_time	adb_clt_rpc_sql_wait_time
5	adb_jdbc_rpc_srv_host	adb_clt_rpc_srv_host
6	adb_jdbc_rpc_srv_port	adb_clt_rpc_srv_port
7	adb_jdbc_sql_delrsvd_use_srvdef	adb_sql_prep_delrsvd_use_srvdef
8	adb_jdbc_sql_hashgrp_area_size	adb_sql_exe_hashgrp_area_size
9	adb_jdbc_sql_hashtbl_area_size	adb_sql_exe_hashtbl_area_size
10	adb_jdbc_sql_max_rthd_num	adb_sql_exe_max_rthd_num
11	adb_jdbc_sql_order_mode	adb_clt_sql_order_mode
12	adb_jdbc_sql_text_out	adb_sql_text_out
13	adb_jdbc_trn_access_mode	adb_clt_trn_access_mode
14	adb_jdbc_trn_iso_lv	adb_clt_trn_iso_lv

3.1.7 Reviewing the scope of scans by antivirus software

If antivirus software has been installed on the machine on which the JDBC driver is installed, review the scope of virus scans. If the files and directories used by the JDBC driver are included in the scope of scans by antivirus software, the JDBC driver might not work correctly. For this reason, you need to configure the antivirus software to not scan the directories and files used by the JDBC driver.

3.2 Handling unresponsive application programs

A problem such as a communication error, temporary failure (including power outage), or disk failure might cause an application program to stop responding. Such an unresponsive application program might delay other application programs and command processing.

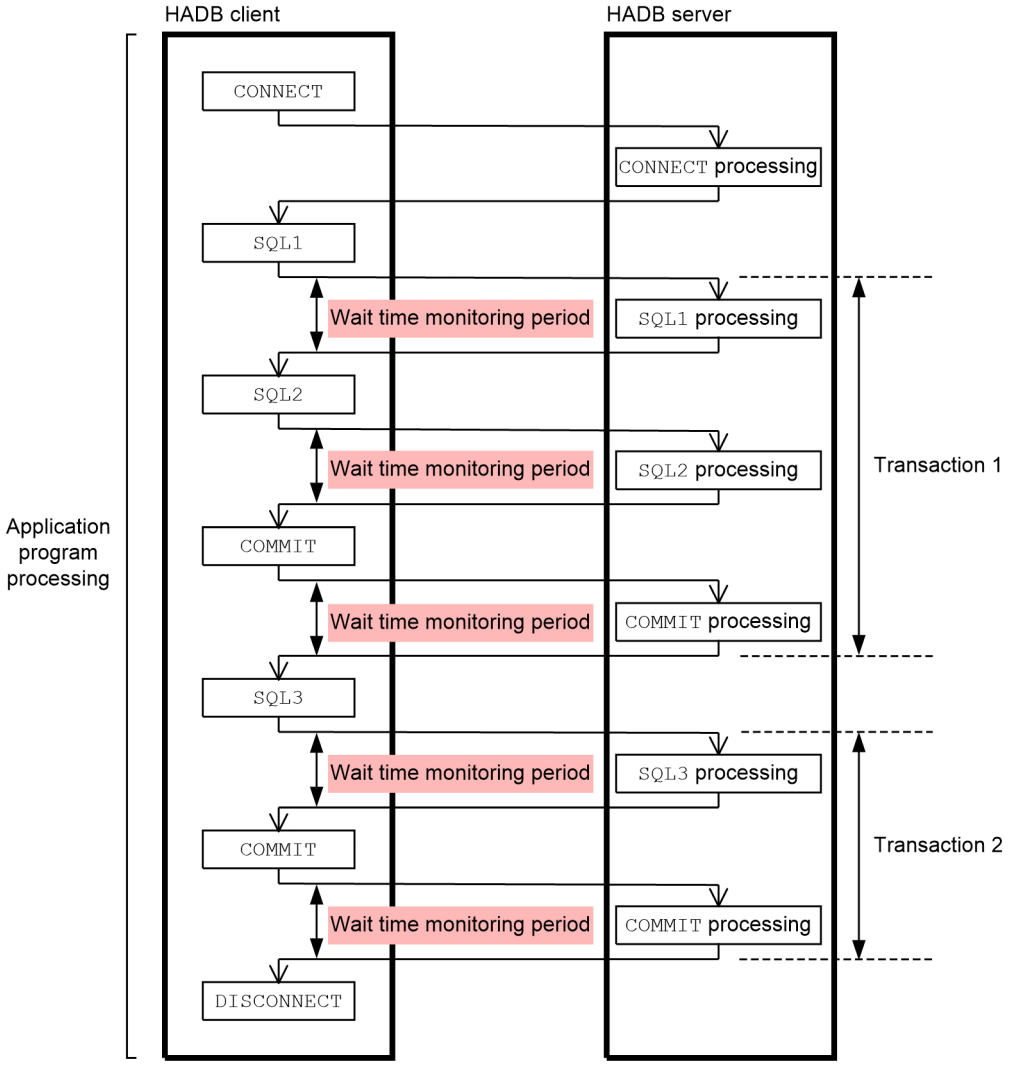
Therefore, in order to minimize the effects of an unresponsive application program, specify the following property as a system property, user property, or connection URL property:

- `adb_clt_rpc_sql_wait_time`

For this property, specify the maximum amount of time to wait for a response after a processing request has been sent from the client (machine on which the JDBC driver is installed) to the HADB server. If there is no response from the HADB server within the specified time, a timeout error whose `SQLCODE` is `-732` (`KFAA30732-E`) is returned to the application. When this occurs, processing of the SQL statement is canceled, and the transaction is rolled back. Then, the application is disconnected from the HADB server.

The following figure shows the wait time monitoring procedure based on the specification of `adb_clt_rpc_sql_wait_time`.

Figure 3-1: Wait time monitoring procedure based on the specification of adb_clt_rpc_sql_wait_time



Explanation:

This example monitors the wait time until a response is received after a processing request has been issued from the client to the HADB server. For example, if 600 seconds is specified in `adb_clt_rpc_sql_wait_time`, a wait time of 600 seconds is set for the monitoring interval. Therefore, as a guideline, specify a wait time at least equal to the longest SQL statement processing time. Specify a realistic value that most likely will indicate when the application program has become unresponsive.

If `adb_clt_rpc_sql_wait_time` is specified, HADB also monitors the following wait times:

- How long to wait to secure processing real threads if a shortage occurs when multiple `SELECT` statements are executed concurrently in the same connection.

If this wait time is exceeded, HADB returns a timeout error whose `SQLCODE` is `-1071570` (`KFAA71570-E`) to the application. When this happens, processing of the SQL statement is canceled but the transaction is not rolled back. Nor is the application disconnected from the HADB server.

For details about the purpose of specifying `adb_clt_rpc_sql_wait_time`, see (4) Note about executing multiple SELECT statements concurrently in the same connection in 7.4.1 How to retrieve data.

Note

For details about `adb_clt_rpc_sql_wait_time`, see 3.1.6 Setting system properties.

3.3 Upgrading the JDBC driver (replacing the JAR file)

The following shows the procedure for upgrading the JDBC driver:

Procedure

1. Copy a compressed file from the HADB client installation CD-ROM to a folder of your choice.
   The file to be copied is as follows:
   - In the 64-bit edition of Windows
     `hitachi_advanced_data_binder_client.zip`
   - In the 32-bit edition of Windows
     `hitachi_advanced_data_binder_client32.zip`
   - In Linux
     `hitachi_advanced_data_binder_client-$VER.tar.gz`

   The `$VER` portion is replaced by an HADB version and release number.
2. Expand the compressed file that you copied.
   For details about the location of the JAR file after the compressed file is expanded, see Table 3-2: Location of the JAR file after the compressed file is expanded.
3. Replace the existing JAR file with the new one.
4. Confirm that a connection to the HADB server is established.
5. Delete the following folders and files:
   - The compressed file you copied in step 1.
   - The folders and files you expanded in step 2.

> **Important**
>
> Steps 1 to 3 in the preceding procedure must be performed by the same OS user. If the OS user changes, the upgrade might not be completed correctly.

■ Points to be checked after the upgrade

When the JDBC driver is upgraded, the default values of some properties# might be changed. Check whether the default values of those properties have been changed. If the default values have been changed, respecify the properties as needed. The default values of properties are the same as the values of the corresponding operands in the client environment definition. For details about the operands in the client definition, see 2.2 Contents of operands in the client definition.

#

The properties in Table 3-3: System properties that can be specified apply.

■ To downgrade the JDBC driver

See the following procedure.

Procedure

1. Replace the current JAR file with the previously used JAR file.
2. Restore the previous values of system properties.
   If the current values of system properties are different from the values that were set before the JDBC driver was upgraded, you must change the values back to the previous ones when downgrading the version.
3. Confirm that a connection to the HADB server is established.

Important

Steps 1 to 2 in the preceding procedure must be performed by the same OS user. If the OS user changes, the JDBC driver might not work correctly.

3.4 Replacing the JDBC driver with a revised version

The procedure for replacing the JDBC driver with a revised version is the same as the procedure for upgrading the JDBC driver. For details about the procedure, see 3.3 Upgrading the JDBC driver (replacing the JAR file).

3.5 Changing the time of the OS on a machine on which the JDBC driver has been installed

For details about how to change the time of the OS on which the JDBC driver has been installed, see 4.9 Changing the OS time on a client machine. When referring to that section, interpret the term client machine as *machine on which the JDBC driver has been installed*.

3.6 Uninstalling the JDBC driver

The following shows the procedure for uninstalling the JDBC driver.

Procedure

1. Delete the JAR file.
2. Delete the specification of the `CLASSPATH` environment variable.

4 Setting Up an Environment for an HADB Client (If the ODBC Driver and CLI Functions Are Used)

This chapter explains how to set up an environment for an HADB client, including installation of an HADB client and specification of environment variables.

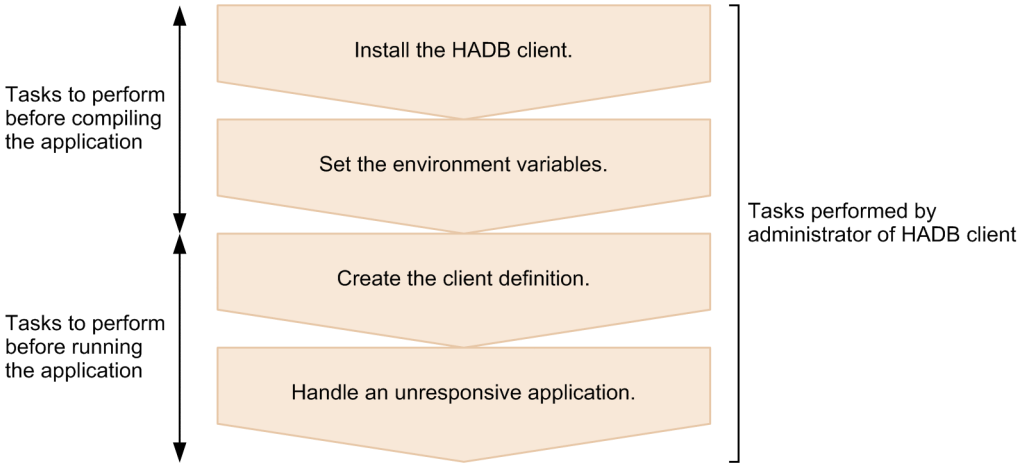
4.1 HADB client environment setup procedure

This section explains how to set up an HADB client environment.

4.1.1 HADB client for Windows

When you create or execute application programs on a computer other than an HADB server, you are using your computer as an HADB client. The following figure shows the environment setup procedure for an HADB client for Windows.

Figure 4-1: Environment setup procedure for an HADB client for Windows



Detailed description of each procedure

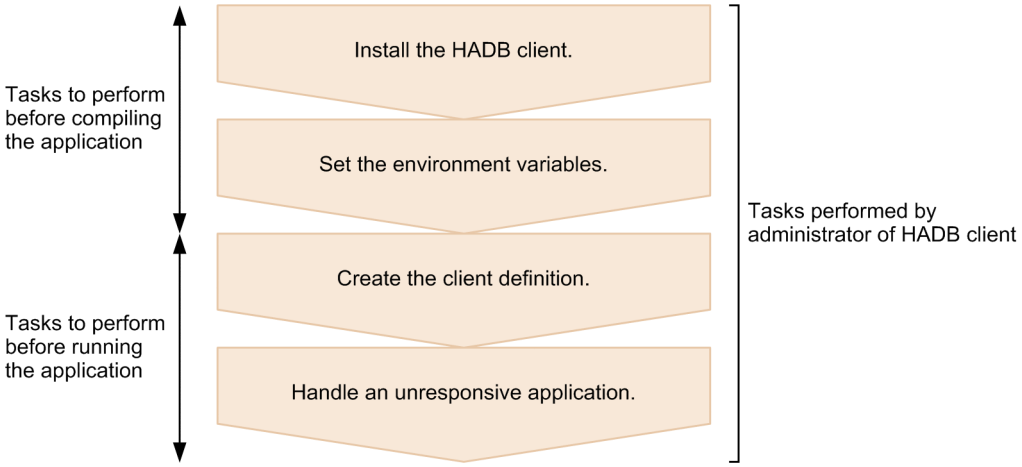
- For details about installing an HADB client, see 4.2.1 HADB client for Windows.
- For details about setting environment variables, see 4.3.1 HADB client for Windows.
- For details about creating client definitions, see 4.4 Creating a client definition.
  For explanations of operands in the client definition, see 2. Designing Client Definitions.
- For details about handling unresponsive application programs, see 4.5 Handling unresponsive application programs.

Before you install an HADB client, estimate the memory requirements for the HADB client. For the formula for estimating the memory requirements for an HADB client, see C. Estimating the Memory Requirements for an HADB Client.

4.1.2 HADB client for Linux

If you create or execute application programs on a computer other than an HADB server, you are using your computer as an HADB client. The following figure shows the environment setup procedure for an HADB client for Linux.

Figure 4-2: Environment setup procedure for an HADB client for Linux



Detailed description of each procedure

- For details about installing an HADB client, see 4.2.2 HADB client for Linux.
- For details about setting environment variables, see 4.3.2 HADB client for Linux.
- For details about creating client definitions, see 4.4 Creating a client definition.
  For explanations of operands in the client definition, see 2. Designing Client Definitions.
- For details about handling unresponsive application programs, see 4.5 Handling unresponsive application programs.

Before you install an HADB client, estimate the memory requirements for the HADB client. For the formula for estimating the memory requirements for an HADB client, see C. Estimating the Memory Requirements for an HADB Client.

4.2 Installing and uninstalling an HADB client

This section explains how to install and uninstall an HADB client.

4.2.1 HADB client for Windows

This subsection explains how to install and uninstall an HADB client for Windows.

Administrator permissions are required to install and uninstall an HADB client.

(1) Installing an HADB client

The following explains how to install an HADB client.

To install an HADB client:

1. Copy the following compressed file from the CD-ROM to any folder on the client computer:
   - 64-bit edition of Windows: `hitachi_advanced_data_binder_client.zip`
   - 32-bit edition of Windows: `hitachi_advanced_data_binder_client32.zip`

   Make sure that the length of the path for the target folder does not exceed 200 bytes.

   For information on the characters that can be used in a copy-to folder path, see <*path name*> under ■ Conventions: Syntax elements in the *Preface*.
2. Expand the compressed file.

Example:

Copy the `hitachi_advanced_data_binder_client.zip` file to `D:\hadb_clt`, and then expand the file.

The folders and files are expanded under `D:\hadb_clt\hitachi_advanced_data_binder_client\HADBCL`.

Note: This is the default target folder when the compressed file is expanded.

> **Important**
>
> To use an ODBC driver, the version information of the ODBC driver must be the same as the version information of the HADB client. Therefore, do not replace some of the DLL files in the `client\bin` folder under the client directory.

■ Tasks to be performed after installation

After you have finished the installation, perform the following tasks:

- Specifying environment variables

  For details about the environment variables to be specified, see 4.3.1 HADB client for Windows.

  Specify in the `ADBCLTDIR` environment variable the absolute path of the client directory. In the installation procedure example provided above, the client directory is `D:\hadb_clt\hitachi_advanced_data_binder_client\HADBCL`.

  A client directory stores a group of files that are related to a single client process.
- Registering the registry key

After you have specified the environment variables, register the registry key by executing the following registry registration command:

• 64-bit edition of Windows: `%ADBCLTDIR%\adbreg.reg`

• 32-bit edition of Windows: `%ADBCLTDIR%\adbreg32.reg`

- Granting write permission for the folder

  Grant write permission for the following folder to each OS user who might use the HADB client (including the ODBC driver):

  • `%ADBCLTDIR%\spool`

  • `%ADBODBTRCPATH%`

  This folder is common to 32 and 64-bit Windows.

- Reviewing the targets of scans by antivirus software

  If antivirus software is installed on the client machine where HADB client is installed, review the scope of virus scans.

  If the files and directories used by the HADB client are included in the scope of scans by antivirus software, the HADB client might not work correctly. For this reason, you need to configure the antivirus software to not scan the client directory.

(2) Uninstalling an HADB client

You must perform the following before you uninstall an HADB client:

- Check that no command or application program is being executed from the HADB client. Uninstallation might fail if an attempt is made to uninstall an HADB client while a command or application program is executing.
- Back up all the files that you might need from the client directory.

The following explains how to uninstall an HADB client.

To uninstall an HADB client:

1. Delete the registry key.
   Delete the registry key that was registered during installation by executing the following registry deletion command:
   - 64-bit edition of Windows: `%ADBCLTDIR%\adbunreg.reg`
   - 32-bit edition of Windows: `%ADBCLTDIR%\adbunreg32.reg`
2. Delete all folders and files that were copied when the HADB client was installed.
   Delete folders and files that were created after the HADB client was installed.

4.2.2 HADB client for Linux

This subsection explains how to install and uninstall an HADB client for Linux.

Important

- An HADB client is installed by the OS user who will be managing the HADB client.
- This OS user's user name must not exceed 32 bytes.

Note that the client directory, as described later in this document, stores the files that relate to a single client process.

(1) Matters to check before installing an HADB client

Before installing an HADB client, you need to make sure that the libraries required for the HADB client to operate correctly have been installed in the operating system.

▪ **How to checks**

You can check which packages are installed in the OS by running the `yum` command. Execute the command as follows:

```
yum list installed
```

Review the output of the command. If all of the packages listed in the following table have been installed, the required libraries are in place in your operating system.

Table 4-1: List of packages to check

No.	Name and version of package to check	Prerequisite library for HADB client contained in package
1	`glibc` (2.12 or later)	`libc.so.6`
2		`librt.so.1`
3		`libm.so.6`
4		`libpthread.so.0`
5		`ld-linux-x86-64.so.2` (runtime loader)
6		`libdl.so.2`
7	`libaio` (0.3.107 or later)	`libaio.so.1`
8	`openssl` (1.0.0 or later)	`libcrypto.so.10`
9	`zlib` (1.2.3 or later)	`libz.so.1`
10	`libuuid` (2.17.2 or later)	`libuuid.so.1`
11	None	`linux-vdso.so.1`#

#

Because this library is a virtual shared library provided by the kernel, you do not need to check for a corresponding package.

If any of the packages listed in the table are not installed, install them in the operating system. For details about how to install packages, see the documentation for your operating system. Install the packages as a superuser.

Note

The following shows an example of how to execute the command to check whether a specific package is installed.

Example:

To check whether the package `libaio` is installed, you execute the command as follows:

```
yum list installed | grep libaio
```

If the package `libaio` appears in the command output, this means that the package is installed. If the package `libaio` does not appear in the command output, this means that the package is not installed.

(2) Installing an HADB client

The following explains how to install an HADB client.

To install an HADB client:

1. Create a directory.

```
mkdir /home/osuser01/client
```

   Create a directory for storing the installation data (`tar.gz` file) and the `adbinstall` command. In the following example, `/home/osuser01/client` is specified as the storage directory path:

2. Assign write permission to the directory you created.

```
chmod 755 /home/osuser01/client
```

   Assign write permission so that the OS user who manages the HADB client can write to the directory you created.

3. Mount the file system CD-ROM.

   Automatically mount the file system CD-ROM that contains the installation data (`tar.gz` file) and the installation command (`adbinstall` command) for the HADB client.

   If the file system CD-ROM cannot be mounted automatically, you must mount it manually. To mount the file system CD-ROM manually, enter the following OS command:

```
mount /dev/cdrom  /media
```

   The underlined part is the mount directory of the CD-ROM file system. It might differ in your environment.

> **Important**
>
> The directory names and file names on the CD-ROM might differ depending on the computer. Execute the `ls` OS command and enter the displayed directory names as they are shown.

4. Copy the installation data (`tar.gz` file) and the installation command (`adbinstall` command) to the directory created in step 1.

```
cp /media/hitachi_advanced_data_binder_client-$VER.tar.gz /home/osuser01/client
cp /media/adbinstall /home/osuser01/client
```

   The underlined part is the mount directory of the CD-ROM file system. It might differ in your environment.

   `$VER` indicates the HADB version and release number.

   You must copy the `tar.gz` file and the `adbinstall` command to the same directory.

5. Assign execution privileges for the install command to the OS user who manages the HADB client.

```
chmod 777 /home/osuser01/client/adbinstall
```

   This command assigns execution privileges for the install command you copied in step 4 (`adbinstall` command) to the OS user who manages the HADB client.

6. Execute the installation command (`adbinstall` command).

```
/home/osuser01/client/adbinstall -c /home/osuser01/clientdir
```

   The HADB client is installed under the directory specified for the `-c` option. This directory becomes the client directory.

> **Note**
>
> If the directory specified for the -c option does not exist, the directory is automatically created when the adbinstall command is executed.

The following rules apply to the client directory:

- The client directory path must not exceed 118 bytes.
- For information on the characters that can be used in a client directory path, see <*path name*> under ■ Conventions: Syntax elements in the *Preface*.
- When you specify a directory for the -c option of the adbinstall command, make sure that the OS user who will manage the HADB client can write to the directory.

■ Action to take when KFAA91553-E message is output

In the -c option of the adbinstall command, if you specify a directory for which the OS user who manages the HADB client does not have write permissions, the KFAA91553-E message is output.

This message is also output if the OS user who manages the HADB client lacks write permissions for the directory that stores the install command (adbinstall) and the installation data (tar.gz file).

If the KFAA91553-E message is output, assign write permission to the OS user for the directory concerned.

■ Action to take when KFAA91558-W message is output

If the root user executes the adbinstall command instead of the OS user who manages the HADB client, the KFAA91558-W message is output.

Under normal circumstances, the OS user who manages the HADB client executes the adbinstall command. If the KFAA91558-W message is output, check whether executing the adbinstall command as a root user might cause any issues.

If doing so might cause an issue, press n or N when prompted for input by the KFAA91559-Q message output after the KFAA91558-W message. Then, execute the adbinstall command using the account of the OS user who manages the HADB client.

> **Note**
>
> - The KFAA91558-W message is not output if a superuser other than root executes the adbinstall command.
> - root is the user whose value is 0 in the output of the id -u OS command. This includes situations in which you use the su command of the OS to elevate an OS user to root, giving that user a value of 0 in the output of the id -u command.

■ Tasks to be performed after installation

- Setting environment variables

  After you have finished the installation, specify environment variables. For details about the environment variables to be specified, see 4.3.2 HADB client for Linux.

  Specify in the ADBCLTDIR environment variable the absolute path of the client directory. In the installation procedure example provided above, the client directory is /home/osuser01/clientdir.

- Reviewing the targets of scans by antivirus software

  If antivirus software is installed on the client machine where HADB client is installed, review the scope of virus scans.

If the files and directories used by the HADB client are included in the scope of scans by antivirus software, the HADB client might not work correctly. For this reason, you need to configure the antivirus software to not scan the client directory.

> **Note**
>
> For details about the structure of the client directory that is created when an HADB client is installed by using the `adbinstall` command, see (1) Structure of the client directory (at installation) in B.2 HADB clients for Linux.

(3) Uninstalling an HADB client

An HADB client is uninstalled by the OS user who installed the HADB client.

Before uninstalling an HADB client, perform the following:

- Check that no command or application program is being executed from the HADB client. Uninstallation might fail if an attempt is made to uninstall an HADB client while a command or application program is executing.
- Back up all the files that you might need from the client directory.

This subsection explains how to uninstall an HADB client.

To uninstall an HADB client:

1. Delete the client directory.
   The client directory must be deleted by the OS user who installed the HADB client.

   ```
   rm -rf /home/osuser01/clientdir
   ```

2. Delete the installation data (`tar.gz` file and `adbinstall` command).
   The installation data (`tar.gz` file and `adbinstall` command) that was used to install the HADB client must be deleted by the OS user who installed the HADB client.

   ```
   rm -rf /home/osuser01/client
   ```

3. Delete the specifications for the environment variables that were set during installation.

4.3 Specifying environment variables

This section explains the environment variables to be specified at the HADB client.

4.3.1 HADB client for Windows

In the HADB client for Windows, specify the environment variables listed in the following table.

Table 4-2: Values to be specified in the environment variables

No.	Environment variable	Value to be specified
1	PATH	Add the following folder to this environment variable: **64-bit edition of Windows:** • %ADBCLTDIR%\client\bin • %ADBCLTDIR%\vclib **32-bit edition of Windows:** • %ADBCLTDIR%\client\bin • %ADBCLTDIR%\vclib32 Specify this environment variable as a system environment variable.
2	TZ	Specify in this environment variable the time zone of the computer on which the HADB client is installed. Specify this environment variable as a system environment variable. Do not specify a time zone that uses leap seconds.
3	ADBCLTDIR	Specify in this environment variable the absolute path of the client directory. Be sure to specify this environment variable before you execute the registry registration or deletion command. Specify the first three characters of the path in accordance with the following rules: • First character Alphabetic character indicating the drive • Second character Colon (:) • Third character Forward slash (/) or backlash (\)
4	ADBCLTLANG	Specify in this environment variable the character encoding used on the HADB client. Specify the same character encoding as used on the HADB server. This must be the character encoding specified in the ADBLANG environment variable for the HADB server. • If using Unicode (UTF-8) on the HADB server Specify UTF8 in this environment variable. • If using Shift-JIS on the HADB server Specify SJIS in this environment variable.
5	ADBMSGLOGSIZE	Specify in this environment variable the maximum size (in megabytes) of a client message log file. The permitted value range is from 1 to 2000. Four client message log files are created on an HADB client. If you omit this environment variable, the maximum size of each client message log file is set to 16 megabytes. If you start multiple client processes from a single client directory, specify for each of them the same value in this environment variable. If different values are specified when multiple client processes are run, the client message log files might become corrupted.
6	ADBODBTRC	Specify whether to output HADB ODBC driver trace information. Specify one of the following values in this environment variable: • YES: Specify this value to output HADB ODBC driver trace information.

No.	Environment variable	Value to be specified
		• NO: Specify this value to not output HADB ODBC driver trace information. If you omit this environment variable or specify an invalid value, NO is assumed. Consider setting this environment variable when using the ODBC driver.
7	ADBODBTRCSIZE	Specify the maximum size of an HADB ODBC driver trace file (in MB). You can specify a value in the range from 32 to 1,024. If you omit this environment variable or specify an invalid value, 256 is assumed. Consider setting this environment variable when using the ODBC driver.
8	ADBODBTRCPATH	Specify the absolute path of the folder in which to store HADB ODBC driver trace files. The path name cannot be longer than 210 bytes. In the following circumstances, HADB ODBC driver trace information is not output even if YES is specified in ADBODBTRC: • This environment variable is omitted. • An invalid folder is specified. • The path name is 211 bytes or longer. Consider setting this environment variable when using the ODBC driver.
9	ADBODBTRCLV	This environment variable specifies the trace level to apply when outputting HADB ODBC driver trace information. Specify one of the following values: • 1: Specify this value to output information at trace level 1. • 2: Specify this value to output information at trace level 2. For details about trace levels, see 17.4.1 About trace levels. If you omit this environment variable or specify an invalid value, 1 is assumed. Consider setting this environment variable when using the ODBC driver.
10	ADBODBAPMODE	Specify the application mode of the HADB ODBC driver. Normally, you do not need to specify this environment variable. The value specified for this environment variable changes the behavior of the HADB ODBC driver. • ACCESS: The HADB ODBC driver operates in Microsoft Access(R) compatibility mode, rather than following the ODBC 3.5 specification. Specify this value when using Microsoft Access to access the HADB server. Specifying this value allows you to avoid certain issues, such as #Deleted replacing search results or errors occurring. • NORMAL: The HADB ODBC driver operates as normal. If you omit this environment variable or specify an invalid value, NORMAL is assumed. Consider setting this environment variable when using the ODBC driver.

4.3.2 HADB client for Linux

In the HADB client for Linux, specify the environment variables listed in the table below. These environment variables are specified by the OS user who manages the HADB client.

Make sure that the values specified for the environment variables take effect on the shell when the HADB client is used. For details about the specification method, see the shell documentation.

Table 4-3: Values to be specified in the environment variables

No.	Environment variable	Value to be specified
1	LANG	Specify in this environment variable the character encoding used in the OS. This must be the same as the value of the LANG environment variable for the HADB server.

No.	Environment variable	Value to be specified
2	LD_LIBRARY_PATH	Add the following directory to the value of this environment variable: • $ADBCLTDIR/client/lib Set this environment variable when either of the following conditions is satisfied: • The adbsql command will be run on the HADB client. • Applications that use the CLI function will be developed or run on the HADB client.
3	PATH	Add the following directory to the value of this environment variable: • $ADBCLTDIR/client/bin
4	TZ	Specify in this environment variable the time zone of the computer on which the HADB client is installed. Do not specify a time zone that uses leap seconds.
5	ADBCLTDIR	Specify in this environment variable the absolute path of the client directory. The first character of the path must be a forward slash (/).
6	ADBCLTLANG	Specify in this environment variable the character encoding used on the HADB client. Specify the same character encoding as used on the HADB server. This must be the character encoding specified in the ADBLANG environment variable for the HADB server. • If using Unicode (UTF-8) on the HADB server Specify UTF8 in this environment variable. • If using Shift-JIS on the HADB server Specify SJIS in this environment variable.
7	ADBMSGLOGSIZE	Specify in this environment variable the maximum size (in megabytes) of a client message log file. The permitted value range is from 1 to 2000. Four client message log files are created on an HADB client. If you omit this environment variable, the maximum size of each client message log file is set to 16 megabytes. If you start multiple client processes from a single client directory, specify for each of them the same value in this environment variable. If different values are specified when multiple client processes are run, the client message log files might become corrupted.
8	ADBSQLNULLCHAR	For this environment variable, specify the character string that will be displayed as a null value in the search result returned by the adbsql command. You can specify a character string that is 0 to 32 bytes long. If you specify a 0-byte character string, a blank is displayed as a null value. If you omit this environment variable, null values are displayed as asterisks (*). Specify this environment variable when retrieved data contains asterisks, or you want a particular character string to appear in place of null values. Note that if you specify multi-byte characters, the search results might become garbled. Consider setting this environment variable when the adbsql command will be run on the HADB client.

4.4 Creating a client definition

Specify a client definition to set up an environment for running the HADB client. The OS user who manages the HADB client creates the client definition.

4.4.1 How to create a client definition

The following explains how to create a client definition.

To create a client definition:

1. Copy the model client definition file (`client.def`#) from `%ADBCLTDIR%\sample\conf\` to `%ADBCLTDIR%\conf\`.

   #: For an HADB client for Linux, the model client definition file is `$ADBCLTDIR/sample/conf/client.def`.

2. Use a text editor to open the copied client definition file (`%ADBCLTDIR%\conf\client.def`#).

   #: For an HADB client for Linux, the model client definition file is `$ADBCLTDIR/conf/client.def`.

3. Specify each operand in the client definition. For explanations of operands in the client definition, see 2. Designing Client Definitions.

4. After you have created the client definition, overwrite the file. Do not change the file name (`client.def`).

Grant read and write permissions for the client definition file only to the OS user who manages the HADB client.

The specified operand values in the client definition are output as the `KFAA50027-I` message to the client message log file when a connection handle is allocated.

4.4.2 Notes about changing a client definition

Before you change a client definition, disconnect from the HADB server the application program that uses that client definition file.

4.4.3 Choosing a client definition

You specify the path of the client definition file in the `ClientDefPath` argument of the `a_rdb_SQLAllocConnect()` CLI function. If you create multiple client definitions, you can choose the appropriate file when you allocate a connection handle.

If the `ClientDefPath` argument is omitted, `%ADBCLTDIR%\conf\client.def`# is assumed. For details about `a_rdb_SQLAllocConnect()`, see 19.2.1 a_rdb_SQLAllocConnect() (allocate a connection handle).

#: For an HADB client for Linux, `$ADBCLTDIR/conf/client.def` is assumed.

4.5 Handling unresponsive application programs

A problem such as a communication error, temporary failure (including power outage), or disk failure might cause an application program to stop responding. Such an unresponsive application program might delay other application programs and command processing.

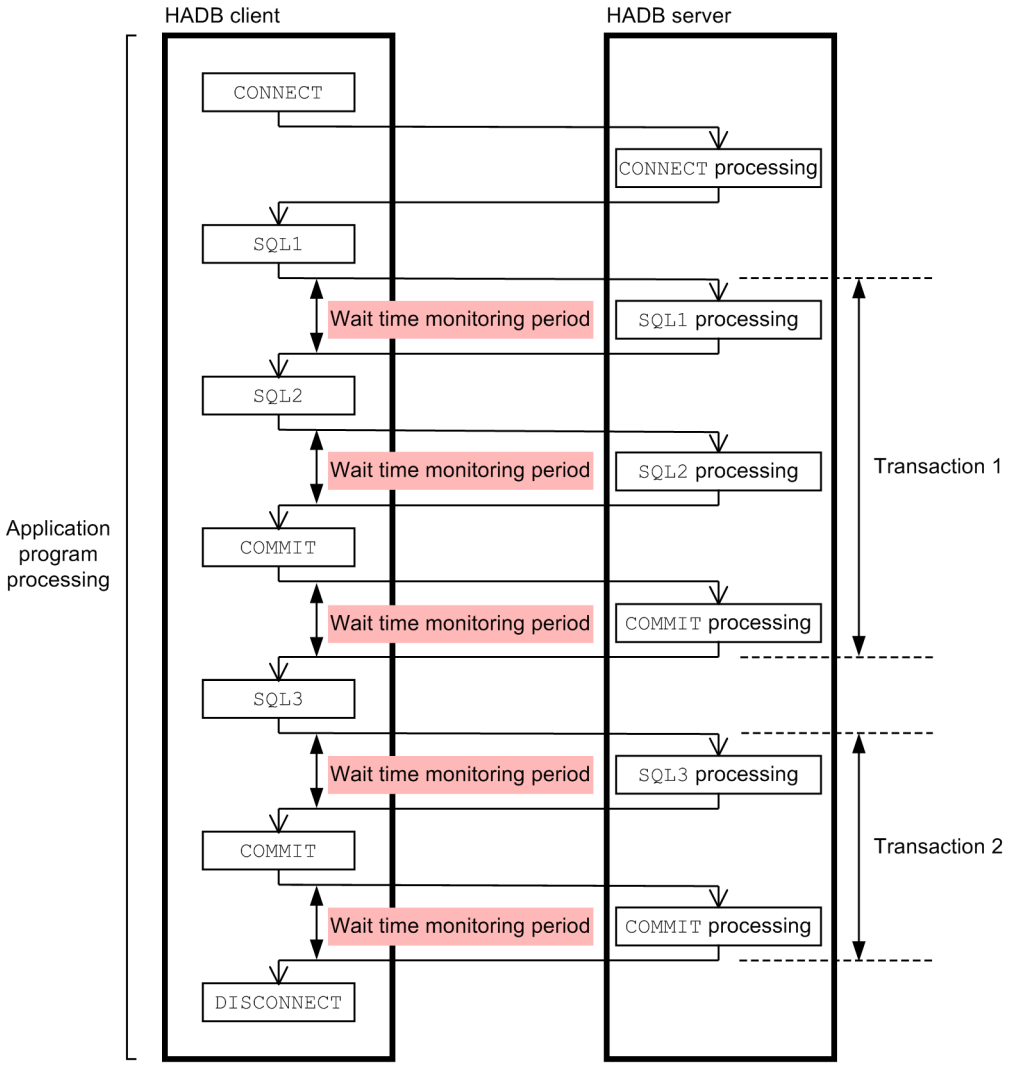
Specify the following operand in the client definition in order to minimize the effects of an unresponsive application program:

- `adb_clt_rpc_sql_wait_time` operand

Specify in this operand the maximum amount of time to wait for a response after a processing request has been sent from the HADB client to the HADB server. If there is no response from the HADB server within the specified time, a timeout error whose `SQLCODE` is `-732` (`KFAA30732-E`) is returned to the application, processing of the SQL statement is canceled, and the transaction is rolled back. Then, the application is disconnected from the HADB server.

The following figure shows the wait time monitoring procedure based on the specification of the `adb_clt_rpc_sql_wait_time` operand.

Figure 4-3: Wait time monitoring procedure based on the specification of the adb_clt_rpc_sql_wait_time operand



Explanation:

This example monitors the wait time until a response is received after a processing request has been issued from the HADB client to the HADB server. For example, if 600 seconds is specified in the `adb_clt_rpc_sql_wait_time` operand, a wait time of 600 seconds is set for the monitoring interval. Therefore, as a guideline, specify a wait time at least equal to the longest SQL processing time. Specify a realistic value that most likely will indicate when the application program has become unresponsive.

For details about the `adb_clt_rpc_sql_wait_time` operand, see the explanation of the adb_clt_rpc_sql_wait_time operand in 2.2.2 Operands related to application program status monitoring.

4.6 Upgrading an HADB client

This section explains how to upgrade an HADB client.

If you are replacing the HADB client with a revised version rather than upgrading it, see 4.8 Replacing HADB client with a revised version. 4.8 Replacing HADB client with a revised version also explains when replacement with a revised version of the HADB client applies.

4.6.1 Preparations before upgrading an HADB client

You must perform the following tasks before you attempt to upgrade an HADB client.

(1) Check environment variables

When an HADB client is upgraded, the values of environment variables might change. For details, see 4.3 Specifying environment variables.

(2) Check the client definition

When an HADB client is upgraded, some client definition default values might change. If you have set client definition operands to their default values, see 2.2 Contents of operands in the client definition and check whether operand values have been changed.

Note that the following operand cannot be specified in the client definition of HADB client version 03-00 or later. If you are using version 03-00 or later and this operand is specified, delete it from the client definition.

- `adb_clt_rpc_open_wait_time`

(3) Back up the client directories

Before you upgrade your HADB client, back up the client directories.

Delete the backup after you have confirmed that the new version operates correctly.

4.6.2 Notes about upgrading

Note the following about upgrading an HADB client:

- You must not change the OS user while the HADB client is being upgraded. If the OS user is changed, the HADB client might not be upgraded successfully.
- You must not change the value of the `ADBCLTLANG` environment variable while the HADB client is being upgraded. If the value is changed, the HADB client might not be upgraded successfully.
- Make sure that the versions of the HADB client and ODBC driver match. If the versions do not match, an error occurs before the HADB client connects to the HADB server.

4.6.3 How to upgrade an HADB client

This subsection provides the steps to upgrade an HADB client.

(1) Upgrading an HADB client (Windows edition)

The OS user who manages the HADB client upgrades the HADB client.

To upgrade an HADB client:

Procedure

1. Log in as the OS user who manages the HADB client.
2. Copy the zip file from the installation media to a folder of your choice.
3. Expand the zip file to the client directory specified in the `ADBCLTDIR` environment variable.
   If you expand the zip file to a path other than the client directory, perform the following additional step:
   - Change the client directory path specified in the `ADBCLTDIR` environment variable.

(2) Upgrading an HADB client (Linux edition)

To upgrade an HADB client:

Procedure

1. Log in as the OS user who manages the HADB client.
2. Create a directory.

   ```
   mkdir /home/osuser01/client
   ```

   Create the directory `/home/osuser01/client` in which the installation data (`tar.gz` file) and the `adbinstall` command will be stored.
3. Assign write permission to the directory you created.

   ```
   chmod 755 /home/osuser01/client
   ```

   Assign write permission to the directory you created so that the OS user who manages the HADB client can write to the directory.
4. Copy the installation data (`tar.gz` file) and installation command (`adbinstall`) from the installation media to the directory you created in step 2.

   ```
   cp /media/hitachi_advanced_data_binder_client-$VER.tar.gz /home/osuser01/client
   cp /media/adbinstall /home/osuser01/client
   ```

   The underlined portion indicates the mount directory of the CD-ROM file system. The actual mount directory will depend on the environment.

   `$VER` represents the HADB version and release number.

   You must copy the `tar.gz` file and the `adbinstall` command to the same directory.
5. Assign execution permission for the installation command to the OS user who manages the HADB client.

   ```
   chmod 777 /home/osuser01/client/adbinstall
   ```

   Assign execution permission for the installation command (`adbinstall`) you copied in step 4 to the OS user who manages the HADB client.

6. Execute the installation command (adbinstall command).

```
/home/osuser01/client/adbinstall -c /home/osuser01/clientdir
```

As the client directory path in the -c option, specify a directory of your choice or the path specified in the ADBCLTDIR environment variable.

The directory you specify must be one for which the OS user who manages the HADB client has write permission.

If you specify a path in the -c option that differs from the path specified in the ADBCLTDIR environment variable, perform the following additional step:

- In the ADBCLTDIR environment variable, specify the path of the directory you specified in the -c option.

> **Note**
>
> For details about how to execute the adbinstall command and the rules, see *adbinstall (Install HADB Server or Client)* in the manual *HADB Command Reference*.

▪ **Action to take when KFAA91553-E message is output**

If the OS user who manages the HADB client does not have write permission for the directory specified in the -c option of the adbinstall command, the KFAA91553-E message is output.

The KFAA91553-E message is also output if the OS user who manages the HADB client does not have write permission for the directory that contains the installation command (adbinstall) and installation data (tar.gz file).

If the KFAA91553-E message is output, assign write permission for the directory concerned.

▪ **Action to take when KFAA91558-W message is output**

If the root user executes the adbinstall command instead of the OS user who manages the HADB client, the KFAA91558-W message is output.

Under normal circumstances, the OS user who manages the HADB client executes the adbinstall command. If the KFAA91558-W message is output, check whether executing the adbinstall command as root might cause any issues.

If doing so might cause an issue, press n or N when prompted for input by the KFAA91559-Q message output after the KFAA91558-W message. Then, execute the adbinstall command using the account of the OS user who manages the HADB client.

> **Note**
>
> - The KFAA91558-W message is not output if a superuser other than root executes the adbinstall command.
> - root is the user whose value is 0 in the output of the id -u OS command. This includes situations in which you use the su command of the OS to elevate an OS user to root, giving that user a value of 0 in the output of the id -u OS command.

4.6.4 Tasks to be performed after upgrading

Use the procedure described below to verify that the HADB client upgraded successfully.

(1) HADB client (Windows edition)

Check one of the following file properties to verify that the new version of the file is installed:

- 64-bit edition of Windows: `%ADBCLTDIR%client\bin\adbclt.dll`
- 32-bit edition of Windows: `%ADBCLTDIR%client\bin\adbclt32.dll`

If an ODBC driver is used, also check the versions of all DLL files whose names begin with `adbodbc`.

(2) HADB client (Linux edition)

Verify that the HADB client has been upgraded successfully:

Procedure

1. Execute the `adbsql` command to connect to the HADB server.
2. Check the contents of the `KFAA70003-I` message.
   Verify that the `KFAA70003-I` message output to the client's message log file displays the new version.

4.7 Downgrading an HADB client version (restoring the previous version)

This section describes how to downgrade an HADB client.

When you downgrade an HADB server, you also need to downgrade the HADB clients. That is, the version of the HADB client must match the version of the HADB server.

> **Note**
>
> For details about downgrading the HADB server, see *Downgrading the HADB server version (restoring the previous version)* under *Building a System* in the *HADB Setup and Operation Guide*.

4.7.1 Preparations before downgrading an HADB client

Before downgrading an HADB client, check the values specified for the following elements:

- Environment variables
- Client definition

Any values that you changed when upgrading the HADB client must be changed back to values appropriate to the earlier version before downgrading.

4.7.2 Notes about downgrading

Note the following when downgrading an HADB client:

- Do not change the OS user during the process of downgrading the HADB client. If you change the OS user, the downgrade process might not work correctly.
- Do not change the value specified for the `ADBCLTLANG` environment variable when downgrading the HADB client. If you change the value, the downgrade process might not work correctly.
- Make sure that the versions of the HADB client and ODBC driver match. If the versions do not match, an error occurs before the HADB client connects to the HADB server.

4.7.3 Downgrade procedure

The following explains the procedure for downgrading an HADB client.

The user who performs this procedure is the OS user who manages the HADB client.

(1) Downgrading an HADB client (Windows edition)

To downgrade an HADB client:

Procedure

1. Log in as the OS user who manages the HADB client.

2. Copy the zip file from the installation media to a folder of your choice.
3. Expand the zip file to the client directory specified in the ADBCLTDIR environment variable.

   If you expand the zip file to a path other than the client directory, perform the following additional step:

   - Change the client directory path specified in the ADBCLTDIR environment variable.

(2) Downgrading an HADB client (Linux edition)

To downgrade an HADB client:

Procedure

1. Log in as the OS user who manages the HADB client.
2. Create a directory.

   ```
   mkdir /home/osuser01/client
   ```

   Create the directory (/home/osuser01/client) in which the installation data (tar.gz file) and the adbinstall command will be stored.
3. Assign write permission to the directory you created.

   ```
   chmod 755 /home/osuser01/client
   ```

   Assign write permission to the directory you created so that the OS user who manages the HADB client can write to the directory.
4. Copy the installation data (tar.gz file) and installation command (adbinstall) from the installation media to the directory you created in step 2.

   ```
   cp /media/hitachi_advanced_data_binder_client-$VR.tar.gz /home/osuser01/client
   cp /media/adbinstall /home/osuser01/client
   ```

   The underlined portions indicate the mount directory of the CD-ROM file system. The actual mount directory will depend on the environment.

   $VR represents the HADB version and release number.

   You must copy the tar.gz file and the adbinstall command to the same directory.
5. Assign execution privilege for the installation command to the OS user who manages the HADB client.

   ```
   chmod 777 /home/osuser01/client/adbinstall
   ```

   Assign execution privilege for the installation command (adbinstall) you copied in step 4 to the OS user who manages the HADB client.
6. Run the installation command (adbinstall).

   ```
   /home/osuser01/client/adbinstall -c /home/osuser01/clientdir
   ```

   As the client directory path in the -c option, specify a directory of your choice or the path specified in the ADBCLTDIR environment variable.

   The directory you specify must be one for which the OS user who manages the HADB client has write permission.

   If you specify a path in the -c option that differs from the path specified in the ADBCLTDIR environment variable, perform the following additional step:

   - In the ADBCLTDIR environment variable, specify the path of the directory you specified in the -c option.

> **Note**
>
> For details about how to run the `adbinstall` command and the rules that apply when doing so, see *adbinstall (Install HADB Server or Client)* in the manual *HADB Command Reference*.

▪ **Action to take when message `KFAA91553-E` is output**

If the OS user who manages the HADB client does not have write permission for the directory specified in the `-c` option of the `adbinstall` command, the message `KFAA91553-E` is output.

The message `KFAA91553-E` is also output if the OS user who manages the HADB client does not have write permission for the directory that contains the installation command (`adbinstall`) and installation data (`tar.gz` file).

If the message `KFAA91553-E` is output, assign write permission for the directory concerned.

4.7.4 Tasks to be performed after downgrading

Use the following procedure to verify that the HADB client has been downgraded successfully.

(1) HADB client (Windows edition)

Check the properties of one of the following files to verify that the old version of the file is installed.

- In 64-bit Windows: `%ADBCLTDIR%client\bin\adbclt.dll`
- In 32-bit Windows: `%ADBCLTDIR%client\bin\adbclt32.dll`

If an ODBC driver is used, also check the versions of all DLL files whose names begin with `adbodbc`.

(2) HADB client (Linux edition)

To verify that the HADB client has been downgraded successfully:

Procedure

1. Connect to the HADB server by executing the `adbsql` command.
2. Check the message `KFAA70003-I`.

   Confirm that the message `KFAA70003-I` output to the client message log file displays the old version.

4.8 Replacing HADB client with a revised version

This section describes how to replace HADB client with a revised version.

When the following condition is met, you can replace HADB client with a revised version instead of upgrading it.

- You intend to perform an overwrite installation of HADB client with an edition that has the same version and revision number but a different code

> **Note**
>
> The following explains how to identify the version number, revision number, and code.
>
> As an example, consider a HADB client with the version information 03-02-/A:
>
> - 03 is the version number.
> - 02 is the revision number.
> - The underlined part (-/A) is the code.

The following shows examples that constitute replacement with a revised version of HADB client is possible, and examples that do not.

▪ **Scenarios where HADB client is replaced with a revised version**

For example, in the following scenarios, because the version numbers and revision numbers are the same, the process constitutes replacement with a revised version.

03-02 -> 03-02-/A

03-02-/B -> 03-02-/D

▪ **Scenarios where HADB client is not replaced with a revised version**

For example, in the following scenarios, because the version number or revision number differs, the process does not constitute replacement with a revised version.

03-02 -> 03-03-/A

03-02-/B -> 03-03-/D

This scenario is considered to be an upgrade of the HADB client. For details about how to upgrade an HADB client, see 4.6 Upgrading an HADB client.

4.8.1 Procedure for replacing HADB client with a revised version

To replace an HADB client with a revised version:

Procedure

1. Perform the required preparation.

   The preparation required before you can replace the HADB client is the same as before upgrading an HADB client. For details, see 4.6.1 Preparations before upgrading an HADB client.

2. Check the cautionary notes that apply to HADB client replacement.

   The cautionary notes regarding HADB client replacement are the same as when upgrading the HADB client. For details, see 4.6.2 Notes about upgrading.

3. Replace the HADB client with the revised version.

The procedure for replacing the HADB client is the same as the upgrade process. For details, see 4.6.3 How to upgrade an HADB client.

4. Confirm that the replacement process is complete.

   Confirm that the code in the version information of the HADB client has changed. For details about the confirmation method, see 4.6.4 Tasks to be performed after upgrading.

4.9 Changing the OS time on a client machine

This section explains how to change the OS time on a client machine on which HADB client is installed.

To change the OS time on a client machine, first read 4.9.1 Notes (changing the OS time), and then perform the procedure described in one of the following subsections:

- 4.9.2 How to advance the OS time on a client machine
- 4.9.3 How to restore the OS time on a client machine

4.9.1 Notes (changing the OS time)

The following notes apply to changing the OS time on a client machine:

- Before you change the OS time on a client machine, make sure that all application programs that connect to the HADB server are stopped. If the OS time is changed on a client machine while an application program connected to the HADB server is running, unexpected HADB server and client operations might occur.
- After the OS time has been changed on a client machine, the new OS time is applied to the items below. This change might affect application programs that handle time.
  - Command execution time
  - Timestamps in message logs in the client message log file

4.9.2 How to advance the OS time on a client machine

First read 4.9.1 Notes (changing the OS time), and then follow the procedure described below. To advance the OS time on a client machine on which HADB client is installed:

Procedure

1. Terminate all application programs that connect to the HADB server.
   Terminate all application programs that connect from the HADB client to the HADB server.
2. Advance the OS time on the client machine.
   After all application programs have terminated, advance the OS time on the client machine on which the HADB client is installed.
3. Start the application programs that connect to the HADB server.
   After you have advanced the OS time on the client machine, start the application programs that were terminated.

The OS time is now advanced on the client machine.

4.9.3 How to restore the OS time on a client machine

First read 4.9.1 Notes (changing the OS time), and then follow the procedure described below. To restore the OS time on a client machine on which HADB client is installed:

Procedure

1. Terminate all application programs that connect to the HADB server.

   Terminate all application programs that connect from the HADB client to the HADB server.

2. Restore the OS time on the client machine.

   After all application programs have terminated, restore the OS time on the client machine on which the HADB client is installed.

3. Start the application programs that connect to the HADB server.

   After you have restored the OS time on the client machine, start the application programs that were terminated.

The OS time is now restored on the client machine.

5 Designs Related to Improvement of Application Program Performance

This chapter explains designs related to improving application program performance.

5.1 How to retrieve tables

HADB supports the following three retrieval methods for searching tables:

- Table scan
- Index scan
- Key scan

HADB determines the table retrieval method automatically. You can check which method was used when executing a SQL statement by checking the access path. For details about access paths, see the following subsections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information displayed in access paths
  See (20) Table retrieval method in 6.1.4 Information displayed in the tree view.

> **Note**
>
> By using an index specification, you can specify the index to use when retrieving a table, or prevent the use of indexes for a particular retrieval. For details about index specifications, see *Specification format and rules for index specifications* in the manual *HADB SQL Reference*.

Note that *table* here refers to a base table.

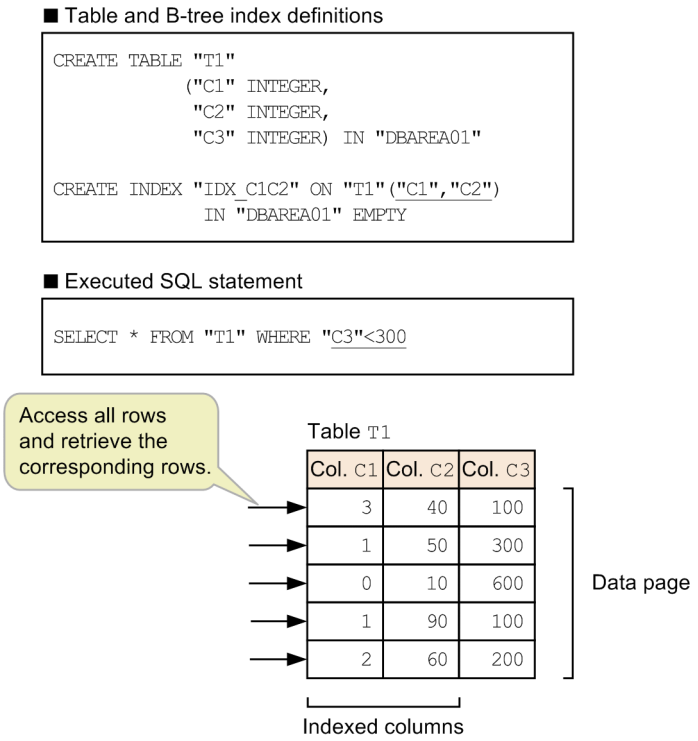
5.1.1 About table scans

A table scan is a base table retrieval method that does not use B-tree indexes and text indexes. A table scan is used in the following cases:

- No B-tree index or text index is defined for the table.
- No search condition is specified that can effectively use a B-tree index or text index.
- `WITHOUT INDEX` is specified in the index specification.

The following figure shows an example of a table scan.

Figure 5-1: Example of a table scan



Explanation:

Because column C3 specified in the search conditions is not an indexed column, B-tree index IDX_C1C2 is not used when the SELECT statement shown above is executed. Therefore, HADB performs a table scan and accesses all rows in the data pages.

If range indexes are defined for the table, the range indexes might also be used.

Note

If you have specified in the index specification that B-tree indexes or text indexes are not to be used, HADB uses a table scan.

Important

If a table scan is to be performed, we recommend that you define range indexes for the table. Using range indexes might improve performance. For details about the conditions under which range indexes are used, see 5.3 Range indexes used during execution of SQL statements.

5.1.2 About index scans

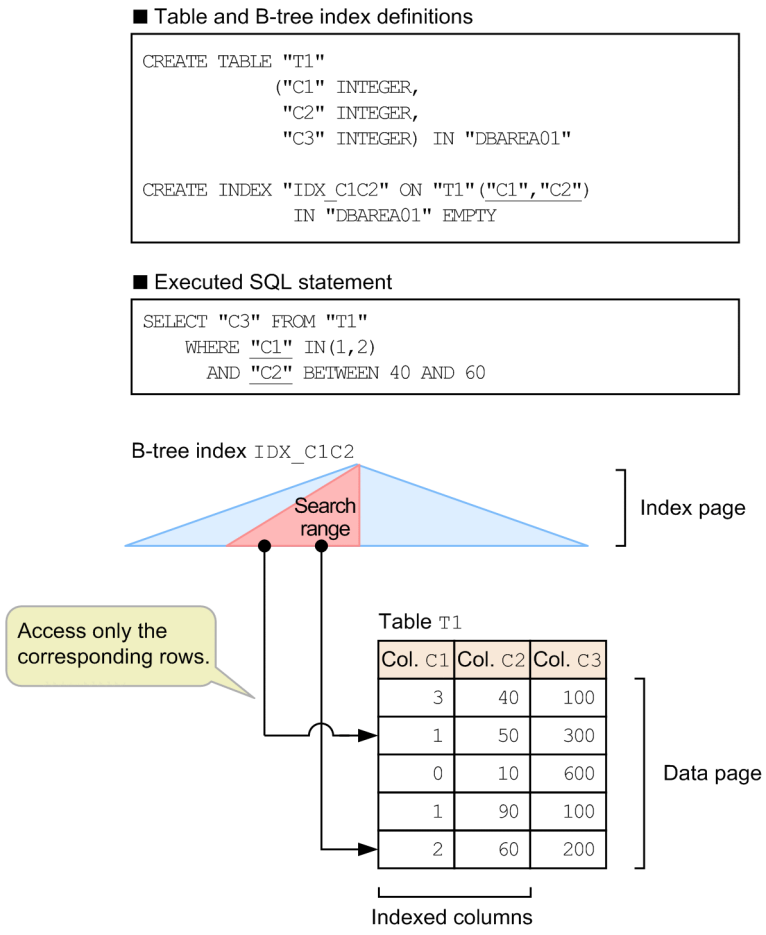
An index scan uses a B-tree index or text index to evaluate the search conditions and retrieves from data pages all rows that satisfy the search conditions. An index scan is run in the following cases:

- B-tree indexes or text indexes are defined in the table, and a search condition is specified that can effectively use these indexes.

- A B-tree index or text index that can be used for retrieval is specified in the index specification.

The following figure shows an example of an index scan.

Figure 5-2: Example of an index scan



Explanation:

This example uses B-tree index `IDX_C1C2` to evaluate the search conditions and accesses a data page to retrieve the column `C3` values.

If range indexes are defined for the table, the range indexes might also be used.

5.1.3 About key scans

A key scan uses a B-tree index to evaluate the search conditions and retrieves from index pages the column values in the rows that satisfy the search conditions. This method can reduce the number of pages to be referenced because it retrieves column values directly from B-tree-indexed columns (keys).

A key scan is performed when a B-tree index is defined as an indexed column for all the columns specified in the SQL statement, and one of the following conditions is met:

- A B-tree index is defined in the table, and a search condition is specified that can effectively use the B-tree index.
- The B-tree index used for retrieval is specified in an index specification.
- A set function `MIN` or `MAX` is specified.#

- `SELECT DISTINCT` is specified.#
- `UNION` or `UNION DISTINCT` is specified.#
- `EXCEPT` or `EXCEPT DISTINCT` is specified.#
- `INTERSECT` or `INTERSECT DISTINCT` is specified.#
- A quantified predicate with `=ANY` specification is specified.#
- An `IN` predicate with a table subquery is specified.#

#

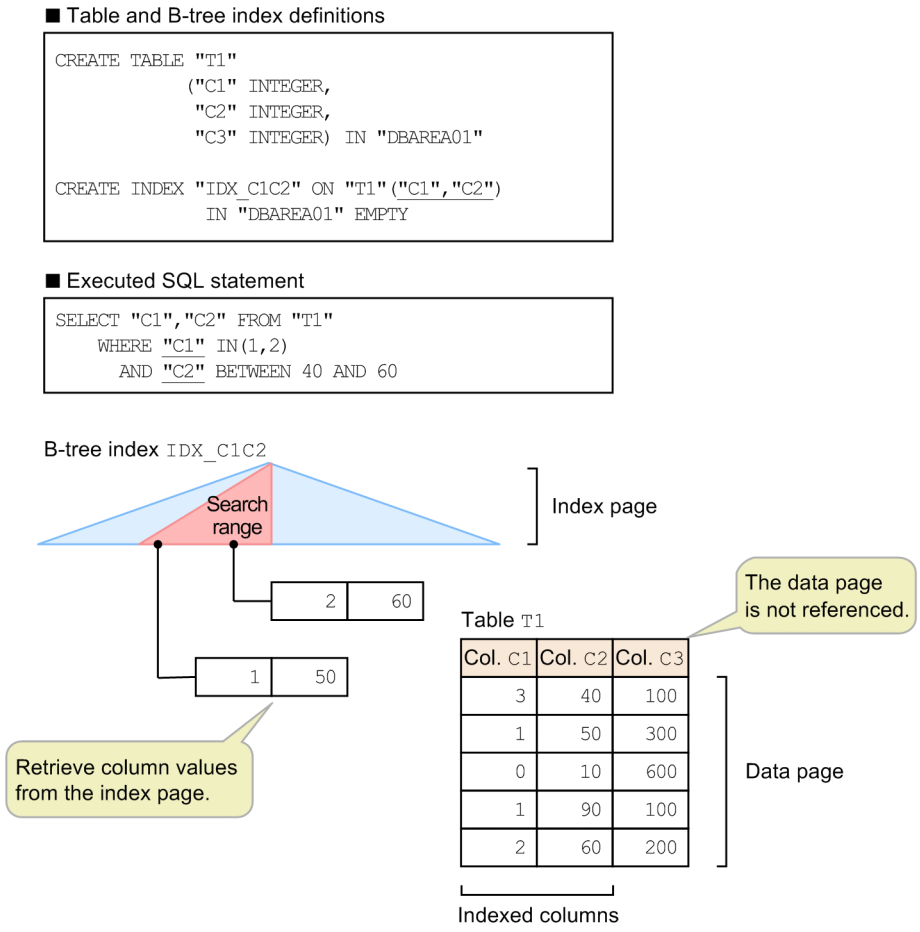
The B-tree index is used only when HADB determines that the B-tree index can be used efficiently.

However, if any of the following conditions are satisfied, HADB might reference the data pages:

- A table in which rows have been added, updated, or deleted is being retrieved
- A column of the `VARCHAR` type that contains data ending with a single-byte space is being retrieved
- Data in the `VARBINARY` column whose binary data ends with `X'00'` is being referenced

The following figure shows an example of a key scan.

Figure 5-3: Example of a key scan



Explanation:

This example uses B-tree index `IDX_C1C2` to evaluate the search conditions. HADB does not access data pages because it retrieves the values of columns `C1` and `C2` directly from the index page.

Note

Key scans are not executed for text indexes.

5.2 B-tree indexes and text indexes used during execution of SQL statements

You must define B-tree indexes and text indexes that are appropriate for your intended search conditions because the availability of B-tree indexes and text indexes greatly affects performance.

This section explains how to determine the B-tree indexes and text indexes to be used during execution of SQL statements, and how to check the index used during execution of an SQL statement.

In this section, the term index refers to both B-tree indexes and text indexes.

Notes

- The index selection method explained here is applicable to query expressions obtained after internal derived tables have been expanded or to search conditions that have been converted by equivalent exchange. For details about expansion of internal derived tables, see the topic *Internal derived tables* in the manual *HADB SQL Reference*. For details about equivalent exchange for search conditions, see 5.11 Equivalent exchange of search conditions.
- If, in a value expression specified in the search conditions, only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.
- When an index is specified, that index is used, regardless of the B-tree index priority or selection rules described here. For details about index specifications, see *Specification format and rules for index specifications* in the manual *HADB SQL Reference*.
- When joining tables, depending on the table joining method, indexes might not be used when evaluating the join condition. For details about table joining methods, see 5.5 Table joining methods.
- You cannot define a text index for a column store table.

Important

B-tree indexes defined for a column store table are used in the following cases:

- An index specification is specified.
  For details about index specifications, see *Specification format and rules for index specifications* in the manual *HADB SQL Reference*.
- A set function `MIN` or `MAX` is specified.#
- `SELECT DISTINCT` is specified.#
- `UNION` or `UNION DISTINCT` is specified.#
- `EXCEPT` or `EXCEPT DISTINCT` is specified.#
- `INTERSECT` or `INTERSECT DISTINCT` is specified.#
- A quantified predicate with `=ANY` specification is specified.#
- An `IN` predicate with a table subquery is specified.#
- A column store table is specified as the table to be updated by the `UPDATE` statement.
- A column store table is specified as the table to be deleted by the `DELETE` statement.
- Cost information for a column store table is collected.

#

The B-tree index is used only when the HADB server determines that the B-tree index can be used efficiently.

For details about how to check the B-tree index to be used, see 5.2.5 How to check the index used during execution of an SQL statement.

5.2.1 Priority and selection rules for indexes

If multiple indexes are defined for a table, HADB determines which index to use based on the search condition specified in the `WHERE` clause, or the `ON` search condition specified for a joined table.

This section explains the priority and selection rules for indexes used during execution of an SQL statement.

(1) Index priority

If the predicate specified for a search condition is in any of the formats shown in the following table, an index is used during execution of an SQL statement. Note that only one index is used during execution of an SQL statement. Therefore, if multiple indexes are defined for a table, the index to be used is determined according to the priority shown in the following table.

Table 5-1: Index priority when multiple indexes are defined for a table

Priority	Condition for index	Example of this priority (C1 is the column for which the index has been defined)
1	Unique index (B-tree index) that specifies all indexed columns in a = condition	`"C1"=100`
		`"C1"=100+?`
		`"T1"."C1"=CAST("T2"."C1" AS INTEGER)`
2	B-tree index with indexed columns specified in a = condition	`"C1"=100`
		`"C1"=100+?`
		`"T1"."C1"=CAST("T2"."C1" AS INTEGER)`
3	B-tree index with indexed columns specified in an `IS NULL` condition	`"C1" IS NULL`
4	B-tree index with indexed columns specified in a `LIKE` predicate that satisfies all the following conditions: • A literal is specified as the pattern character string • The predicate performs a leading-match search that specifies the special character `%` Escape characters must be specified as literals.	`"C1" LIKE 'ABC%'`
		`"C1" LIKE 'AB\%C%' ESCAPE '\'`
5	B-tree index with indexed columns specified in a `LIKE` predicate that satisfies both of the following conditions: • A literal is specified as the pattern character string • The predicate performs a leading-match search specified in a way other than that described for priority 4 Escape characters must be specified as literals.	`"C1" LIKE 'ABC__'`
		`"C1" LIKE 'ABC%E'`
		`"C1" LIKE 'AB\_C%E' ESCAPE '\'`
	B-tree index with indexed columns specified in a `LIKE` predicate that uses complete-match retrieval with a literal specified as the pattern character string.	`"C1" LIKE 'ABCDE'`
		`"C1" LIKE 'AB\_CDE' ESCAPE '\'`

Priority	Condition for index	Example of this priority (C1 is the column for which the index has been defined)
	Escape characters must be specified as literals.	
	B-tree index with indexed columns specified in a LIKE predicate that specifies a user information acquisition function as the pattern character string. Escape characters must be specified as literals.	"C1" LIKE CURRENT_USER
		"C1" LIKE CURRENT_USER ESCAPE '\'
	B-tree index with indexed columns specified in a LIKE predicate in which a dynamic parameter is used as the pattern character string. Escape characters must be specified as literals.	"C1" LIKE ?
		"C1" LIKE ? ESCAPE '\'
	B-tree index with indexed columns specified in a LIKE predicate that specifies a dynamic parameter as an escape character. A literal, user information acquisition function, or dynamic parameter must be specified as the pattern character string.	"C1" LIKE 'AB\%C%' ESCAPE ?
		"C1" LIKE CURRENT_USER ESCAPE ?
		"C1" LIKE ? ESCAPE ?
6	Text index with indexed columns specified in a CONTAINS scalar function that includes a word-context search specification (in which a notation-correction-search specification or a synonym-search specification is not included)#2, #8	CONTAINS("C1",'WORDCONTEXT("ABC")')>0
7	Text index with indexed columns specified in a CONTAINS scalar function that satisfies either of the following conditions#2, #4, #8: • A word-context search specification and notation-correction-search specification are included. • A word-context search specification and synonym-search specification are included.	CONTAINS("C1",'WORDCONTEXT(IGNORECASE("ABC"))')>0
		CONTAINS("C1",'WORDCONTEXT(SORTCODE("ABC"))')>0
		CONTAINS("C1",'WORDCONTEXT(SYNONYM("DIC1","ABC"))')>0
8	Text index with indexed columns specified in a LIKE predicate#1	"C1" LIKE 'ABC'
		"C1" LIKE 'ABC%'
		"C1" LIKE '%ABC%'
		"C1" LIKE '%ABC'
		"C1" LIKE ?
	Text index with indexed columns specified in a CONTAINS scalar function (in which a notation-correction-search specification, synonym-search specification, or word-context search specification is not included)#2	CONTAINS("C1",'"ABC"')>0
9	Text index that satisfies all of the following conditions#1, #3: • OR logical operators are used to specify multiple conditions that specify LIKE predicates. • An indexed column is specified as the match value of the LIKE predicate.	"C1" LIKE 'ABC' OR "C1" LIKE 'DEF'
		"C1" LIKE 'ABC%' OR "C1" LIKE 'DEF%'
		"C1" LIKE '%ABC%' OR "C1" LIKE '%DEF%'
		"C1" LIKE ? OR "C1" LIKE ?
	Text index with indexed columns specified in a CONTAINS scalar function that includes either of the following specifications#2, #4: • Notation-correction-search specification (only when a word-context search specification is not included) • Synonym-search specification (only when a word-context search specification is not included)	CONTAINS("C1",'IGNORECASE("ABC")')>0
		CONTAINS("C1",'SORTCODE("ABC")')>0
		CONTAINS("C1",'SYNONYM("DIC1","ABC")')>0
	Text index that satisfies all of the following conditions#1, #2, #3, #4:	"C1" LIKE '%ABC%' OR

Priority	Condition for index	Example of this priority (C1 is the column for which the index has been defined)
	• At least one each of a condition that specifies a LIKE predicate and a condition that specifies the CONTAINS scalar function are specified using the OR logical operator. • An indexed column is specified as the match value of the LIKE predicate. • An indexed column is specified in the CONTAINS scalar function.	`CONTAINS("C1",'"DEF"')>0`

		`"C1" LIKE '%ABC%'`
		`OR`
		`CONTAINS("C1",'"DEF"')>0`
		`OR`
		`CONTAINS("C1",'IGNORECASE("GHI")')>0`

	Text index that satisfies all of the following conditions[#2, #3, #4]:
	• Two or more of a condition that specifies the CONTAINS scalar function are specified using the OR logical operator.
	• An indexed column is specified in the CONTAINS scalar function.
		`CONTAINS("C1",'"ABC"')>0`
		`OR`
		`CONTAINS("C1",'"DEF"')>0`

		`CONTAINS("C1",'"ABC"')>0`
		`OR`
		`CONTAINS("C1",'IGNORECASE("ABC")')>0`
		`OR`
		`CONTAINS("C1",'SORTCODE("ABC")')>0`

		`CONTAINS("C1",'"ABC"')>0`
		`OR`
		`CONTAINS("C1",'IGNORECASE("DEF")')>0`
		`OR`
		`CONTAINS("C1",'SORTCODE("GHI")')>0`
		`OR`
		`CONTAINS("C1",'SYNONYM("DIC1","JKL")')>0`

		`CONTAINS("C1",'WORDCONTEXT("ABC")')>0`
		`OR`
		`CONTAINS("C1",'WORDCONTEXT("DEF")')>0`

10	Text index with indexed columns specified in a match value of a LIKE_REGEX predicate[#4, #5]	`"C1" LIKE_REGEX '^ABC'`
		`"C1" LIKE_REGEX '^ABC' FLAG IGNORECASE`

11	Text index that satisfies both of the following conditions[#3, #4, #5]:
	• Two or more conditions that specify LIKE_REGEX predicates are specified using an OR logical operator.
	• An indexed column is specified in the match values of the LIKE_REGEX predicates.
		`"C1" LIKE_REGEX '^ABC'`
		`OR`
		`"C1" LIKE_REGEX '^DEF'`

	Text index that meets all of the following conditions[#1, #3, #4, #5]:
	• At least one each of a condition that specifies a LIKE predicate and a condition that specifies a LIKE_REGEX predicate are specified using an OR logical operator.
	• An indexed column is specified as a match value of the LIKE predicate.
	• An indexed column is specified as a match value of a LIKE_REGEX predicate.
		`"C1" LIKE '%ABC%'`
		`OR`
		`"C1" LIKE_REGEX '^DEF'`

	Text index that meets all of the following conditions[#2, #3, #4, #5]:
	• At least one each of a condition that specifies a LIKE_REGEX predicate and a condition that specifies a CONTAINS scalar function are specified using an OR logical operator.
		`"C1" LIKE_REGEX '^ABC'`
		`OR`
		`CONTAINS("C1",'"XYZ"')>0`

Priority	Condition for index	Example of this priority (C1 is the column for which the index has been defined)
	• An indexed column is specified in a match value of a LIKE_REGEX predicate. • An indexed column is specified in a CONTAINS scalar function.	
	Text index that meets all of the following conditions[#1, #2, #3, #4, #5]: • Multiple conditions that specify a LIKE predicate, a LIKE_REGEX predicate, and a CONTAINS scalar function are specified using OR logical operators. • An indexed column is specified as the match value of a LIKE predicate. • An indexed column is specified in a match value of a LIKE_REGEX predicate. • An indexed column is specified in a CONTAINS scalar function.	"C1" LIKE_REGEX '^ABC' OR "C1" LIKE '%DEF%' OR CONTAINS("C1",'"XYZ"')>0
12	B-tree index with indexed columns specified in an IN predicate whose comparison values are value specifications only	"C1" IN (10,20,30)
	B-tree index with indexed columns specified in an IN predicate whose comparison values are only value specifications that include scalar operations	"C1" IN (10,20,30+?)
		"T1"."C1" IN (CASE WHEN 100=? THEN 10 ELSE 20 END,30,40)
13	B-tree index with indexed columns specified in a BETWEEN predicate	"C1" BETWEEN 20 AND 40
		"C1" BETWEEN 20 AND 40+?
		"T1"."C1" BETWEEN "T2"."C1"-6 MONTH AND "T2"."C1"
	B-tree index with indexed columns specified in a range condition that combines two comparison predicates	"C1">=20 AND "C1"<=40
14	B-tree index with indexed columns specified in an IN predicate that satisfies the following condition: • A subquery is specified whose comparison values do not include an external reference column.	"C1" IN (SELECT "C1" FROM "T2")
	B-tree index with indexed columns specified in an =ANY condition that specifies a subquery that does not include an external reference column	"C1"=ANY(SELECT "C1" FROM "T2")
	B-tree index with indexed columns specified in a =SOME condition that specifies a subquery that does not include an external reference column	"C1"=SOME(SELECT "C1" FROM "T2")
15	B-tree index with indexed columns specified in a >, >=, <, or <= condition	"C1">50
		"C1"<=200
		"C1">=50+?
		"T1"."C1" <"T2"."C1"||'X'
16	B-tree index with indexed columns specified in an IN predicate whose comparison values include a column specification	"T1"."C1" IN (10,"T2"."C1")
		"T1"."C1" IN (10,"T2"."C1"+?,50)
17	B-tree index with indexed columns specified in a condition that specifies an OR logical operator[#6]	"C1"<20 OR "C1">40
18	B-tree index with indexed columns specified in a LIKE predicate that satisfies both of the following conditions[#7]: • A literal is specified in the pattern character string.	"C1" LIKE '%BCD%'
		"C1" LIKE '%B_CD%' ESCAPE '\'

Priority	Condition for index	Example of this priority (C1 is the column for which the index has been defined)
	• The search is not a leading-match search.	
	B-tree index with indexed columns specified in a LIKE predicate that satisfies the following condition: • The pattern character string specifies a value expression that contains a column specification.	`"T1"."C1" LIKE "T2"."C2" || '%'`
	B-tree index with indexed columns specified in a LIKE predicate that satisfies the following condition: • The pattern character string specifies a value expression that contains a scalar operation.	`"C1" LIKE CURRENT_USER || '%'`
	B-tree index with indexed columns specified in a LIKE predicate that satisfies the following condition: • A value expression that specifies a column specification is specified in an escape character.	`"T1"."C1" LIKE 'A\%B@_C%' ESCAPE "T2"."C1"`
	B-tree index with indexed columns specified in a LIKE predicate that satisfies the following condition: • A value expression that includes a scalar operation is specified in an escape character.	`"C1" LIKE 'A\%B@_C%' ESCAPE CASE WHEN 10=? THEN '\' ELSE '@' END`

Notes

- For a multiple-column index, the priority shown in this table is applied sequentially starting from the first B-tree indexed column.
- Basically, the index to be used is determined by the priority shown above. However, depending on the specified search conditions, the priority shown above might not result in an effective evaluation. In this case, a index that does not follow the above priority might be used. If you want to check the index that was actually used for a retrieval, see 5.2.5 How to check the index used during execution of an SQL statement.

#1

This condition does not apply when the LIKE predicate specifies a pattern character string in the following formats:

- The pattern character string consists only of the special character %.
  Example: `"C1" LIKE '%'`
- The pattern character string is an empty string.
  Example: `"C1" LIKE ''`
- The pattern character string does not specify two or more consecutive non-special characters.
  Example: `"C1" LIKE '%A%', "C1" LIKE '%A%B%'`
- The pattern character string specifies a column in the same table as the column in the match value.
  Example: `"T1"."C1" LIKE "T1"."C2"`

#2

This condition does not apply when the CONTAINS scalar function specifies a search character string in the following formats. It does apply if a synonym-search specification is not specified, or one is specified but its synonym does not exist in relation to the search character string.

- The search character string is an empty string.
  Example 1: `CONTAINS("C1",'""')>0`
  Example 2: `CONTAINS("C1",'IGNORECASE("")')>0`
  Example 3: `CONTAINS("C1",'SORTCODE("")')>0`

Example 4: `CONTAINS("C1",'WORDCONTEXT("")')>0`

Example 5: `CONTAINS("C1",'WORDCONTEXT_PREFIX("")')>0`

- The search character string consists of a single character.

  Example 1: `CONTAINS("C1",'"A"')>0`

  Example 2: `CONTAINS("C1",'IGNORECASE("A")')>0`

  Example 3: `CONTAINS("C1",'SORTCODE("A")')>0`

  Example 4: `CONTAINS("C1",'WORDCONTEXT("A")')>0`

  Example 5: `CONTAINS("C1",'WORDCONTEXT_PREFIX("A")')>0`

  This condition also does not apply when a synonym-search specification is specified in a `CONTAINS` scalar function and its synonym exists in relation to the search character string, if the synonym is a single character such as A.

- A word-context search specification is included in a `CONTAINS` scalar function, and the number of characters in the search character string after elimination of the following symbols is no more than 1:
  - Half-width space (`0x20`), tab (`0x09`), line break (`0x0A`), return (`0x0D`), period, question mark, exclamation mark, and other single-byte symbols (`0x21` to `0x2F`, `0x3A` to `0x40`, `0x5B` to `0x60`, and `0x7B` to `0x7E`)

  Example 1: `CONTAINS("C1",'WORDCONTEXT("###A")')>0`

  Example 2: `CONTAINS("C1",'WORDCONTEXT_PREFIX("###A")')>0`

#3

Every condition included in the scope of the `OR` logical operator must be one that uses a `LIKE` predicate, a `LIKE_REGEX` predicate, or a `CONTAINS` scalar function. All columns in the `OR` logical operator must be indexed columns of the text index subject to selection.

#4

In either of the following circumstances, only text indexes defined with `CORRECTIONRULE` (notation-correction-search text-index specification) specified in a `CREATE INDEX` statement are subject to selection.

- A `LIKE_REGEX` predicate is specified that specifies `IGNORECASE` (or `I`) for `FLAG`.
- A `CONTAINS` scalar function is specified that specifies a notation-correction-search specification.

#5

This condition does not apply when the regular expression character string of the `LIKE_REGEX` predicate consists of 1 or fewer characters.

#6

All the columns included in the `OR` logical operator must be indexed columns of the B-tree index subject to selection. The index priority order might change depending on the number of predicates and the format of the condition specified in the `OR` logical operator.

#7

This condition does not apply if the data type of an indexed column for which a `LIKE` condition is specified is a variable-length character string data type and a pattern character string in one of the following formats:

- The special character `%` (percent sign) is not specified at the end.

  Example 1: `"C1" LIKE '%BCD'`

  Example 2: `"C1" LIKE '%BCD_'`

- The special character `%` (percent sign) specified at the end is preceded by a single-byte space or the special character _ (underscore).

  Example 1: `"C1" LIKE '%BCDΔ%'`

Example 2: `"C1" LIKE '%BCD_%'`

Legend: Δ: Half-width space

#8

Only a text index for a word-context search can be selected as the index to be used during a search.

(2) Selection rules for indexes

The index priority described in (1) Index priority is not the only factor that determines which index is used during execution of an SQL statement. The conditions described here are just one of the factors that determine which index is selected.

(a) Selection rules for B-tree indexes

The following table describes the selection rules for B-tree indexes.

Note that selection rules are applied sequentially in ascending order. That is, if selection rule No. 1 does not determine which index to use, selection rule No. 2 is applied next.

Table 5-2: Selection rules for B-tree indexes

Selection rule	Method of condition specification
1	An = condition is specified for all columns of a unique index.
2	The index with a = join condition contained in consecutive = conditions at the beginning of a search condition is prioritized.
3	If a search condition is specified for the first B-tree indexed column of the indexes, selection is based on the priority of that search condition.
4	The index with more search conditions is prioritized.
5	The index with more key conditions is prioritized.
6	The index with fewer indexed columns is prioritized. For indexes that satisfy selection rule 1, the index with more indexed columns is prioritized.
7	An index that uses a condition specified previously in an SQL statement as a search condition is prioritized.
8	An index that does not create a work table is prioritized.
9	A unique index is prioritized. For indexes for which no search condition is specified, the non-unique index is prioritized.
10	The index with the shorter key length is prioritized.
11	If none of selection rules from 1 to 10 is applied, the B-tree index is selected depending on internal processing.

For details about search conditions and key conditions, see 5.4.1 Evaluation method when B-tree indexes are used.

(b) Selection rules for text indexes

The following table describes the selection rules for text indexes.

Note that selection rules are applied sequentially in ascending order. That is, if selection rule No. 1 does not determine the index to use, selection rule No. 2 is tested next.

Table 5-3: Selection rules for text indexes

Selection rule	Method of condition specification
1	• A text index that can evaluate more conditions that specify `LIKE` predicates is prioritized.

Selection rule	Method of condition specification
	• A text index that can evaluate more conditions that specify LIKE_REGEX predicates is prioritized. • A text index that can evaluate more conditions that specify the scalar function CONTAINS is prioritized.
2	• A text index with the longer pattern character string for LIKE predicates is prioritized. • A text index with the longer regular expression character string for LIKE_REGEX predicates is prioritized. • A text index with the longer search character string for the scalar function CONTAINS is prioritized.
3	A text index with the shorter indexed column definition is prioritized.
4	• A text index that evaluates fewer LIKE predicates, LIKE_REGEX predicates, and CONTAINS scalar functions specified in OR conditions is prioritized. • If CONTAINS scalar functions that specify synonym-search specification are specified, the number of synonyms for the search string is added to the number of such scalar functions and compared.
5	If none of selection rules 1 to 4 result in an index being selected, internal processing determines the text index that is selected.

(c) Notes

- If HADB determines that it cannot use the index effectively based on cost information or other resources, these selection rules might be set aside in favor of a different approach.
- If HADB determines that it cannot use the index effectively for such reasons as only not predicates being specified in the search conditions for a WHERE clause or the ON search condition for a joined table, the index might not be used.

5.2.2 Examples of indexes that are used during retrieval of a table

This section describes examples of indexes that are used during retrieval of a table.

(1) Example 1 (B-tree index(single-column indexes))

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
       ON "T1" ("C1")
     IN "DBAREA01"
     EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1"=100

SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 AND "C2"='U0100'
```

When the SELECT statement shown above is executed, B-tree index IDX_C1 is used.

■ When a B-tree index is not used

A B-tree index is not used in the following case.

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
       ON "T1" ("C1")
     IN "DBAREA01"
     EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 OR "C2"='U0100'
```

With an OR condition, if a B-tree index is defined only for column C1 (or only for column C2), that B-tree index is not used. Therefore, when the SELECT statement shown above is executed, B-tree index IDX_C1 is not used.

If B-tree indexes are defined for both columns C1 and C2, those B-tree indexes are not used.

(2) Example 2 (B-tree index (multiple-column indexes))

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C2C1"
      ON "T1" ("C2","C1")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1"=100

SELECT "C1","C2","C3" FROM "T1"
    WHERE "C2"='U0100'

SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 AND "C2"='U0100'

SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 OR "C2"='U0100'
```

If the SELECT statement shown above is executed, B-tree index IDX_C2C1 is used.

(1) and (2) above are typical examples of cases in which a B-tree index is used and in which a B-tree index is not used. Even in the case above where the B-tree index is used, the B-tree index might not actually be used depending on the format of the specified search condition. For details about how to determine the B-tree index that was actually used during retrieval, see 3.2.5 How to check the index that was used for retrieval.

(3) Example 3 (text index)

Example of text index definition:

```
CREATE INDEX "IDX_TXT_C1"
       ON "T1" ("C1")
     IN "DBAREA01"
     EMPTY
     INDEXTYPE TEXT
```

Example of SELECT statement to execute:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1" LIKE '%XYZ%'

SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1" LIKE '%XYZ%' AND "C2"='U0100'
```

When you execute this SELECT statement, HADB uses the text index IDX_TXT_C1.

- **Scenario in which a text index is not used**

  In the following scenario, the text index will not be used.

  Example of text index definition:

```
CREATE INDEX "IDX_TXT_C1"
       ON "T1" ("C1")
     IN "DBAREA01"
     EMPTY
     INDEXTYPE TEXT
```

  Example of SELECT statement to execute:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1" LIKE '%XYZ%' OR "C2"='U0100'
```

  In the case of an OR condition, if a text index is defined only for column C1 (or only for column C2), then that text index will not be used. Consequently, when you execute this SELECT statement, HADB does not use the text index IDX_TXT_C1.

The examples described in (1) to (3) are representative examples of situations in which indexes are used and not used. In any of these examples in which the index is used, the index might not be used if the search condition is written in a certain way. If you want to check the index that will actually be used for a retrieval, see 5.2.5 How to check the index used during execution of an SQL statement.

5.2.3 Examples of indexes that are used during retrieval of a table (examples of index priority)

If multiple indexes are defined for a table, the index to be used is determined on the basis of the priority shown in Table 5-1: Index priority when multiple indexes are defined for a table.

This section explains typical examples of the priority of indexes that are used during retrieval of a table.

If you want to check the index that will actually be used for a retrieval, see 5.2.5 How to check the index used during execution of an SQL statement.

(1) Example 1 (priority between single-column indexes)

The following is an example of the relative priority between B-tree indexes (single-column indexes):

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
      ON "T1" ("C1")
    IN "DBAREA01"
    EMPTY

CREATE INDEX "IDX_C2"
```

```
      ON "T1" ("C2")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 AND "C2"='U0100'
```

In this example, the B-tree index IDX_C1 has a priority of 15, and the B-tree index IDX_C2 has a priority of 2. As such, the B-tree index IDX_C2 is used for retrieval.

Important

In the example above, if only one of the B-tree indexes is selected to be used on the basis of the priority, change the search condition, if possible, in such a manner that the B-tree index that can narrow the search most effectively will be used. You can expect an improvement in performance by using a B-tree index that provides more effective narrowing.

(2) Example 2 (priority between single-column indexes)

The following is an example of the relative priority between B-tree indexes (single-column indexes):

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
      ON "T1" ("C1")
    IN "DBAREA01"
    EMPTY

CREATE INDEX "IDX_C2"
      ON "T1" ("C2")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT * FROM "T1"
    WHERE "C2"='U0100' AND "C1"=100
```

In this example, B-tree indexes IDX_C1 and IDX_C2 both have a priority of 2. Because a B-tree index for the first column specified in the search condition has the higher priority, B-tree index IDX_C2 is used for retrieval.

(3) Example 3 (priority between multiple-column indexes)

The following is an example of the relative priority between B-tree indexes (multiple-column indexes):

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1C2"
      ON "T1" ("C1","C2")
    IN "DBAREA01"
    EMPTY
```

```
CREATE INDEX "IDX_C2C3"
      ON "T1" ("C2","C3")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C2"='U0100'
```

In this example, B-tree index `IDX_C2C3` is used.

`IDX_C1C2` has column `C2` as its second indexed column, while `IDX_C2C3` has column `C2` as its first indexed column, so `IDX_C2C3` is used.

(4) Example 4 (priority between single-column index and multiple-column index)

The following is an example of the relative priority between single-column and multiple-column B-tree indexes:

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
      ON "T1" ("C1")
    IN "DBAREA01"
    EMPTY

CREATE INDEX "IDX_C3C2"
      ON "T1" ("C3","C2")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 AND "C2"='U0100'
```

In this example, B-tree index `IDX_C1` is used.

`IDX_C3C2` has column `C2` as its second indexed column, while `IDX_C1` has column `C1` as its indexed column, so `IDX_C1` is used.

(5) Example 5 (priority between single-column index and multiple-column index)

The following is an example of the relative priority between single-column and multiple-column B-tree indexes:

Example of the B-tree index definition:

```
CREATE INDEX "IDX_C1"
      ON "T1" ("C1")
    IN "DBAREA01"
    EMPTY

CREATE INDEX "IDX_C2C3"
```

```
      ON "T1" ("C2","C3")
    IN "DBAREA01"
    EMPTY
```

Example of the SELECT statement to be executed:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1">100 AND "C2"='U0100'
```

In this example, the B-tree index IDX_C1 has a priority of 15, and the B-tree index IDX_C2C3 has a priority of 2. As such, the B-tree index IDX_C2C3 is used.

IDX_C1 that has column C1 as an indexed column and IDX_C2C3 that has column C2 as the first indexed column are subject to comparison to determine priority, as shown in Table 5-1: Index priority when multiple indexes are defined for a table.

(6) Example 6 (priority between text index and B-tree index)

The following is an example of the relative priority of a text index and B-tree index (single-column index).

Example of index definitions:

```
CREATE INDEX "IDX_TXT_C1"      <- Definition of text index
      ON "T1" ("C1")
    IN "DBAREA01"
    EMPTY
    INDEXTYPE TEXT

CREATE INDEX "IDX_C2"          <- Definition of B-tree index
      ON "T1" ("C2")
    IN "DBAREA01"
    EMPTY
```

Example of SELECT statement to execute:

```
SELECT "C1","C2","C3" FROM "T1"
    WHERE "C1" LIKE '%XYZ%' AND "C2" LIKE 'ABC%'
```

In this example, the text index IDX_TXT_C1 has a priority of 8, and the B-tree index IDX_C2 has a priority of 4. As such, the B-tree index IDX_C2 is used for retrieval.

Important

In this scenario, if the relative priority results in only one of the indexes being used, if possible, change the search condition so that the index that can narrow the search scope most effectively is used. You can expect to see an improvement in performance by using an index that more efficiently narrows the search scope.

(7) If you want to change the index to be used during retrieval

This subsection explains how to change the index to be used during retrieval.

(a) If there are multiple indexed columns with the same priority value, specify the one that you want to use first

As explained in (2) Example 2 (priority between single-column indexes), a B-tree index for the first column specified in the search condition is used if multiple indexes with the same priority value are specified. You can use this rule to change the B-tree index to be used. The following example changes the search condition specification:

Before change:

```
SELECT * FROM "T1"
    WHERE "C3"='A001' AND "C2"='U0100'
```

After change:

```
SELECT * FROM "T1"
    WHERE "C2"='U0100' AND "C3"='A001'
```

Before the change, B-tree index IDX_C3 defined for column C3 would be used, but after change, B-tree index IDX_C2 defined for column C2 is used.

(b) To lower the priority value of the current index that is used

You can lower the priority value of the index that would be used, so that another index will be used. The following example changes the search condition specification:

Before change:

```
SELECT * FROM "T1"
    WHERE "C1"=100 AND "C2" LIKE 'ABC%'
```

After change:

```
SELECT * FROM "T1"
    WHERE "C1" BETWEEN 100 AND 100 AND "C2" LIKE 'ABC%'
```

Before the change, the priority of IDX_C1 was 2 and the priority of IDX_C2 was 4, but after the change, the priority of IDX_C1 is 13. Therefore, B-tree index IDX_C1 would have been used before the change, but B-tree index IDX_C2 will be used after the change.

(c) Using the index specification

You can specify the index to be used for retrieval. The following shows an example.

Before change:

```
SELECT * FROM "T1"
    WHERE "C1"=100 AND "C2" LIKE 'ABC%'
```

The priority of B-tree index IDX_C1 defined for column C1 is 2 and the priority of B-tree index IDX_C2 defined for column C2 is 4. Therefore, B-tree index IDX_C1 is used.

After change:

```
SELECT * FROM "T1" /*>> WITH INDEX (IDX_C2) <<*/
    WHERE "C1"=100 AND "C2" LIKE 'ABC%'
```

You can specify the index to be used for retrieval by using the index specification indicated by the underlining. When the SELECT statement shown here is executed, index IDX_C2 will be used.

For details about the index specification, see *Index specification* in the manual *HADB SQL Reference*.

5.2.4 Cases where an index is not used

If any of the following search conditions is specified, no index is used. In the examples below, C1 and C2 are the names of table columns.

- Negative conditions are specified.

  As shown in the example below, if a index that has column C1 as its indexed column is defined, but negative conditions are specified as the search conditions, that index is not used.

  Example:

```
WHERE "C1"<>100
WHERE "C1" IS NOT NULL
WHERE "C1" NOT LIKE 'ABC%'
WHERE "C1" NOT IN (10,20,30)
WHERE "C1" NOT BETWEEN 20 AND 40
```

- Logical operator NOT is specified.

  Example:

```
WHERE NOT ("C1"=100)
```

- Conditions containing scalar operations, such as arithmetic operations and CASE expressions, are specified.

  As shown in the example below, if an index that has column C1 as its indexed column is defined, but the specified search condition contains a scalar operation, such as arithmetic operations or CASE expressions, that index is not used.

  Example:

```
WHERE C1*10=200
```

  When a scalar operation is specified in a certain way, its search condition is automatically subjected to equivalent exchange to allow the use of indexes. For details about equivalent exchange for scalar operations, see 5.11.4 Equivalent exchange for scalar operations.

  If only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

- An IN subquery in which is specified a subquery containing an external reference column is specified.

  Example:

```
WHERE "T1"."C1" IN (SELECT "C1" FROM "T2" WHERE "C2"="T1"."C2")
```

- An =ANY quantified predicate in which is specified a subquery containing an external reference column is specified.

  Example:

```
WHERE "T1"."C1"=ANY(SELECT "C1" FROM "T2" WHERE "C2"="T1"."C2")
```

- An =SOME quantified predicate in which is specified a subquery containing an external reference column is specified.

  Example:

```
WHERE "T1"."C1"=SOME(SELECT "C1" FROM "T2" WHERE "C2"="T1"."C2")
```

- A quantified predicate other than =ANY or =SOME is specified.

  Example:

```
WHERE "C1"<>ANY(SELECT "C1" FROM "T2")
WHERE "C1"<>SOME(SELECT "C1" FROM "T2")
WHERE "C1"=ALL(SELECT "C1" FROM "T2")
```

- The EXISTS predicate is specified.

  Example:

```
WHERE EXISTS(SELECT * FROM "T2")
```

- The pattern character string of a LIKE predicate is specified as follows:
  - The pattern character string consists only of the special character %.

    Example:

```
WHERE "C1" LIKE '%'
```

  - The pattern character string is an empty string.

    Example:

```
WHERE "C1" LIKE ''
```

  - The pattern character string does not contain two or more consecutive non-special characters.

    Example:

```
WHERE "C1" LIKE '%A%'
WHERE "C1" LIKE '%A%B%'
```

  - The pattern character string specifies a column in the same table as the column in the match value.

    Example:

```
WHERE "T1"."C1" LIKE "T1"."C2"
```

  When search conditions are specified as in the preceding examples, even if a text index is defined that has the C1 column as an indexed column, that index will not be used.

5.2.5 How to check the index used during execution of an SQL statement

You can check which index was used during execution of an SQL statement by checking the access path. For details about access paths, see the following sections:

- How to check access paths

  See 6.1.2 How to check access paths.
- Information displayed in access paths

  See (b) Information related to indexes in (1) Information related to table retrieval methods and indexes in 6.1.5 Information displayed in the details view.

By checking access paths, you can find out whether indexes were used as intended.

If an index is defined for a target table, it is used for search processing according to the specified search condition. However, depending on the specified search condition, the index might not be used or a different index might be used unexpectedly. If the results obtained by the above check indicate that the index was not used as intended, the specified search condition or index definition might not be appropriate. In such a case, consider changing the index definition or the search condition.

If the index definition needs to be changed, the application program developer must request that the HADB system designer or system administrator change the index definition.

5.2.6 Notes on searching using a text index

If you use the logical operator OR to specify multiple LIKE predicates, LIKE_REGEX predicates, or CONTAINS scalar functions that specify text indexed columns, the search is conducted using the search strings from all the conditions.

Example: Suppose that C1 is a text indexed column.

```
SELECT * FROM "T1" WHERE "C1" LIKE 'ABCDEFG%'
                      OR "C1" LIKE_REGEX 'XYZ[0-9]+'
```

When this SELECT statement is executed, the search strings (ABCDEFG% and XYZ[0-9]+) specified in the conditions of the LIKE predicate and the LIKE_REGEX predicate are used in the search processing.

If the total number of characters in the pattern character strings specified in the LIKE predicates, the regular expressions in LIKE_REGEX predicates, and the search conditions in CONTAINS scalar functions exceeds 1,000 characters, search processing might slow down significantly. If the total number of characters is 1,001 or more, consider using one of the following approaches:

- Change the SQL statement, for example by using a UNION to join the search results.
- Use index specification or other means to conduct a search that avoids the use of a text index.

For details about the UNION statement, see *Query expression* in the manual *HADB SQL Reference*. For details about index specification, see *Index specification* in the manual *HADB SQL Reference*.

5.3 Range indexes used during execution of SQL statements

This section explains how to determine the range indexes are used during execution of SQL statements, and how to check the range index used during execution of an SQL statement.

5.3.1 Conditions under which range indexes are used during execution of an SQL statement

If range indexes are defined for a table, whether the range indexes are actually used during execution of an SQL statement is determined by the search conditions specified in the `WHERE` clause, the `ON` search conditions for joined tables, and the B-tree indexes or text indexes used during execution of an SQL statement.

This subsection explains the conditions under which range indexes are used for skipping chunks, and the conditions under which they are used for skipping segments.

Notes

- The conditions explained here are those, used by range indexes, that are applied to query expressions after expansion of internal derived tables or applied to search conditions converted by an equivalent exchange of search conditions. For details about expanding internal derived tables, see *Internal derived tables* in the manual *HADB SQL Reference*. For details about the equivalent exchange of search conditions, see 5.11 Equivalent exchange of search conditions.
- If only literals are specified in a scalar operation in the value expression specified in the search conditions, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.
- When joining tables, range indexes might not be used when evaluating the join condition. This depends on the method used to join the table. For details about table joining methods, see 5.5 Table joining methods.
- If subqueries are specified, depending on how they are processed, the range index might not be used. For details about how to process subqueries, see 5.6 How to process subqueries.

(1) Range indexes used for skipping chunks

If all the conditions shown in the following table are satisfied, a range index is used to skip chunks during execution of an SQL statement.

Table 5-4: Conditions under which range indexes are used

No.	Conditions under which range indexes are used	Example
1	A range-indexed column must be specified in one of the predicates listed below in a search condition specified in the `WHERE` clause or in an `ON` search condition for joined tables.	--
	• Comparison predicate Note that no range index is used if the same table's columns are specified in both the left-hand and the right-hand terms of a comparison operator.	Example 1 Example 2 Example 3 Example 14 Example 15
	• `BETWEEN` predicate Note that no range index is used if the same table's columns are specified in value expression 1 and value expression 2 or 3.	Example 4 Example 5

No.	Conditions under which range indexes are used	Example
	• IN predicate (value expression) Note that no range index is used if the same table's columns are specified in value expression 1 and value expression 2 or any subsequent value expression.	Example 6 Example 7
	• IN predicate (table subquery) The range index is used in only a case where hash execution is applied as the subquery processing method.	Example 17
	• LIKE predicate If LIKE is specified, one of the following conditions must be satisfied. Note that if ESCAPE is specified in the LIKE predicate, the escape characters must be specified in a literal or a dynamic parameter. • Only a user information acquisition function is specified in the pattern character string. • Only a dynamic parameter is specified in the pattern character string. • Only a literal that begins with 'character-string%' (% is a special character) is specified in the pattern character string. • Only a literal that begins with 'character-string_' (_ is a special character) is specified in the pattern character string. • Only a literal that does not contain a special character (% or _) is specified in the pattern character string (complete match). If NOT LIKE is specified, one of the following conditions must be satisfied: • Only a dynamic parameter is specified in the pattern character string. • Only a literal that begins with 'character-string%' (% is a special character) is specified in the pattern character string.	Example 8 Example 9
	• Quantified predicate The range index is used in only a case where hash execution is applied as the subquery processing method.	Example 17
2	If a predicate explained in 1 is specified in the condition using the logical operator OR or NOT, no range index is used.	Example 10
3	If a scalar operation is used in a predicate explained in 1, no range index is used.	Example 11
4	If columns from three or more different tables are specified in a predicate explained in 1, no range index is used.	Example 12
5	• A predicate explained in 1 is specified in a subquery that contains an external reference column and nested loop execution is applied as the subquery processing method. However, if a predicate explained in 1 contains external reference columns, range indexes whose indexed columns are those external reference columns are not used. • A predicate explained in 1 is specified in a subquery that contains an external reference column and hash execution is applied as the subquery processing method. The range index is used in only a case where a predicate explained in 1 contains an external reference column in *external-reference-column=column-specification* format or *column-specification=external-reference-column* format.	Example 13 Example 16

If multiple range indexes that satisfy all the conditions in Table 5-4: Conditions under which range indexes are used are defined for the table, all range indexes that satisfy all conditions are used.

To determine if a range index can be used for skipping chunks, see *Checking a range index (whether it can skip chunks)* in the *HADB Setup and Operation Guide*.

(2) Range indexes used for skipping segments

If all the conditions shown in Table 5-4: Conditions under which range indexes are used are satisfied, a range index is used to skip segments during execution of an SQL statement. If multiple range indexes that satisfy all the conditions in Table 5-4: Conditions under which range indexes are used are defined for the table, all range indexes that satisfy all conditions are used.

However, if a B-tree index or text index is used during execution of an SQL statement, segments are not skipped by using a range index.

5.3.2 Examples of range indexes used during retrieval

(1) Example 1

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1">10
```

Explanation:

In this example, range index `RIDX1` is used to skip chunks and segments that do not contain data that satisfies the conditions.

(2) Example 2

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
CREATE INDEX "RIDX2" ON "T1"("C2") IN "DBAREA01" EMPTY INDEXTYPE RANGE
CREATE INDEX "RIDX3" ON "T1"("C3") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1"=10 AND "C2">20 AND "C3">30
```

Explanation:

In this example, range indexes `RIDX1`, `RIDX2`, and `RIDX3` are used to skip chunks and segments that do not contain data that satisfies the conditions.

(3) Example 3 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1">"C2"
```

Explanation:

When columns from the same table are specified in the right-hand and left-hand terms of a comparison operator, a range index is not used (chunks and segments that do not contain data that satisfies the conditions are not skipped). In this example, range index `RIDX1` is not used because columns `C1` and `C2` both belong to table `T1`.

(4) Example 4

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" BETWEEN 10 AND 30
```

Explanation:

In this example, range index RIDX1 is used to skip chunks and segments that do not contain data that satisfies the conditions.

(5) Example 5 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" BETWEEN "C2" AND "C3"
```

Explanation:

When columns from the same table are specified in value expression 1 and value expression 2 or 3, a range index is not used (chunks and segments that do not contain data that satisfies the conditions are not skipped). In this example, range index RIDX1 is not used because columns C1, C2, and C3 all belong to table T1.

(6) Example 6

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" IN (10,20,30)
```

Explanation:

In this example, range index RIDX1 is used to skip chunks and segments that do not contain data that satisfies the conditions.

(7) Example 7 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" IN ("C2",20,30)
```

Explanation:

When columns from the same table are specified in value expression 1 and value expression 2 or any subsequent value expression, a range index is not used (chunks and segments that do not contain data that satisfies the conditions

are not skipped). In this example, range index RIDX1 is not used because columns C1 and C2 both belong to table T1.

(8) Example 8

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" LIKE CURRENT_USER
SELECT * FROM "T1" WHERE "C1" LIKE ?
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC%'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC%E'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC%E%'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC%E%G'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC_'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC_E'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC_E_'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC_E_G'
SELECT * FROM "T1" WHERE "C1" LIKE 'ABC'
SELECT * FROM "T1" WHERE "C1" NOT LIKE ?
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC%'
SELECT * FROM "T1" WHERE "C1" LIKE 'AB\_C%' ESCAPE '\'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'AB\_C%' ESCAPE '\'
```

Explanation:

In this example, range index RIDX1 is used to skip chunks and segments that do not contain data that satisfies the conditions when any of these SELECT statements is executed.

If a dynamic parameter is specified in the LIKE predicate, the range index is used. However, if the pattern character string that does not satisfy the following conditions is specified, there is no benefit to using range indexes:

- LIKE
  - A pattern character string that starts with 'character-string%' (% is a special character)
  - A pattern character string that starts with 'character-string_' (_ is a special character)
  - A pattern character string that does not contain special character % or _ (complete match)
- NOT LIKE
  - The pattern character string 'character-string%' (% is a special character)

(9) Example 9 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" LIKE '%ABC%'
SELECT * FROM "T1" WHERE "C1" LIKE '%ABC'
SELECT * FROM "T1" WHERE "C1" LIKE '_ABC_'
SELECT * FROM "T1" WHERE "C1" LIKE '_ABC'
SELECT * FROM "T1" WHERE "C1" NOT LIKE CURRENT_USER
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC%E'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC%E%'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC%E%G'
SELECT * FROM "T1" WHERE "C1" NOT LIKE '%ABC%'
```

```
SELECT * FROM "T1" WHERE "C1" NOT LIKE '%ABC'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC_'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC_E'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC_E_'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC_E_G'
SELECT * FROM "T1" WHERE "C1" NOT LIKE '_ABC_'
SELECT * FROM "T1" WHERE "C1" NOT LIKE '_ABC'
SELECT * FROM "T1" WHERE "C1" NOT LIKE 'ABC'
SELECT * FROM "T1" WHERE "C1" LIKE '%AB\_C%' ESCAPE '\'
SELECT * FROM "T1" WHERE "C1" LIKE 'A\%B@_C%'
                      ESCAPE CASE WHEN 10=? THEN '\' ELSE '@' END
SELECT * FROM "T1" WHERE "C1" NOT LIKE '%AB\_C%' ESCAPE '\'
```

Explanation:

None of the SELECT statements shown above satisfies the condition for *LIKE predicate* in No. 1 in Table 5-4: Conditions under which range indexes are used. Therefore, range index RIDX1 is not used (chunks and segments that do not contain data that satisfies the conditions are not skipped).

(10) Example 10 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1"=10 OR "C2"=20
SELECT * FROM "T1" WHERE NOT("C1"=10)
```

Explanation:

When the predicate specified in the condition uses the logical operator OR or NOT, a range index is not used (chunks and segments that do not contain data that satisfies the conditions are not skipped).

(11) Example 11 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "T1"."C1"+10=20
```

Explanation:

When a scalar operation containing a range-indexed column is used, no range index is used (chunks and segments that do not contain data that satisfies the conditions are not skipped).

(12) Example 12 (range indexes are not used)

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1","T2","T3" WHERE "T1"."C1" BETWEEN "T2"."C2" AND "T3"."C3"
SELECT * FROM "T1","T2","T3" WHERE "T1"."C1" IN ("T2"."C2","T3"."C3")
```

Explanation:

When columns from three or more different tables are specified in the predicates, a range index is not used (chunks and segments that do not contain data that satisfies the conditions are not skipped).

(13) Example 13

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1"("C2") IN "DBAREA01" EMPTY INDEXTYPE RANGE
CREATE INDEX "RIDX2" ON "T1"("C3") IN "DBAREA01" EMPTY INDEXTYPE RANGE
CREATE INDEX "RIDX3" ON "T2"("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" "X"
    WHERE "X"."C1"=ANY(SELECT "T2"."C1" FROM "T2"
                       WHERE "X"."C2"=10 AND "X"."C3"="T2"."C1")
```

Explanation:

Range indexes `RIDX1` and `RIDX2` will not be used because their indexed columns (columns `C2` and `C3` of table `T1`) are specified as external reference columns.

Range index `RIDX3` will be used to skip chunks and segments that do not contain data that satisfies the conditions because its indexed column (column `C1` of table `T2`) is not specified as an external reference column.

(14) Example 14

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T2" ("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1","T2" WHERE "T1"."C1"<"T2"."C1"
SELECT * FROM "T1" INNER JOIN "T2" ON "T1"."C1"<"T2"."C1"
```

Explanation:

If a nested loop join that uses table `T1` as an outer table and table `T2` as an inner table is applied, range index `RIDX1` is used to skip chunks and segments. For details about a nested loop join, see 5.5.1 About nested-loop join.

(15) Example 15

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T2" ("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1","T2" WHERE "T1"."C1"="T2"."C1"
SELECT * FROM "T1" INNER JOIN "T2" ON "T1"."C1"="T2"."C1"
```

Explanation:

If a hash join that uses table `T1` as an outer table and table `T2` as an inner table is applied as the table joining method, range index `RIDX1` is used to skip chunks and segments. For details about a hash join, see 5.5.2 About hash join.

(16) Example 16

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1" ("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" "X"
    WHERE EXISTS(SELECT * FROM "T2" WHERE "X"."C1"="T2"."C1")
```

Explanation:

If hash execution is applied as the processing method to a subquery that contains an external reference column, range index `RIDX1` is used to skip chunks and segments. For details about hash execution as the processing method of a subquery that contains an external reference column, see (3) Hash execution in 5.6.3 Methods for processing subqueries that contain an external reference column.

(17) Example 17

Definition of range indexes

```
CREATE INDEX "RIDX1" ON "T1" ("C1") IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

Example SQL statement

```
SELECT * FROM "T1" WHERE "C1" IN (SELECT "T2"."C1" FROM "T2")
SELECT * FROM "T1" WHERE "C1"=ANY(SELECT "T2"."C1" FROM "T2")
```

Explanation:

If hash execution is applied as the processing method to a subquery that does not contain an external reference column, range index `RIDX1` is used to skip chunks and segments. For details about hash execution as the processing method of a subquery that does not contain an external reference column, see (4) Hash execution in 5.6.1 Methods for processing subqueries that do not contain an external reference column.

5.3.3 How to check the range index used during execution of an SQL statement

By checking the access path, you can check the range index used during execution of an SQL statement. For details about access paths, see the following:

- How to check access paths

  See 6.1.2 How to check access paths.
- Information displayed in access paths

  See (b) Information related to indexes in (1) Information related to table retrieval methods and indexes in 6.1.5 Information displayed in the details view.

5.4 How to evaluate the search conditions when indexes are used

There are two ways to evaluate the search conditions when indexes are used:

- Evaluation method when B-tree indexes are used
- Evaluation method when range indexes are used

This section explains how to evaluate the search conditions specified in the `WHERE` clause or the `ON` condition when indexes are used.

5.4.1 Evaluation method when B-tree indexes are used

When B-tree indexes are used, the search conditions are evaluated according to the range search condition and the key condition.

(1) About the range search condition

A condition for specifying a search range when B-tree indexes are used is called a range search condition. This includes mainly the following conditions:

- =, not-equal sign, `IS NULL`, `LIKE` predicate leading match (literal specification), `LIKE` predicate dynamic parameter specification, `IN` predicate#, `BETWEEN` predicate, quantified predicate (`=ANY`, `=SOME`)

  #

  An `IN` predicate (that is not a table subquery specification) specified for an inner table of a nested loop join might not be a search condition.

The following figure shows an example of the evaluation method based on a range search condition.

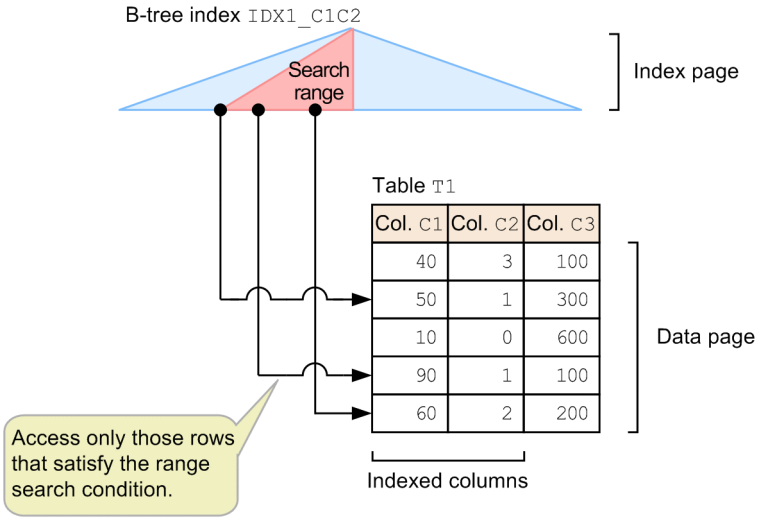
Figure 5-4: Example of evaluation method based on a range search condition

■ Table and B-tree index definitions

```
CREATE TABLE "T1"
            ("C1" INTEGER,
             "C2" INTEGER,
             "C3" INTEGER) IN "DBAREA01"

CREATE INDEX "IDX_C1C2" ON "T1"("C1","C2")
              IN "DBAREA01" EMPTY
```

■ Executed SQL statement

```
SELECT * FROM "T1" WHERE "C1" BETWEEN 50 AND 100
```



Explanation:

In this example, the range search condition is $50 \leq C1 \leq 100$. HADB accesses the data pages that contain the rows that satisfy the range search condition.

(2) About the key condition

A condition that can be evaluated only by B-tree-indexed columns is called a key condition. A key condition cannot be used to narrow down the search range when B-tree indexes are used, as a range search condition can do. However, a key condition can reduce the number of times a data page is referenced because the key condition can be evaluated by using only B-tree index pages, thereby improving search performance.

The following figure shows an example of the evaluation method based on a key condition.

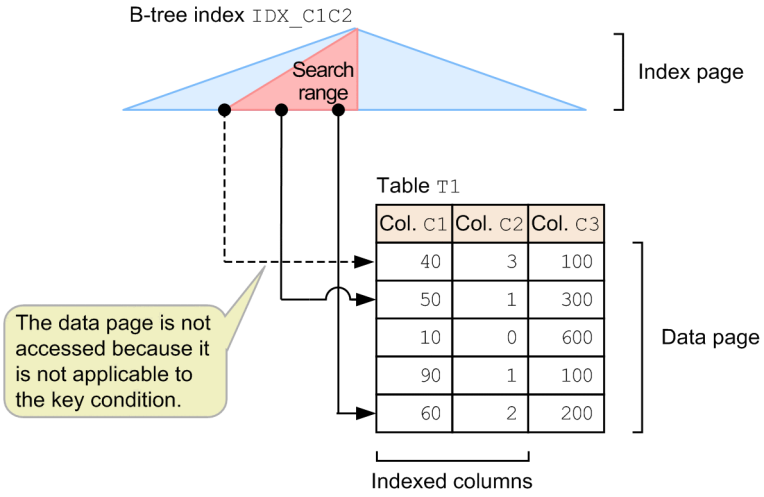
Figure 5-5: Example of evaluation method based on a key condition

■ Table and B-tree index definitions

```
CREATE TABLE "T1"
            ("C1" INTEGER,
             "C2" INTEGER,
             "C3" INTEGER) IN "DBAREA01"

CREATE INDEX "IDX_C1C2" ON "T1"("C1","C2")
              IN "DBAREA01" EMPTY
```

■ Executed SQL statement

```
SELECT * FROM "T1"
    WHERE "C1" BETWEEN 40 AND 60
      AND "C2" IN(1,2)
```



Explanation:

In this example, the conditions are as follows:

- Range search condition: $40 \leq C1 \leq 60$
- Key condition: `C2 IN(1,2)`

HADB uses the range search condition to narrow down the search range and accesses only those data pages that contain the rows that satisfy the key condition.

5.4.2 Evaluation method when range indexes are used

When range indexes are used, the search conditions are evaluated according to the following two range index conditions:

- Condition in which range indexes are used to skip chunks

  This condition uses range indexes to skip chunks that do not contain data that satisfies the search condition. Because this condition reduces the number of chunks to be accessed, the amount of data pages to be referenced is reduced and search performance improves.

  This condition is used only for range indexes that can be used to skip chunks.

  To determine whether a range index can be used to skip chunks, see *When checking whether the index is a range index that can skip chunks* in *Searching a dictionary table* in the *HADB Setup and Operation Guide*.

- Condition in which range indexes are used to skip segments

This condition uses range indexes to skip segments that do not contain data that satisfies the search condition. Because this condition reduces the number of segments to be accessed, the amount of data pages to be referenced is reduced and search performance improves.

> **Note**
>
> For details about skipping chunks and segments, see *Range indexes* in the *HADB Setup and Operation Guide*.

(1) Example of skipping chunks by using range indexes

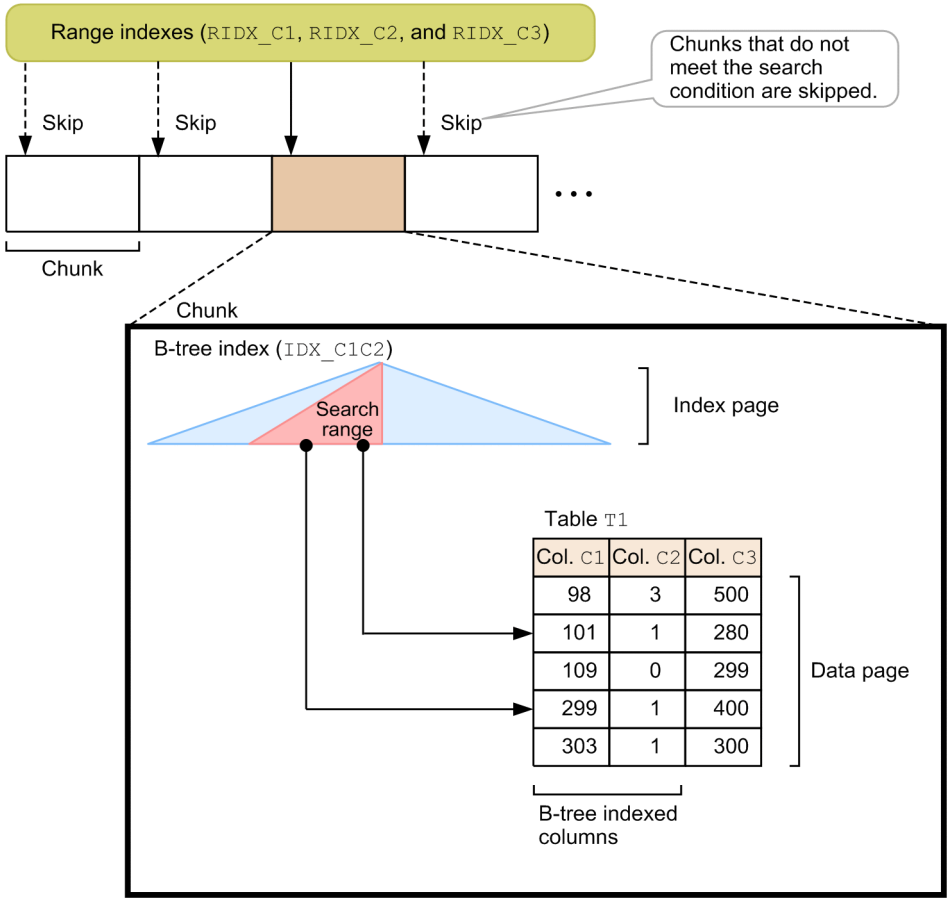
The following provides an example of skipping chunks by using range indexes.

■ Table, B-tree index, and range index definitions

```
CREATE TABLE "T1"
            ("C1" INTEGER,
             "C2" INTEGER,
             "C3" INTEGER) IN "DBAREA01" CHUNK

CREATE INDEX "IDX_C1C2" ON "T1"("C1","C2")          <-- B-tree index definition
              IN "DBAREA01" EMPTY

CREATE INDEX "RIDX_C1" ON "T1"("C1")                |
              IN "DBAREA01" EMPTY INDEXTYPE RANGE   |
                                                    |
CREATE INDEX "RIDX_C2" ON "T1"("C2")                | Range index definitions
              IN "DBAREA01" EMPTY INDEXTYPE RANGE   |
                                                    |
CREATE INDEX "RIDX_C3" ON "T1"("C3")                |
              IN "DBAREA01" EMPTY INDEXTYPE RANGE   |
```

■ Executed SQL statement

```
SELECT * FROM "T1"
    WHERE "C1" BETWEEN 100 AND 300
      AND "C2" IN(1,2)
      AND "C3"<300
```



Explanation:

The following describes the procedure for retrieval processing.

1. Before accessing chunks, HADB evaluates the range index condition used to skip chunks.
   - Range index condition used to skip chunks: `100≤C1≤300, C2 IN(1, 2), C3<300`
2. HADB evaluates the search condition and key condition based on the B-tree indexes of the search target chunks.
   - Search condition: `100≤C1≤300`
   - Key condition: `C2 IN(1,2)`
3. HADB accesses the data page and evaluates the remaining search condition.
   - Search condition to be evaluated after the data page is accessed: `C3<300`

Note that range indexes defined in columns `C1`, `C2`, and `C3` are not used to skip a segment.

(2) Example of skipping segments by using range indexes

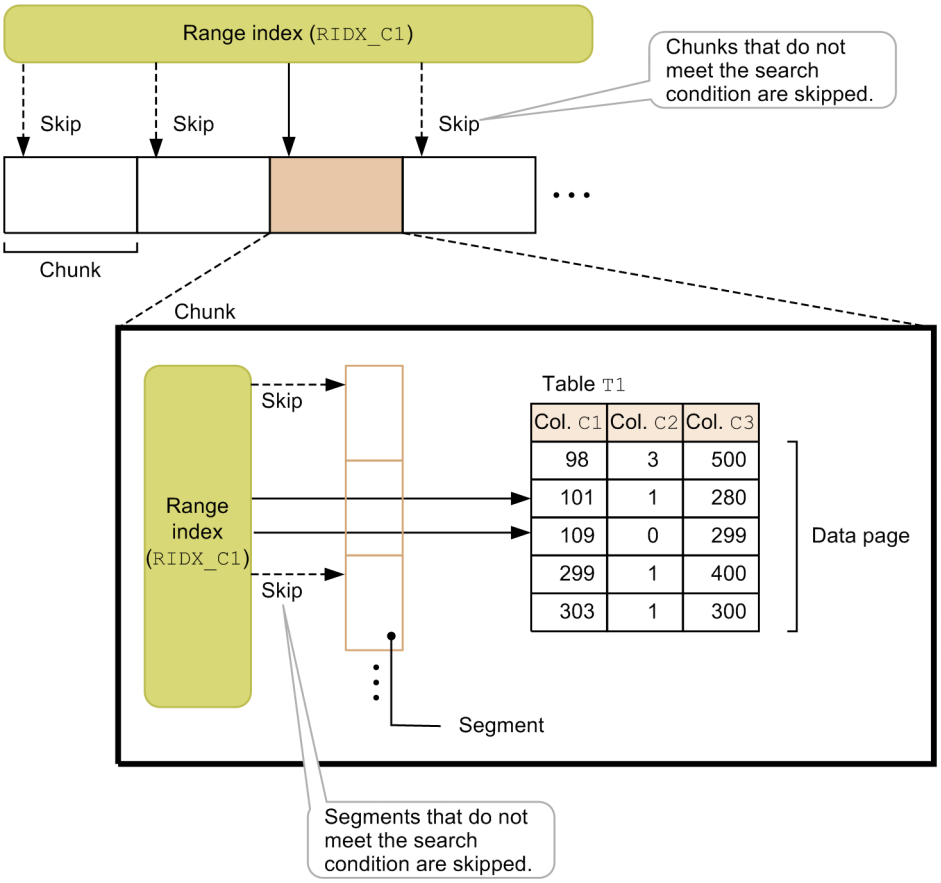
The following provides an example of skipping segments by using range indexes.

■ Table and range index definitions

```
CREATE TABLE "T1"
            ("C1" INTEGER,
             "C2" INTEGER,
             "C3" INTEGER) IN "DBAREA01" CHUNK

CREATE INDEX "RIDX_C1" ON "T1"("C1")      ← Range index definition
              IN "DBAREA01" EMPTY INDEXTYPE RANGE
```

■ Executed SQL statement

```
SELECT * FROM "T1"
    WHERE "C1" BETWEEN 100 AND 110
      AND "C2" IN(1,2)
```



Explanation:

The following describes the procedure for retrieval processing.

1. Before accessing chunks, HADB evaluates the range index condition used to skip chunks.
   - Range index condition used to skip chunks: `100≤C1≤110`
2. HADB evaluates the range index condition used to skip segments in the search target chunk.
   - Range index condition used to skip segments: `100≤C1≤110`
3. HADB accesses the data page and evaluates the search condition.
   - Search condition to be evaluated after the data page is accessed: `C2 IN(1,2)`

5.5 Table joining methods

There are two table joining methods:

- Nested loop join
- Hash join

This section explains these two joining methods and their characteristics.

HADB automatically determines the table joining method to use. You can find out which join method was used by checking the access path after the SQL statement was executed. For details about access paths, see the following subsections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information that is displayed as access paths
  See (27) Table joining methods in 6.1.4 Information displayed in the tree view.

> **Note**
>
> In the case of a joined table, you can specify the join method in a join method specification. For details about join method specifications, see *Specification format and rules for join method specifications* in the manual *HADB SQL Reference*.

5.5.1 About nested-loop join

HADB joins tables by repeating as many times as there are rows in the outer table a matching process that involves using the value in the joined column in the outer table as the basis for searching the joined column in the inner table. This joining method is called *nested loop join*.

If an index is defined for the column that is specified in the join condition for joining the outer and inner tables, the index is used when evaluating the join condition. This narrows the search range of the inner table.

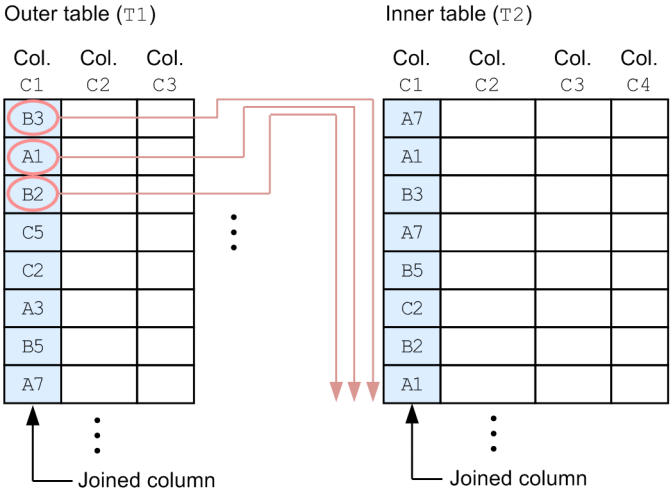
The following explains the table joining method using a nested loop join by way of an example in which a nested loop join is used when the `SELECT` statement shown below is executed.

Example:

```
SELECT * FROM "T1","T2" WHERE "T1"."C2">10 AND "T1"."C1">"T2"."C1"
```

HADB determines the table that is to be the outer table and the table that is to be the inner table. In this example, table `T1` is the outer table and table `T2` is the inner table. Columns `T1.C1` and `T2.C1` specified in the underlined join condition are the joined columns.

Figure 5-6: Table joining method using a nested loop join



Explanation:

HADB retrieves the value of joined column `B3` from the outer table and matches it with the value of the joined column in the inner table. Next, HADB retrieves the value of joined column `A1` from the outer table and matches it with the value of the joined column in the inner table. HADB repeats this processing as many times as there are rows in the outer table.

Consider the example of a nested loop join with table `T1` as the outer table and table `T2` as the inner table. If an index is defined for the columns specified in the join condition (`"T1"."C1"` and `"T2"."C1"`), the index might be used when evaluating the join condition (`"T1"."C1">"T2"."C1"`). This can narrow the search range of the inner table.

Note

In the case of a joined table, you can specify the outer table in a join method specification. For details about join method specifications, see *Specification format and rules for join method specifications* in the manual *HADB SQL Reference*.

5.5.2 About hash join

HADB joins tables by matching the hash table created based on the joined column of the outer table with the results of hashing the joined column of the inner table. This joining method is called *hash join*.

Indexes defined for columns specified in the join condition used to join the outer and inner tables will not be used when evaluating the join condition. The join condition will be evaluated using hashing. However, if a range index is defined for a column of the inner table that is specified in a join condition, if conditions allow, the range index might be used.

(1) Joining tables using the hash join method

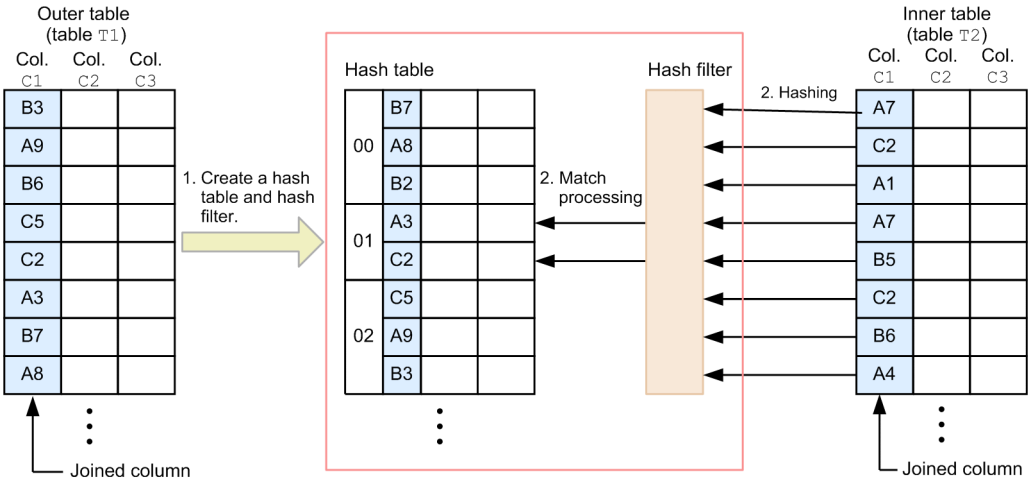
The following explains how tables are joined by a hash join method, by way of an example in which the hash join is applied when the following `SELECT` statement is executed.

Example:

```
SELECT * FROM "T1","T2" WHERE "T1"."C2">10 AND "T1"."C1"="T2"."C1"
```

HADB determines which table is to be the outer table and which table is to be the inner table. In this example, table T1 is the outer table and table T2 is the inner table. The columns T1.C1 and T2.C1 specified in the underlined join condition become the joined column.

Figure 5-7: Joining tables using the hash join method



Explanation:

1. HADB creates a hash table and hash filter based on the values in the joined column in the outer table (table T1).
2. Next, HADB matches the result of hashing the value in the joined column of the inner table (table T2) with the hash table, and joins the tables. Before matching the joined column in the inner table with the hash table, HADB performs filtering by using the hash filter. This reduces the number of times the joined column in the inner table is matched with the hash table.

For a hash join where table T1 is the outer table and table T2 is the inner table, even if an index is defined for the columns ("T1"."C1" and "T2"."C1") specified in the join condition, that index is not used when evaluating the join condition ("T1"."C1"="T2"."C1").

If an index is defined for "T1"."C2", that index might be used when evaluating "T1"."C2">10.

The range index might be used when a hash join is processed if both of the following two conditions are met:

- The range index is defined for a joined column of the inner table for a hash join (column C1 of table T2 in the preceding example).
- The conditions under which the range index can be used are met.

  For details about the conditions under which range indexes are used, see 5.3.1 Conditions under which range indexes are used during execution of an SQL statement.

If both of the preceding two conditions are met, when a hash table is created from a joined column of the outer table during hash join processing, the maximum and minimum values of that joined column are obtained. Then, when the inner table is searched, the range index is used to skip the inner table's chunks or segments that are not within the obtained maximum and minimum values of the joined column.

HADB creates the hash table in the hash table area. The size of the hash table area is specified in the adb_sql_exe_hashtbl_area_size operand in the server definition or the client definition.

A hash filter is created in the hash filter area. The size of the hash filter area is specified in the adb_sql_exe_hashflt_area_size operand in the server definition or the client definition.

(2) Example where hash join is applied

Example 1:

```
SELECT * FROM "T1","T2" WHERE "T1"."C1"="T2"."C1"
```

A hash join is applied when a single column specification is specified on each side of a join condition specified by a = operator.

Example 2:

```
SELECT * FROM "T1","T2" WHERE "T1"."C1"="T2"."C1"+10
```

A hash join is applied when the following condition is met:

- A single column is specified on one side of a join condition specified by a = operator, and the other side specifies a scalar operation that includes a column specification.

Example 3:

```
SELECT * FROM "T1" INNER JOIN "T2"
    ON "T1"."C1"=CAST("T2"."C1" AS INTEGER)
```

A hash join is applied when the following condition is met:

- A single column is specified on one side of a join condition specified by a = operator, and the other side specifies a scalar function that includes a column specification.

Example 4:

```
SELECT * FROM "T1" LEFT JOIN "T2"
    ON "T1"."C1"="T2"."C1"||"T2"."C2"
```

A hash join is applied when the following condition is met:

- A single column is specified on one side of a join condition specified by a = operator, and the other side specifies a concatenation operation that includes a column specification.

(3) Notes on applying hash joins

Try to make the data type and data length of the value expressions on the left and right of the = of the join condition the same if possible. When the data type and data length of the value expressions on the left and right of the = of the join condition differ, they are converted to the same data type and length before creating the hash table. Hashing takes place after this process has finished. This conversion process incurs an overhead.

For details about the data types after conversion, see *Data types that can be converted, assigned, and compared* in the manual *HADB SQL Reference*.

Note that if the data type after conversion is `DECIMAL`, precision and scaling are determined based on the following equations:

Equations

```
Precision = Pmax + Smax
Scaling = Smax
Pmax = MAX (p1-s1,p2-s2)
Smax = MAX (s1,s2)
```

p1, *s1*: The precision and scaling of the value expression specified on the left side of the join condition specified by =

p2, *s2*: The precision and scaling of the value expression specified on the right side of the join condition specified by =

Note that if the data type prior to conversion is INTEGER, these equations are calculated as DECIMAL(20,0). If the data type prior to conversion is SMALLINT, these equations are calculated as DECIMAL(10,0).

Example:

The following shows an example of determining precision and scaling when the data type after conversion is DECIMAL.

Table definitions

```
CREATE TABLE "T1"("C1" INTEGER,"C2" CHAR(3),"C3" DATE) IN "DBAREA01"
CREATE TABLE "T2"("C1" DECIMAL(7,3),"C2" CHAR(3),"C3" DATE) IN "DBAREA01"
```

Example of SQL statement

```
SELECT * FROM "T1","T2" WHERE "T1"."C1"="T2"."C1"
```

The column specified on the left side ("T1"."C1") of the underlined join condition specified by the = operator has a different data type and data length from the column specified on the right side ("T2"."C1"). This means that the data type and data length undergo conversion. The column "T1"."C1" is INTEGER type data, and the column "T2"."C1" is DECIMAL type data.

In this scenario, the data types of the columns specified on the left and right sides of the join condition specified by the = operator are converted to DECIMAL type data. Column "T1"."C1" is treated as if it were DECIMAL(20,0). The precision and scaling of the converted DECIMAL type data are calculated as follows:

```
Pmax = MAX(p1-s1,p2-s2) = MAX (20-0,7-3) = 20
Smax = MAX (s1,s2) = MAX (0,3) = 3
Precision = Pmax + Smax = 23
Scaling = Smax = 3
```

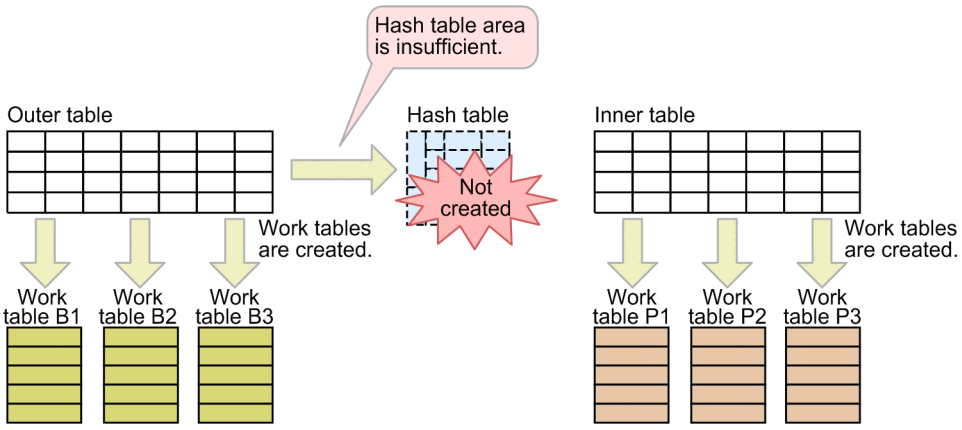
The data type and data length of the value expressions on the left and right of the = of the join condition are converted to DECIMAL(23,3), the hash table is created, and hashing takes place.

(4) Action to take when the hash table area has insufficient space

▪ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

1. If there is insufficient hash table area to create the hash table, the outer table data is stored in multiple work tables. Similarly, the inner table data is also stored in multiple work tables.

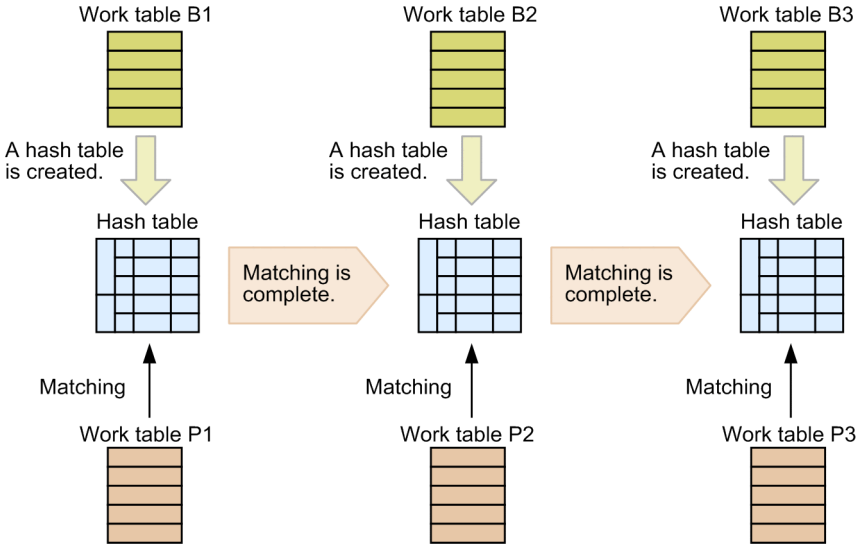


> **Note**
>
> In the preceding example, three work tables are created for the outer table, and three work tables are created for the inner table. The number of work tables that are created differs depending on conditions, such as the specification of the SQL statement.

2. A hash table is created with a work table for the outer table (work table `B1`), and then the hash table is matched with a work table for the inner table (work table `P1`).

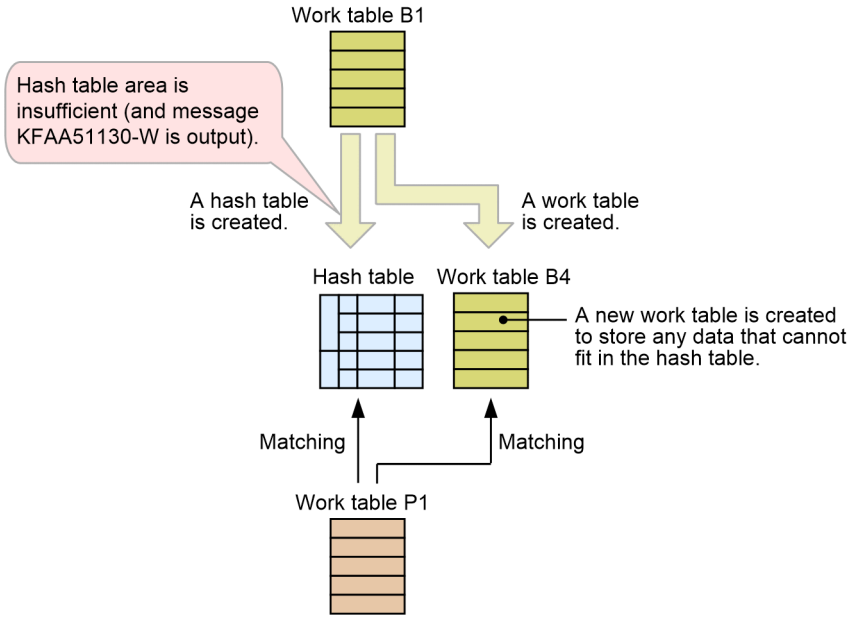
   After matching between the hash table and work table `P1` is complete, a hash table is created with work table `B2`, and the hash table is matched with work table `P2`.

   After matching between the hash table and work table `P2` is complete, a hash table is created with work table `B3`, and the hash table is matched with work table `P3`.



If the hash table area becomes insufficient during the processing in step 2

If the hash table area becomes insufficient during the processing in step 2, any data that cannot fit in the hash table is stored in another work table. In this case, in addition to matching between a work table for the inner table and a hash table, matching between a work table for the inner table and the newly created work table (work table `B4`) occurs.



Note

If a new work table (work table `B4`) is created due to insufficient hash table area during the processing in step 2, the `KFAA51130-W` message is output to the server message log file.

■ Action to take when the hash table area has insufficient space

If the hash table area is insufficient, processing time for the SQL statement might take longer due to work table creation and matching. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

(5) Conditions where a hash join is not applied

A hash join will not be applied as the join method when any of the following conditions are met:

- `0` is specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or client definition.
- `0` is specified for the `adb_sys_uthd_num` operand in the server definition.
- `0` is specified for the `adb_sql_exe_max_rthd_num` operand in the server definition or client definition.
- The query is specified in an update SQL statement.
- There is no join condition specified by = between the tables to be joined.
- There is a join condition specified by = between the tables to be joined, but it meets one of the following conditions:
  - The left or right of the join condition specified by = specifies a scalar operation that includes a column specification.
  - One side of the join condition specified by = specifies a scalar operation that includes a column specification, the other side specifies a single column specification, and any of the following conditions is met:

    - The scalar operation that includes a column specification specifies a subquery that includes an external reference column.

    - The scalar operation that includes a column specification specifies a column in the same table as the single column specification specified on the other side of the join condition specified by =.

- The post-conversion data type of the value expressions specified on the left and right of the join condition specified by = is DECIMAL, and the precision is greater than 38.
- All of the following conditions are met:

  - The data type of the value expression specified on either the left or right of the join condition specified by = is VARCHAR, and the data length of the value expression exceeds 32,000 bytes.

  - The data types or data lengths of the value expressions specified on the left and right of the join condition specified by = are different.
- The tables to be joined are specified in the subquery that contains external reference columns.

- There is a join condition specified by = between the tables to be joined, but one or other of the tables is a derived table derived by a table value constructor.
- The query specifies ROW in a selection expression.
- Nested loop join is specified as the join method in the join method specification.

> **Note**
>
> You can specify the outer table of a joined table by join method specification. For details about join method specification, see *Specification format and rules for join method specifications* in the manual *HADB SQL Reference*.

(6) Conditions where a hash filter is applied

1. A hash filter is applied during hash join when all the following conditions are satisfied:
   - 0 is not specified for the adb_sql_exe_hashflt_area_size operand in the server definition or client definition.
   - Tables are joined by a comma join or INNER JOIN specified.
2. If the size of the hash filter area specified for the adb_sql_exe_hashflt_area_size operand in the server definition or client definition is too small, a shortage might occur in the hash filter area allocated for each hash retrieval. As a result, a hash filter is not applied to any hash retrieval for which the size of the hash filter is insufficient. If you want to apply the hash filter to all types of hash retrieval, change the value of the adb_sql_exe_hashflt_area_size operand so that the following condition is satisfied:

```
value-specified-for-adb_sql_exe_hashflt_area_size>↑A×B×number-of-processing-real-threads-in-SQL-statement÷1024↑
```

A:

Number of hash filters to be used for the hash retrieval to which a hash filter was not applied

If there are two or more cases in which a hash filter was not applied to hash retrieval, determine the number of hash filters for each hash retrieval, and then assign the largest value among those values. The following shows the number of hash filters to be used for a hash retrieval process:

- For hash join: Number of = join conditions for hash join
- For subqueries to which hash execution is applied, and which do not contain an external reference: 1
- For subqueries to which hash execution is applied, and which contain external references: The number of = conditions that contains an external reference column

B:

Sum total of all the following specified in SQL statements

- Number of hash joins to which a hash filter is applied

- Number of subqueries to which hash execution using a hash filter is applied

5.5.3 Characteristics of the joining methods

The following table describes the characteristics of the two joining methods.

Table 5-5: Characteristics of the table joining methods

Joining method	Speed of the initial data retrieval operation	Advantage	Disadvantage
Nested loop join	Fast	Retrieval is faster when the joined column in the inner table can be narrowed down using a B-tree index or text index.	If there are many hit rows in the outer table, processing performance decreases.
Hash join	Slow	If there are few hit rows in the outer table and many hit rows in the inner table, high-speed retrieval can be achieved.	If there are many hit rows in the outer table, the hash table area might become large. If a shortage occurs in the hash table area, processing performance decreases because the data is first saved to a work table.

5.6 How to process subqueries

There are two types of subquery processing methods:

- Methods for processing subqueries that do not contain an external reference column
- Methods for processing subqueries that contain an external reference column

This section explains these subquery processing methods and their characteristics.

Note that you can check which processing method was used by viewing the access path after the SQL statement is executed. For details about access paths, see the following subsections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information that is displayed as access paths
  See (2) Subquery processing methods in 6.1.4 Information displayed in the tree view.

For details about external reference columns, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.

5.6.1 Methods for processing subqueries that do not contain an external reference column

There are four methods for processing subqueries that do not contain an external reference column:

- Work table execution
- Row value execution
- Work table row value execution
- Hash execution

These processing methods are explained below.

(1) Work table execution

Subquery processing might be performed with work table execution applied in the following cases:

- A quantified predicate is specified
- A table subquery is specified in the `IN` predicate

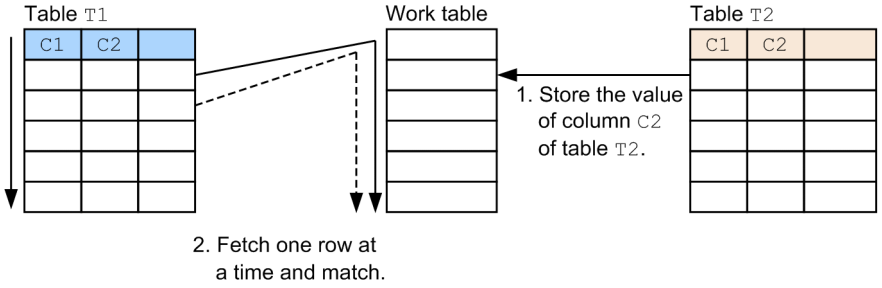
The following shows an example of work table execution.

■ `SELECT` statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE ABS("T1"."C2")=ANY(SELECT "T2"."C2" FROM "T2")
```

This example assumes that no B-tree index or text index is defined for column `C2` in table `T1`.

Figure 5-8: Processing method for work table execution



Explanation:

1. Stores the result of the subquery in the work table.

   This example searches table T2 specified in the subquery, and then stores the value of column C2 of table T2 in the work table.

2. Executes the query that is outside the subquery. Each time HADB retrieves one row of query outside the subquery, HADB matches that row with the result of the subquery (in the work table) and evaluates the search condition.

   This example retrieves one row of table T1 at a time, and then matches the absolute value of column C2 of table T1 with the value of column C2 of table T2 stored in the work table to evaluate the search condition.

(2) Row value execution

Subquery processing might be performed with row value execution applied in the following cases:

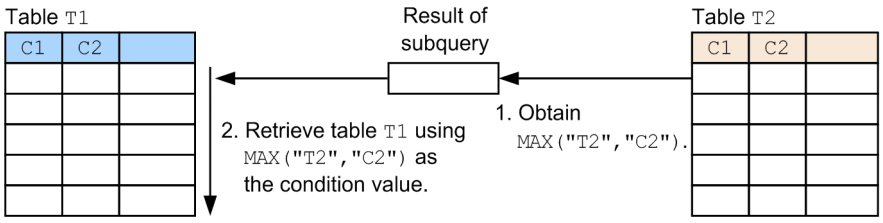
- A scalar subquery is specified.
- The EXISTS predicate is specified.

The following shows an example of row value execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE "T1"."C2"< (SELECT MAX("T2"."C2") FROM "T2")
```

Figure 5-9: Processing method for row value execution



Explanation:

1. Obtains the result of the subquery.

   This example searches table T2 specified in the subquery, and then obtains MAX("T2"."C2").

2. Uses the result of the subquery to evaluate the condition that contains a subquery of the query outside the subquery. For a comparison predicate, B-tree indexes or text indexes might be used to execute the query outside the subquery.

   This example retrieves table T1 using MAX("T2"."C2") obtained in 1 as the condition value. Depending on the condition, B-tree indexes or text indexes might be used for the retrieval.

(3) Work table row value execution

Subquery processing might be performed with work table row value execution applied in the following cases:

- A quantified predicate is specified
- A table subquery is specified in the `IN` predicate

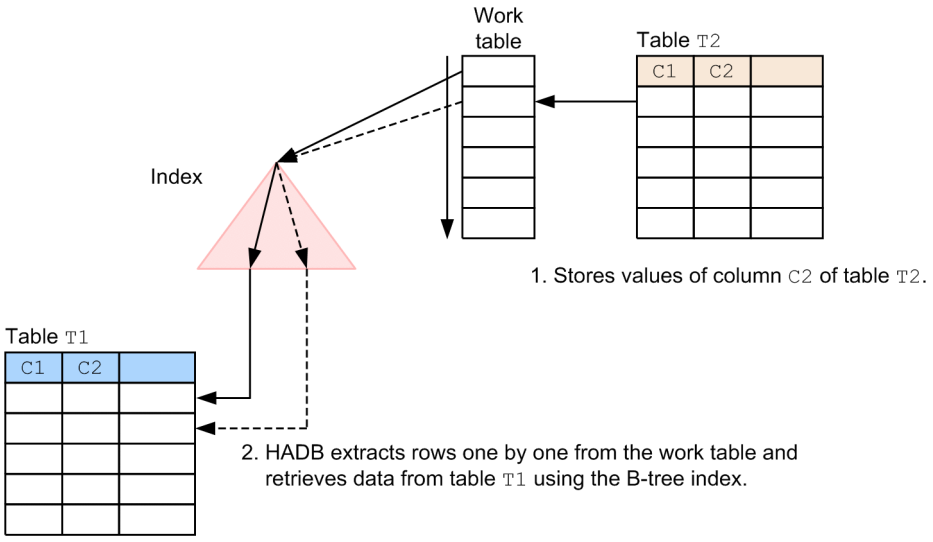
The following shows an example of work table row value execution.

■ `SELECT` statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE "T1"."C2"=ANY(SELECT "T2"."C2" FROM "T2")
```

This example assumes that B-tree indexes are defined for column `C2` of table `T1`.

Figure 5-10: Processing method for work table row value execution



Explanation:

1. Stores the result of the subquery in the work table.

   This example searches table `T2` specified in the subquery, and then stores the value of column `C2` of table `T2` in the work table.

2. Fetches one row value from the work table at a time and executes the query outside the subquery to evaluate the search condition. A B-tree index is used for this processing. A text index is used if one is defined.

   This example fetches the value of column `C2` of table `T2` from the work table one row at a time and retrieves table `T1` by using the B-tree index defined for the column `C2` of table `T1`.

(4) Hash execution

A method for processing subqueries by using a hash table is called *hash execution*. Hash execution might be applied in the following cases:

- A quantified predicate is specified
- A table subquery is specified in the `IN` predicate

If hash execution is applied during subquery processing, HADB first creates a hash table on the basis of the result of the subquery. Then, HADB executes the query outside the subquery, and then generates hash values from the value of

the column specified to the left of the quantified predicate or IN predicate. Finally, processing is performed to match the values with the hash table.

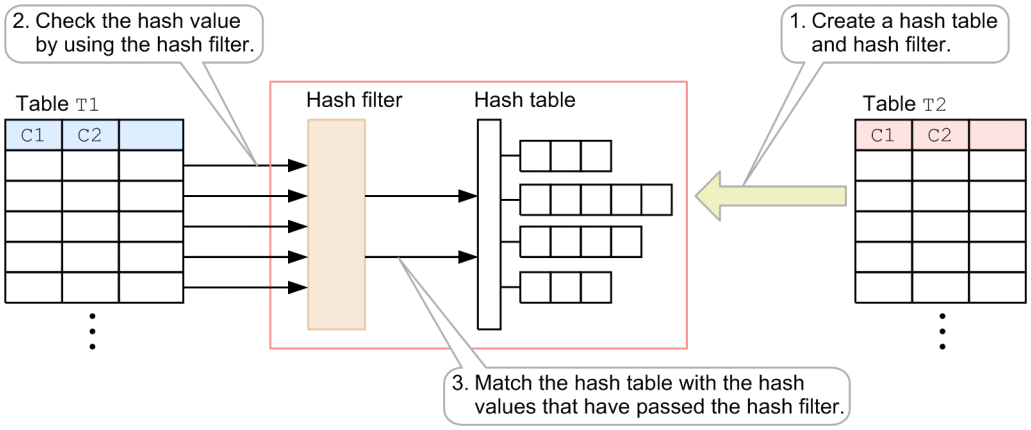
When a hash table is created, a hash filter is also created. HADB filters hash values by using the hash filter before matching the hash values with the hash table. This reduces the number of times hash values are matched with the hash table.

The following shows an example of hash execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE "T1"."C2"=ANY(SELECT "T2"."C2" FROM "T2")
```

Figure 5-11: Processing method for hash execution



Explanation:

1. Creates a hash table and hash filter on the basis of the result of the subquery (underlined portion in the example SQL statement). This example retrieves table T2 specified in the subquery, and then creates a hash table and hash filter from the value of column C2 of table T2.
2. Executes the query outside the subquery, and then generates a hash value from the value of the column specified to the left of the quantified predicate (in the example SQL statement, column C2 of table T1). The hash value is checked by using the hash filter. This example fetches one row from table T1 at a time, generates the hash value from the value of column C2 of table T1, and then checks the hash value by using the hash filter.
3. Performs processing to match the hash table with the hash values that have passed the hash filter.

The range index might be used when a hash execution is processed if both of the following two conditions are met:

- There is a table specified in a query outside a subquery (table T1 in the preceding example). In the table, the range index is defined for the column specified to the left of the quantified predicate or IN predicate (column T1.C2 in the preceding example).
- The conditions under which the range index can be used are met.

  For details about the conditions under which range indexes are used, see 5.3.1 Conditions under which range indexes are used during execution of an SQL statement.

If both of the preceding two conditions are met, when a hash table is created from the subquery result during processing of a hash execution, the maximum and minimum values of the subquery are obtained. The range index is then used when the table specified in a query outside the subquery (table T1 in the preceding example) is searched. The range index is used to skip the table's chunks or segments that are not within the obtained maximum and minimum values of the subquery result.

A hash table is created in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. Note that when `0` is specified in the `adb_sql_exe_hashtbl_area_size` operand, hash execution is not applied.

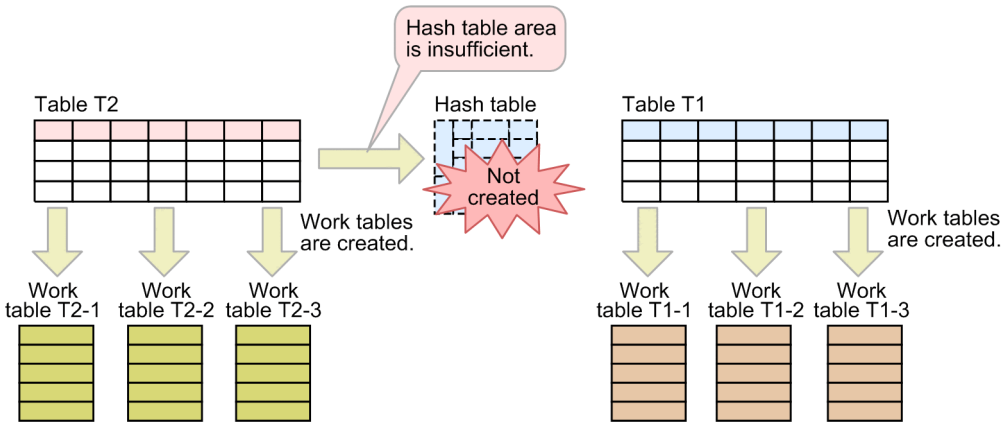
A hash filter is created in the hash filter area. The size of the hash filter area is specified in the `adb_sql_exe_hashflt_area_size` operand in the server definition or the client definition. If `0` is specified for the `adb_sql_exe_hashflt_area_size` operand, a hash filter is not applied during hash execution.

■ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

Note that the tables `T1` and `T2` in the following explanation correspond to tables `T1` and `T2` in Figure 5-11: Processing method for hash execution.

1. If there is insufficient hash table area to create the hash table, the data of table `T2` is stored in multiple work tables. Also, the data of table `T1` is stored in multiple work tables in the same way as for the data of table `T2`.

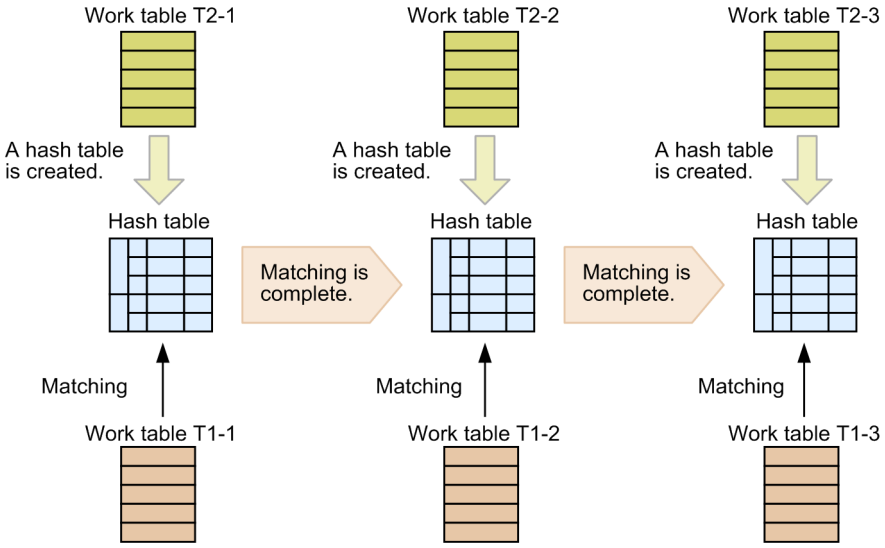


> **Note**
>
> In the preceding example, three work tables are created for table `T2`, and three work tables are created for table `T1`. The number of work tables that are created differs depending on conditions, such as the specification of the SQL statement.

2. A hash table is created with a work table for table `T2` (work table `T2-1`), and then the hash table is matched with a work table for table `T1` (work table `T1-1`).

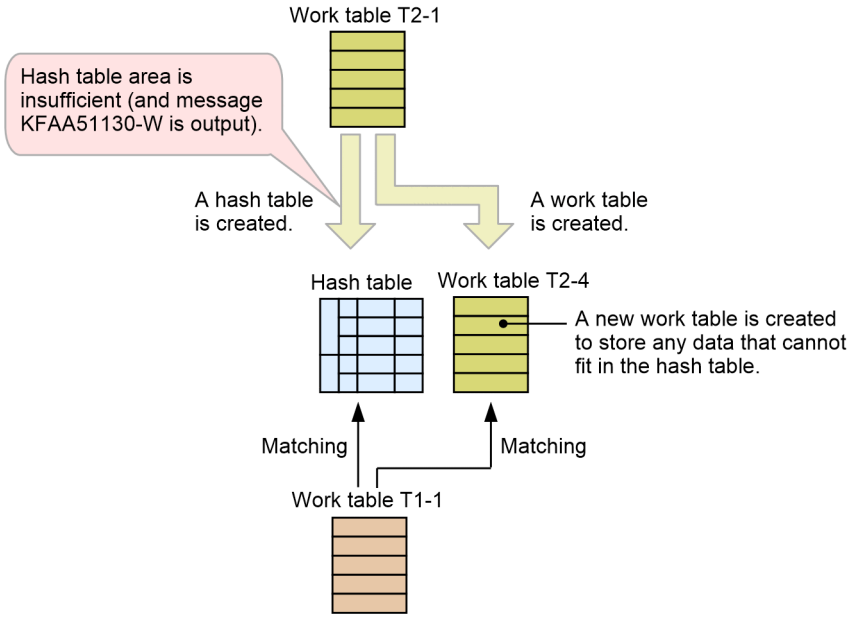
   After matching between the hash table and work table `T1-1` is complete, a hash table is created with work table `T2-2`, and the hash table is matched with work table `T1-2`.

   After matching between the hash table and work table `T1-2` is complete, a hash table is created with work table `T2-3`, and the hash table is matched with work table `T1-3`.



If the hash table area becomes insufficient during the processing in step 2

If the hash table area becomes insufficient during the processing in step 2, any data that cannot fit in the hash table is stored in another work table. In this case, in addition to matching between a work table for table `T1` and a hash table, matching between a work table for table `T1` and the newly created work table (work table `T2-4`) occurs.



> **Note**
>
> If a new work table (work table `T2-4`) is created due to insufficient hash table area during the processing in step 2, the `KFAA51130-W` message is output to the server message log file.

■ Action to take when the hash table area has insufficient space

If the hash table area is insufficient, processing time for the SQL statement might take longer due to work table creation and matching. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

■ **Action to take when the hash filter area has insufficient space**

If the size of the hash filter area specified for the `adb_sql_exe_hashflt_area_size` operand in the server definition or client definition is too small, a shortage might occur in the hash filter area allocated for each hash retrieval. As a result, a hash filter is not applied to any hash retrieval for which the size of the hash filter is insufficient. If you want to apply the hash filter to all types of hash retrieval, increase the value of the `adb_sql_exe_hashflt_area_size` operand. For details about the formula for estimating the value to be specified, see (6) Conditions where a hash filter is applied in 5.5.2 About hash join.

5.6.2 Characteristics of the methods for processing subqueries that do not contain an external reference column

The following table describes the characteristics of each method for processing subqueries that do not contain an external reference column.

Table 5-6: Methods for processing subqueries that do not contain an external reference column

No.	Processing method	Benefits	Disadvantages
1	Work table execution	This method can be applied to the conditions of all subqueries that require work tables.	Processing performance decreases when there are many queries outside the subquery.
2	Row value execution	B-tree indexes or text indexes can be used for queries outside the subquery. When there are many such queries, this allows data to be retrieved at a higher speed when B-tree indexes or text indexes are used to narrow the search range.	Processing performance decreases when there are many queries outside the subquery and the predicates containing the subquery cannot be narrowed down by using B-tree indexes or text indexes.
3	Work table row value execution	B-tree indexes or text indexes can be used for queries outside the subquery. When there are many such queries and the subquery hit count is low, this allows data to be retrieved at a higher speed when B-tree indexes or text indexes are used to narrow the search range.	Processing performance decreases when the subquery hit count is high, because B-tree indexes or text indexes are used for as many queries outside the subquery as there are rows resulting from the subquery.
4	Hash execution	Data can be retrieved at a higher speed if all data required for the processing can be stored in the hash table.	If a large amount of data must be stored in the hash table, the size of the hash table area becomes large. Processing performance decreases if a shortage occurs in the hash table area, because all data is first saved to a work table.

5.6.3 Methods for processing subqueries that contain an external reference column

There are three methods for processing subqueries that contain an external reference column:

- Nested loops work table execution
- Nested loops row value execution
- Hash execution

When using the nested loops row value execution method, cache area is sometimes created to store the results of the subquery. This is to reduce the number of times the subquery is executed.

These processing methods are explained below.

(1) Nested loops work table execution

Subquery processing might be performed with nested loops work table execution applied in the following cases:

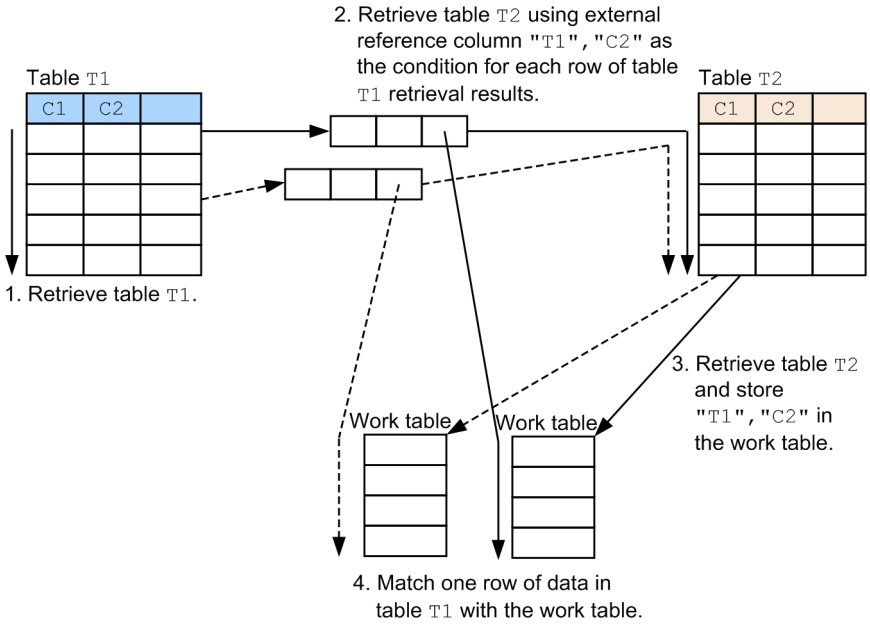
- A quantified predicate is specified
- A table subquery is specified in the `IN` predicate

The following shows an example of nested loops work table execution.

■ `SELECT` statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE "T1"."C1"=ANY(SELECT "T2"."C1" FROM "T2"
                            WHERE "T2"."C2"="T1"."C2")
```

Figure 5-12: Processing method for nested loops work table execution



Explanation:

1. Executes the query that is outside the subquery

   This example retrieves table `T1`.

2. Executes the subquery by using the value of an external reference column each time one row of query outside the subquery is fetched.

   This example retrieves table `T2` by using the value of an external reference column (`"T1"."C2"`) as the condition value for each row of table `T1` retrieval results.

3. Creates a work table based on the result of the executed subquery.

   This example retrieves table `T2` and stores the value of `"T2"."C1"` in the work table.

4. Uses the created work table to evaluate the condition that contains a subquery outside the subquery.

   This example evaluates the condition containing the subquery by matching with the value of corresponding `"T2"."C1"` in the work table for each row of table `T1` retrieval results.

(2) Nested loops row value execution

Subquery processing might be performed with nested loops row value execution applied in the following cases:

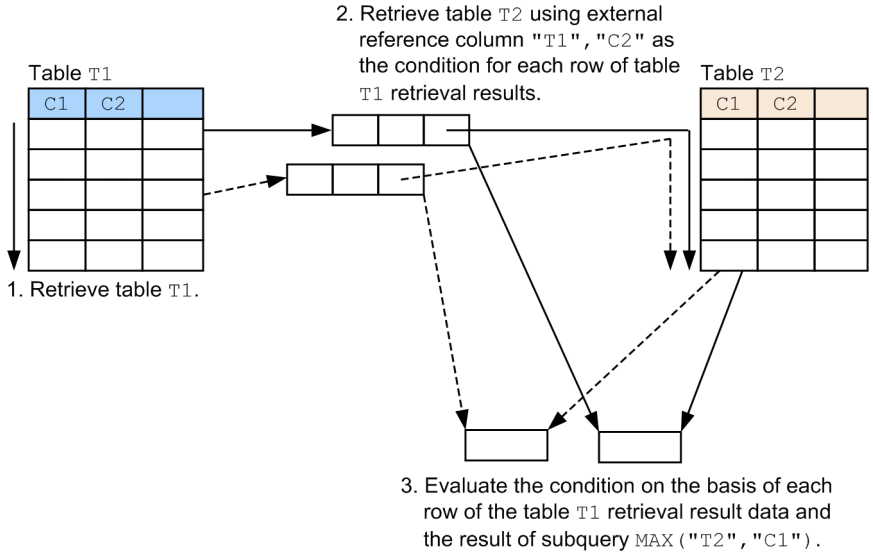
- A scalar subquery is specified
- The EXISTS predicate is specified

The following shows an example of nested loops row value execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
    WHERE "T1"."C1"=(SELECT MAX("T2"."C1") FROM "T2"
                          WHERE "T2"."C2"="T1"."C2")
```

Figure 5-13: Processing method for nested loops row value execution



Explanation:

1. Executes the query that is outside the subquery.

   This example retrieves table T1.

2. Executes the subquery by using the value of an external reference column each time one row of query outside the subquery is fetched.

   This example obtains the result of subquery MAX("T2"."C1") by using an external reference column ("T1"."C2") for each row of table T1 retrieval results.

3. Obtains the results of the executed subquery (no work table is created). HADB then uses the results of the subquery to evaluate the condition that contains a subquery outside the subquery.

   This example evaluates the condition by using the value of the corresponding MAX("T2"."C1") for each row of table T1 retrieval results.

(3) Hash execution

A method for processing subqueries by using a hash table is called *hash execution*. Hash execution might be applied in the following cases:

- The EXISTS predicate is specified
- A scalar subquery is specified

If hash execution is applied during subquery processing, HADB first executes the subquery from which a condition containing the external reference column is excluded, and then creates a hash table from the result. Then, HADB executes the query outside the subquery, and then generates a hash value from the value of the external reference column. Finally, processing is performed to match the values with the hash table.

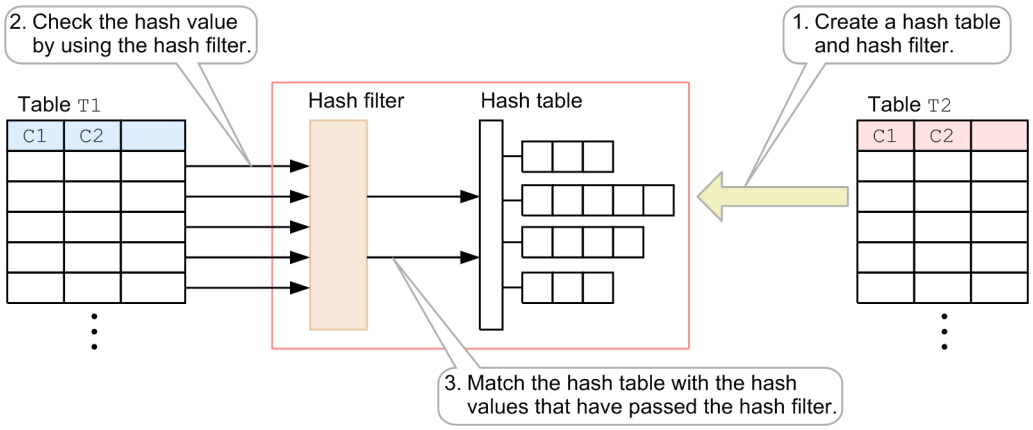
When a hash table is created, a hash filter is also created. HADB filters hash values by using the hash filter before matching the hash values with the hash table. This reduces the number of times hash values are matched with the hash table.

The following shows an example of hash execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
   WHERE "T1"."C3"<(SELECT "T2"."C3" FROM "T2"
                        WHERE "T2"."C1"='A' AND "T2"."C2"="T1"."C2")
```

Figure 5-14: Processing method for hash execution



Explanation:

1. Executes the subquery from which a condition containing the external reference column (underlined portion in the example SQL statement) is excluded, and then creates a hash table and hash filter based on the result.
2. Executes the query outside the subquery, and then generates a hash value from the value of the external reference column (`"T1"."C2"`). The hash value is checked by using the hash filter. This example retrieves table `T1`, generates a hash value from the value of the external reference column (`"T1"."C2"`), and then checks the hash value by using the hash filter.
3. Performs processing to match the hash table with the hash values that have passed the hash filter.

The range index might be used when a hash execution is processed if both of the following two conditions are met:

- There is a table specified in a query outside a subquery (table `T1` in the preceding example). In the table, the range index is defined for an external reference column (`T1.C2` in the preceding example).
- The conditions under which the range index can be used are met.

  For details about the conditions under which range indexes are used, see 5.3.1 Conditions under which range indexes are used during execution of an SQL statement.

If both of the preceding two conditions are met, the maximum and minimum values of the column to be compared with an external reference column are obtained when: a hash table is created from the result of executing a subquery excluding the conditions that contain the external reference column during processing of a hash execution. The range index is then used when the table specified in a query outside the subquery (table `T1` in the preceding example) is searched. The range index is used to skip the table's chunks or segments that are not

within the obtained maximum and minimum values of the column to be compared with the external reference column.

A hash table is created in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. Note that when `0` is specified in the `adb_sql_exe_hashtbl_area_size` operand, hash execution is not applied.

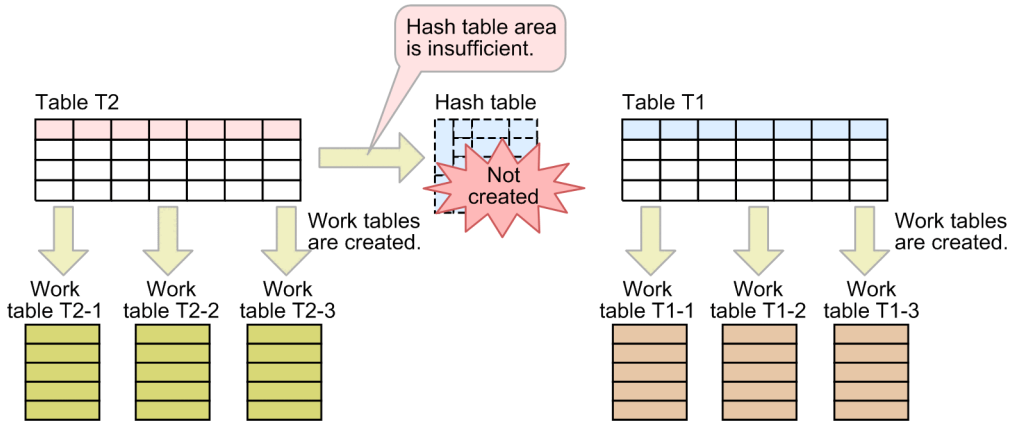
A hash filter is created in the hash filter area. The size of the hash filter area is specified in the `adb_sql_exe_hashflt_area_size` operand in the server definition or the client definition. If `0` is specified for the `adb_sql_exe_hashflt_area_size` operand, a hash filter is not applied during hash execution.

■ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

Note that tables `T1` and `T2` in the following explanation correspond to tables `T1` and `T2` in Figure 5-14: Processing method for hash execution.

1. If there is insufficient hash table area to create the hash table, the data of table `T2` is stored in multiple work tables. Also, the data of table `T1` is stored in multiple work tables in the same way as for the data of table `T2`.

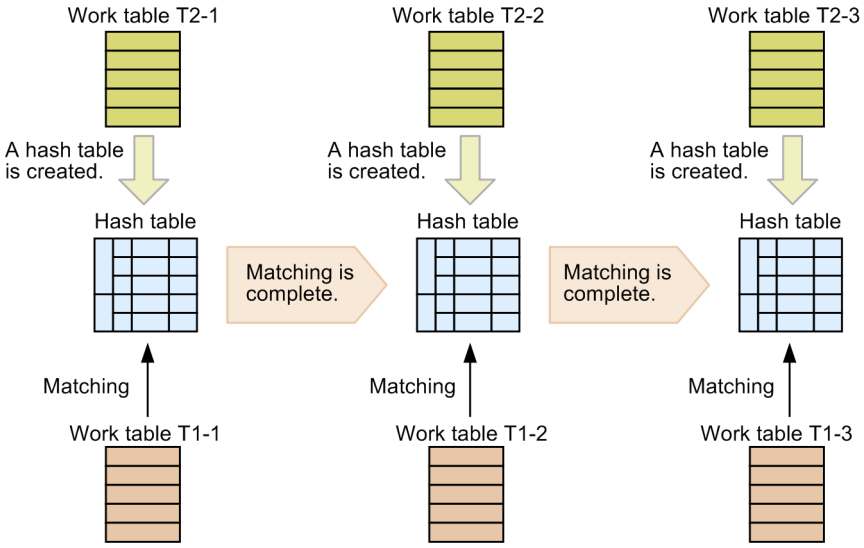


> **Note**
>
> In the preceding example, three work tables are created for table `T2`, and three work tables are created for table `T1`. The number of work tables that are created differs depending on conditions, such as the specification of the SQL statement.

2. A hash table is created with a work table for table `T2` (work table `T2-1`), and then the hash table is matched with a work table for table `T1` (work table `T1-1`).

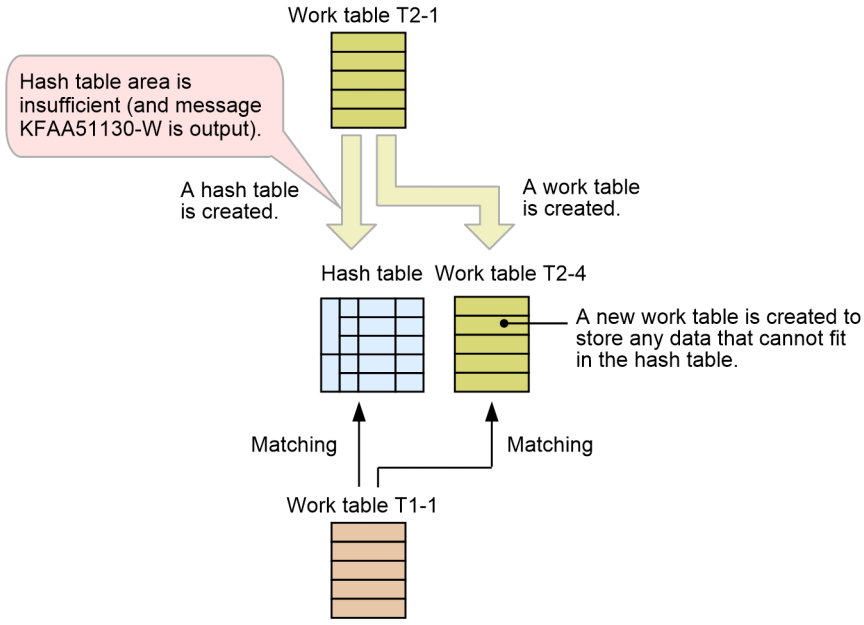
   After matching between the hash table and work table `T1-1` is complete, a hash table is created with work table `T2-2`, and the hash table is matched with work table `T1-2`.

   After matching between the hash table and work table `T1-2` is complete, a hash table is created with work table `T2-3`, and the hash table is matched with work table `T1-3`.



If the hash table area becomes insufficient during the processing in step 2

If the hash table area becomes insufficient during the processing in step 2, any data that cannot fit in the hash table is stored in another work table. In this case, in addition to matching between a work table for table `T1` and a hash table, matching between a work table for table `T1` and the newly created work table (work table `T2-4`) occurs.



> **Note**
>
> If a new work table (work table `T2-4`) is created due to insufficient hash table area during the processing in step 2, the `KFAA51130-W` message is output to the server message log file.

■ Action to take when the hash table area has insufficient space

If the hash table area is insufficient, processing time for the SQL statement might take longer due to work table creation and matching. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

■ **Action to take when the hash filter area has insufficient space**

If the size of the hash filter area specified for the `adb_sql_exe_hashflt_area_size` operand in the server definition or client definition is too small, a shortage might occur in the hash filter area allocated for each hash retrieval. As a result, a hash filter is not applied to any hash retrieval for which the size of the hash filter is insufficient. If you want to apply the hash filter to all types of hash retrieval, increase the value of the `adb_sql_exe_hashflt_area_size` operand. For details about the formula for estimating the value to be specified, see (6) Conditions where a hash filter is applied in 5.5.2 About hash join.

5.6.4 Characteristics of the methods for processing subqueries that contain an external reference column

The following table describes the characteristics of each method for processing subqueries that contain an external reference column.

Table 5-7: Methods for processing subqueries that contain an external reference column

No.	Processing method	Benefits	Disadvantages
1	Nested loops work table execution	B-tree indexes or text indexes can be used for subquery search conditions that contain an external reference column. This allows data to be retrieved at a higher speed when B-tree indexes or text indexes are used to narrow the search range.	Processing performance decreases when the hit count of the queries outside the subquery is high.
2	Nested loops row value execution		
3	Hash execution	Data can be retrieved at a higher speed when all data required for the processing can be stored in the hash table.	If a large amount of data must be stored in the hash table, the size of the hash table area becomes large. Processing performance decreases if a shortage occurs in the hash table area, because all data is first saved to a work table.

5.7 Grouping methods

There are two types of grouping methods:

- Hash grouping
- Sort grouping

Grouping is performed when the `GROUP BY` clause or the `DISTINCT` set function is specified. This section explains the grouping methods.

HADB automatically determines which grouping method to use. You can find out which grouping method was used by viewing the access path after the SQL statement is executed. For details about access paths, see the following subsections:

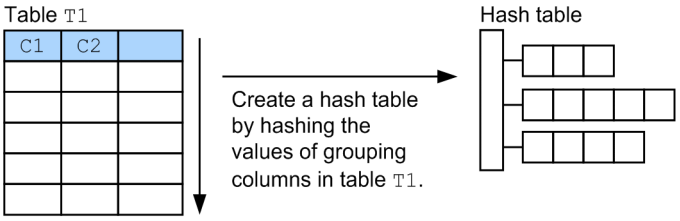
- How to check access paths
  See 6.1.2 How to check access paths.
- Information that is displayed as access paths
  See (13) Grouping methods in 6.1.4 Information displayed in the tree view.

> **Note**
>
> You can prevent the application of global hash grouping by specifying a grouping method specification. For details about grouping method specifications, see *Specification format and rules for GROUP BY clauses* in the manual *HADB SQL Reference*.

5.7.1 Hash grouping

A hash grouping method groups data while creating a hash table by hashing the values of grouped columns. The following figure shows the hash grouping methods.

Figure 5-15: Hash grouping method



The two types of hash grouping are described below.

(1) Local hash grouping

First, a hash table is created for each SQL processing real thread and grouping is performed. Next, the results grouped by SQL processing real threads are collected, and then the entire data is grouped. This processing method is called local hash grouping.

A hash table is created for each SQL processing real thread in the hash grouping area. The size of a hash grouping area per hash table is specified in the `adb_sql_exe_hashgrp_area_size` operand in the server definition or the client definition.

(2) Global hash grouping

A hash table to be shared among multiple SQL processing real threads is created, and then grouping is performed. This processing method is called global hash grouping.

A hash table is created in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. Note that when `0` is specified in the `adb_sql_exe_hashtbl_area_size` operand, global hash grouping is not applied.

If an SQL statement containing the `DISTINCT` set function is executed, global hash grouping might be applied to eliminate duplicate retrieval results.

■ Action to take when the hash table area has insufficient space

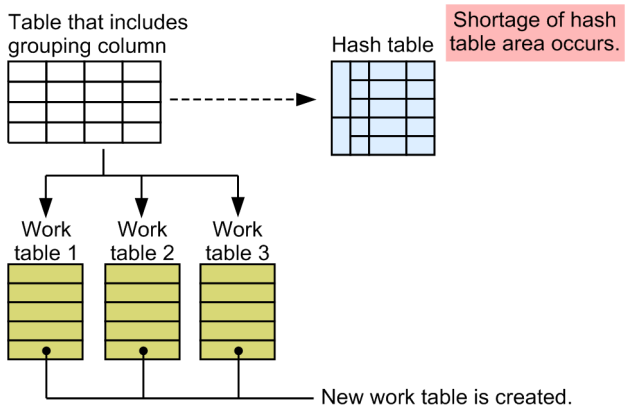
When the hash table area has insufficient space, the data stored in the hash table is spread over multiple work tables. This results in SQL statements taking longer to process. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

When a work table is created due to insufficient hash table area, the `KFAA51130-W` message is output to the server message log file.

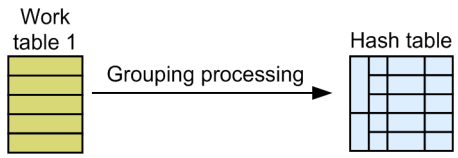
■ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

1. If there is insufficient space in the hash table area when creating the hash table, HADB creates multiple work tables. The data stored in the hash table is spread across these work tables.

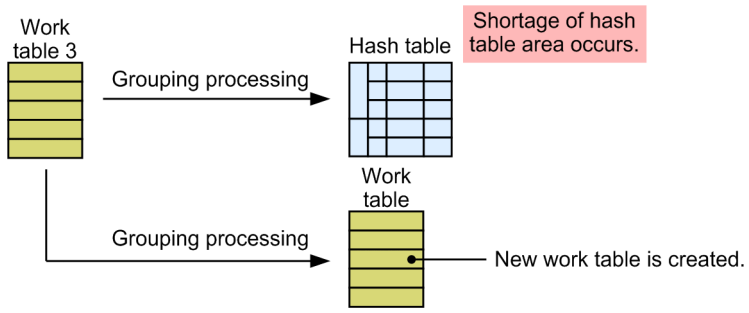


2. HADB performs grouping while creating the hash table for work table 1.



When this grouping has finished, HADB performs grouping while creating the hash table for work table 2. It then performs the same processing for work table 3.

If the hash table area runs out of space when creating the hash table for a work table, HADB creates a new work table. The data that did not fit in the hash table area is stored in the new work table. In this situation, additional grouping using the newly created work table takes place.



5.7.2 Sort grouping

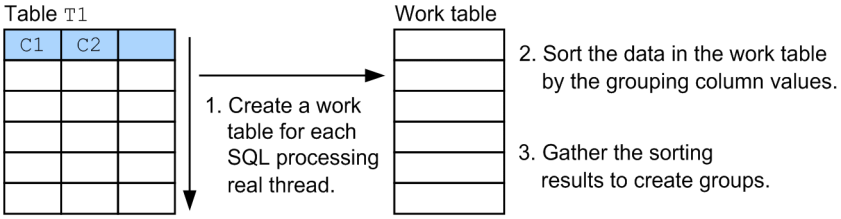
Sort grouping is a grouping method that groups data after sorting the data.

First, a work table is created for each SQL processing real thread. Next, the data in each work table is sorted by the values of the grouping column, and then the sorted data for all SQL processing real threads is collected to create a group.

If an SQL statement containing the `DISTINCT` set function is executed, sort grouping might be applied to eliminate duplicate retrieval results.

The following figure shows the sort grouping methods.

Figure 5-16: Sort grouping method



5.7.3 Characteristics of each type of grouping

The following table describes the characteristics of each type of grouping.

Table 5-8: Characteristics of each type of grouping

No.	Grouping type		Benefits	Disadvantages
1	Hash grouping	Local hash grouping	Grouping is performed at a higher speed if the data required for grouping can fit in the hash grouping area for each SQL processing real thread, such as when there are only a few groups.	Processing performance decreases when there are many groups.
2		Global hash grouping	Grouping is performed at a higher speed if the data required for grouping can fit in the hash table area.	Processing performance decreases when grouping requires a large amount of data, and a shortage of space occurs in the hash table area, because the data is first stored in the work table.
3	Sort grouping		Grouping is possible even if no hash grouping area or hash table area is allocated.	If there are many retrieval results, a large amount of data is stored in the work table. Processing performance decreases

No.	Grouping type	Benefits	Disadvantages
			because the data stored in the work table is sorted, and then is grouped.

5.8 Methods for processing set operations

If you specify a set operation (excluding `UNION ALL`), the set operation is executed by using either of the following processing methods:

- Hash execution
- Work table execution

HADB automatically determines which method to use. You can find out which processing method is used by viewing the access path. For details about access paths, see the following sections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information that is displayed as access paths
  See (12) Processing method for duplicate removal in 6.1.4 Information displayed in the tree view.

> **Note**
>
> - You can prevent the application of hash execution by specifying a set operation method specification. For details about set operation method specifications, see *Specification format and rules for query expressions* in the manual *HADB SQL Reference*.
> - If you specify `UNION ALL`, data of the query expression body specified in the operands of the set operation will be returned as is.

5.8.1 Hash execution

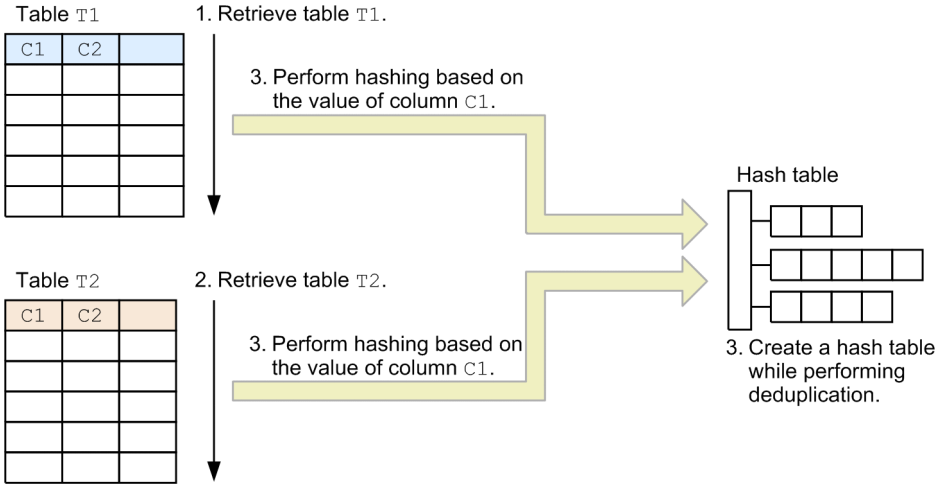
If hash execution is used as the method for processing the set operation, HADB performs deduplication while creating a hash table by hashing retrieval results.

The following shows an example of hash execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
UNION
SELECT "T2"."C1" FROM "T2"
```

Figure 5-17: Processing method for hash execution



Explanation:

1. Retrieves table `T1`, and then extracts the value of column `C1` of table `T1`.
2. Retrieves table `T2`, and then extracts the value of column `C1` of table `T2`.
3. Performs deduplication while creating a hash table by hashing the results of steps 1 and 2.

A hash table is created in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. Note that when `0` is specified in the `adb_sql_exe_hashtbl_area_size` operand, hash execution is not applied.

■ Action to take when the hash table area has insufficient space

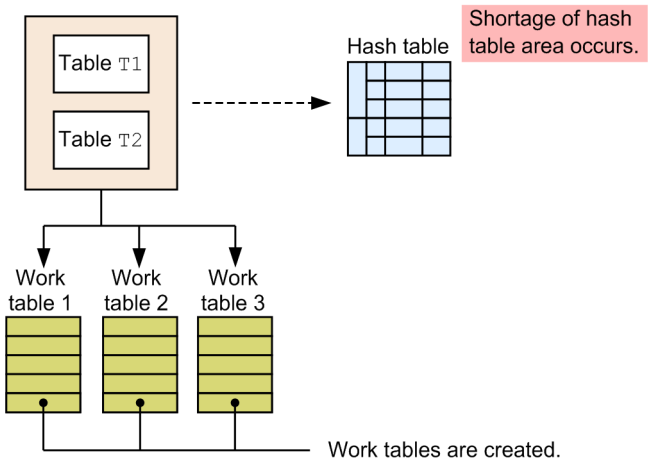
When the hash table area has insufficient space, the data stored in the hash table is spread over multiple work tables. This results in SQL statements taking longer to process. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

When a work table is created as a result of insufficient hash table area, the message `KFAA51130-W` is output to the server message log file.

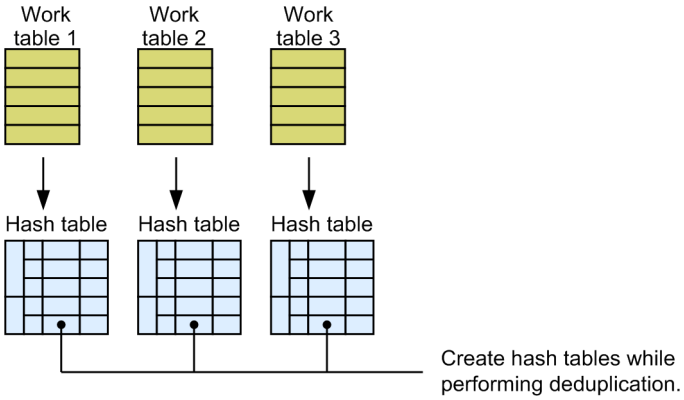
■ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

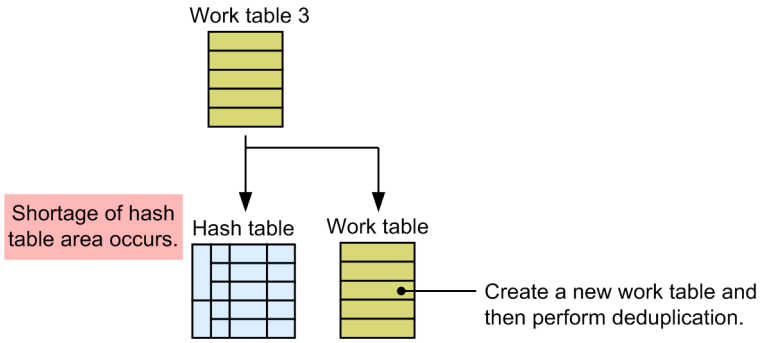
1. If there is insufficient space in the hash table area to create the hash table, HADB creates multiple work tables. The data to be stored in the hash table is spread across these work tables.



2. A hash table is created for each work table.



If the hash table area runs out of space when creating hash tables for work tables, HADB creates a new work table. HADB stores in this work table the data that did not fit in the hash table area, and then performs deduplication.



5.8.2 Work table execution

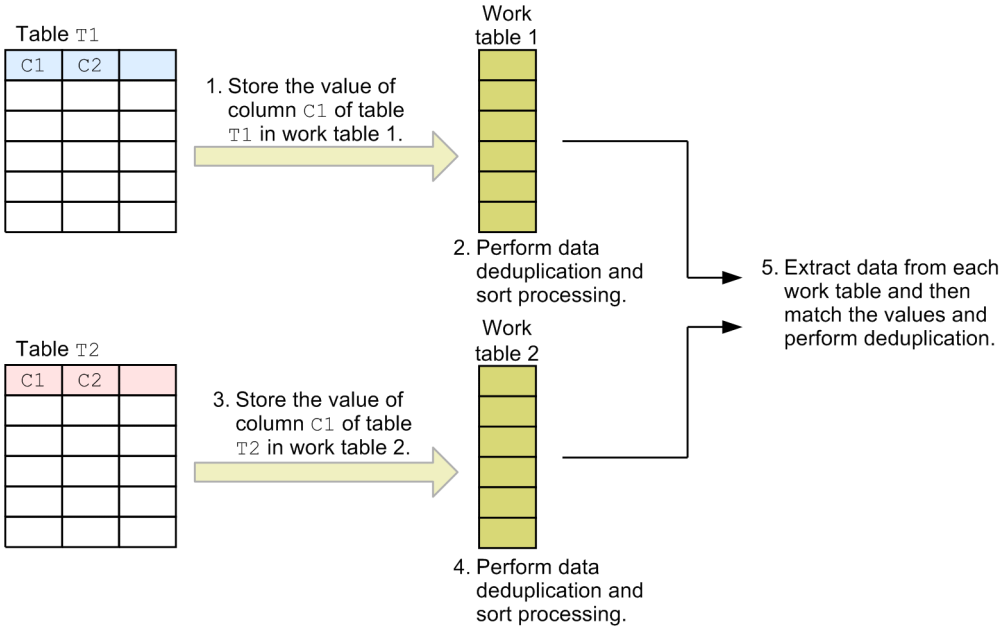
If work table execution is used as the method for processing the set operation, HADB creates a work table for each query specification, and then sorts data. If the set operation does not contain an `ALL` specification, HADB performs deduplication of data in the work table. Then, HADB evaluates the set operation by matching data in each work table. If the set operation does not contain an `ALL` specification, HADB performs deduplication of the evaluation result of the set operation.

The following shows an example of work table execution.

■ SELECT statement to be executed

```
SELECT "T1"."C1" FROM "T1"
UNION
SELECT "T2"."C1" FROM "T2"
```

Figure 5-18: Processing method for work table execution



Explanation:

1. Creates work table 1 for storing the value resulting from query specification `SELECT "T1"."C1" FROM "T1"` (retrieves table `T1`, and then stores the value of column `C1` of table `T1` in work table 1).
2. Performs data deduplication and sort processing for work table 1 created in step 1.
3. Creates work table 2 for storing the value resulting from query specification `SELECT "T2"."C1" FROM "T2"` (retrieves table `T2`, and then stores the value of column `C1` of table `T2` in work table 2).
4. Performs data deduplication and sort processing for work table 2 created in step 3.
5. Fetches data from one row at a time, and then matches the values and performs deduplication.

5.8.3 Characteristics of the methods for processing set operations

The following table shows the characteristics of each method for processing set operations.

Table 5-9: Characteristics of the methods for processing set operations

Methods for processing set operations	Benefits	Disadvantages
Hash execution	Results can be obtained at a higher speed when the data required for the set operation can be stored in the hash table area.	If a shortage occurs in the hash table area due to a large amount of data required for the set operation, processing performance decreases because the data is temporarily saved in a work table.
Work table execution	This method is applicable to all set operations.	If there are a large number of retrieval results, the amount of data stored in the work table becomes large. In this case, processing performance decreases because deduplication is performed after the data stored in the work table is sorted.

5.9 Method for processing SELECT DISTINCT

If you specify DISTINCT in the SELECT statement, deduplication is performed by using either of the following processing methods:

- Hash execution
- Work table execution

HADB automatically determines which method to use. You can find out which processing method is used by viewing the access path. For details about access paths, see the following sections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information that is displayed as access paths
  See (12) Processing method for duplicate removal in 6.1.4 Information displayed in the tree view.

Note

- You can prevent the application of hash execution by specifying the SELECT deduplication method specification. For details about the SELECT deduplication method specification, see *Specification format and rules for query specifications* in the manual *HADB SQL Reference*.
- If HADB determines that deduplication processing for SELECT DISTINCT is unnecessary, deduplication processing is not performed.
- The method for processing SELECT DISTINCT is determined based on the query expressions rewritten by expanding internal derived tables and the search conditions rewritten by equivalent exchange of search conditions. For details about expanding internal derived tables, see 5.13 Expanding internal derived tables. For details about equivalent exchange for search conditions, see 5.11 Equivalent exchange of search conditions.
- If only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

5.9.1 Hash execution

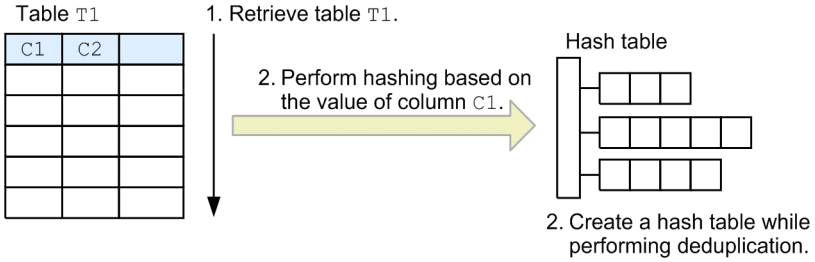
If hash execution is used as the method for processing SELECT DISTINCT, HADB performs deduplication while creating a hash table by hashing retrieval results.

The following shows an example of hash execution.

■ SELECT statement to be executed

```
SELECT DISTINCT "C1" FROM "T1"
```

Figure 5-19: Processing method for hash execution



Explanation:

1. Retrieves table `T1`, and then extracts the value of column `C1` of table `T1`.
2. Performs deduplication while creating a hash table by hashing the result of step 1.

A hash table is created in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. Note that when `0` is specified in the `adb_sql_exe_hashtbl_area_size` operand, hash execution is not applied.

■ Action to take when the hash table area has insufficient space

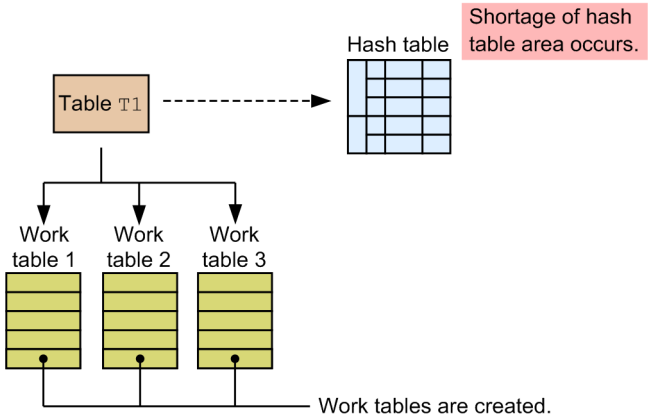
When the hash table area has insufficient space, the data stored in the hash table is spread over multiple work tables. This results in SQL statements taking longer to process. To remedy a situation in which the hash table area has insufficient space, increase the value specified for the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition. This operand specifies the size of the hash table area.

When a work table is created as a result of insufficient hash table area, the message `KFAA51130-W` is output to the server message log file.

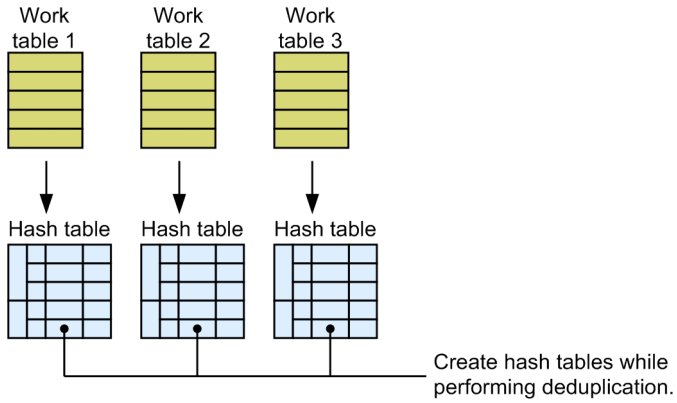
■ Flow of processing when hash table area has insufficient space

The following explains the flow of processing when the hash table area has insufficient space to create the hash table.

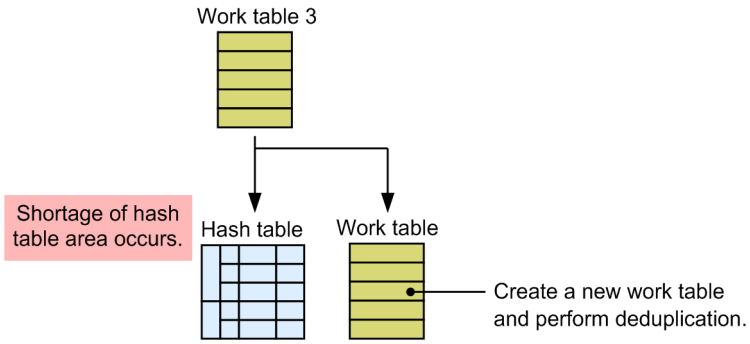
1. If there is insufficient space in the hash table area to create the hash table, HADB creates multiple work tables. The data to be stored in the hash table is spread across these work tables.



2. A hash table is created for each work table.



If the hash table area runs out of space when creating hash tables for work tables, HADB creates a new work table. HADB stores in this work table the data that did not fit in the hash table area, and then performs deduplication.



5.9.2 Work table execution

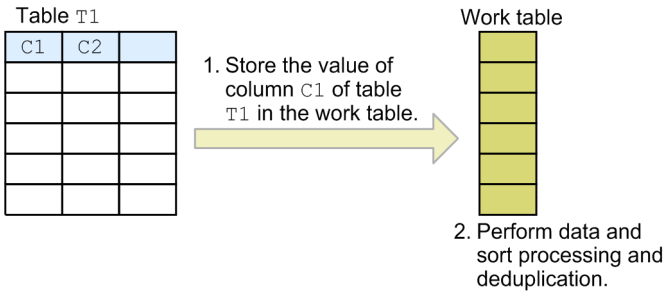
If work table execution is used as the method for processing `SELECT DISTINCT`, HADB creates a work table containing the retrieval result. Then, HADB sorts data in the work table and performs deduplication.

The following shows an example of work table execution.

■ SELECT statement to be executed

```
SELECT DISTINCT "C1" FROM "T1"
```

Figure 5-20: Processing method for work table execution



Explanation:

1. Retrieves table `T1`, and then stores the value of column `C1` of table `T1` in the work table.
2. Performs data sort processing and deduplication for the work table created in step 1.

5.9.3 Characteristics of the methods for processing SELECT DISTINCT

The following table shows the characteristics of each method for processing `SELECT DISTINCT`.

Table 5-10: Characteristics of the methods for processing SELECT DISTINCT

Method for processing SELECT DISTINCT	Benefits	Disadvantages
Hash execution	Results can be obtained at a higher speed when the data required for executing `SELECT DISTINCT` can be stored in the hash table area.	If a shortage occurs in the hash table area due to a large amount of data required for executing `SELECT DISTINCT`, processing performance decreases because the data is temporarily saved in a work table.
Work table execution	This method is applicable to any `SELECT DISTINCT`.	If there are a large number of retrieval results, the amount of data stored in the work table becomes large. In this case, processing performance decreases because deduplication is performed after the data stored in the work table is sorted.

5.10 Considerations when executing an SQL statement that creates work tables

If data sort processing or deduplication is performed during execution of an SQL statement, a work table might be created in the work table DB area. Data being sorted or data after deduplication is stored temporarily in a work table. Therefore, if you sort a large amount of data or perform deduplication for such data, sufficient performance improvement might not be obtained due to the workload of creating work tables.

> **Important**
>
> To avoid performance degradation, you must estimate accurately the size of the work table DB area that is needed. If more SQL statements for which work tables are created are executed than initially expected, the application program developer must request that the HADB system designer or system administrator re-estimate the size of the work table DB area.

5.10.1 Types of work tables

The two types of work tables are global work tables and local work tables.

(1) Global work table

Global work tables are used to share data in work tables among multiple real threads when the same SQL statement is processed by multiple real threads. These work tables are used for joining tables and processing subqueries.

The number of pages in the global buffer that is used by global work tables is specified in the `adb_dbbuff_wrktbl_glb_blk_num` server definition operand. For details about the `adb_dbbuff_wrktbl_glb_blk_num` operand, see the topic *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*.

(2) Local work table

A local work table is created for each SQL processing real thread. These work tables are used when the `ORDER BY` clause is specified and for grouping when local hash grouping is selected as the grouping method.

The number of local work table buffer pages that are used by local work tables is specified in the `adb_dbbuff_wrktbl_clt_blk_num` server definition operand, the `adb_dbbuff_wrktbl_clt_blk_num` client definition operand, or the `adb_export_wrktbl_blk_num` export option (when the `adbexport` command is executed). For details about these operands and option, see the following sections:

- `adb_dbbuff_wrktbl_clt_blk_num` operand in the server definition: Topic *Operands related to performance (set format)* in *Detailed descriptions of the server definition operands* in *Designing the Server Definition* in the *HADB Setup and Operation Guide*
- `adb_dbbuff_wrktbl_clt_blk_num` operand in the client definition: Explanation of the adb_dbbuff_wrktbl_clt_blk_num operand in 2.2.3 Operands related to performance
- `adb_export_wrktbl_blk_num` export option: *Specification format for the adbexport command* in *adbexport (Export Data)* in the manual *HADB Command Reference*

5.10.2 Work tables created when SQL statements are executed

Work tables are created when you execute the SQL statements listed in the table below.

Important

If the row length of the work table exceeds the maximum row length, the SQL statement will result in an error. For details about the maximum row length of a work table, see *Maximum and minimum values related to database* in the *HADB Setup and Operation Guide*. For details about how to determine the row length of a work table, see the description of variable *ROWSZ* in *Determining the number of pages for base rows that are needed for storing work tables* in the *HADB Setup and Operation Guide*.

Table 5-11: SQL statements for which work tables are created

No.	SQL statement for which work table is created		Purpose of work table	Columns of work table	Work table type
1	`ORDER BY` clause is specified.	Results of a set operation are used as a sort key.	Sorting the retrieval results	• Store the values of columns resulting from query expressions	Local work table
		Results of a query specification are used as a sort key.		• Store the results of value expressions specified for sort keys in the `ORDER BY` clause • Store the row IDs of the rows in the retrieval results for the base table (when the base table is specified in the `FROM` clause)#1 • Store the values of retrieval target columns (when the base table with key scan used as the table retrieval method is specified in the `FROM` clause) • Store the values of retrieval target columns in the column store table#11 (when a column store table is specified in the `FROM` clause) • Store the column values of results derived from query expression bodies in derived tables (when a derived table is specified in the `FROM` clause)#6 • Store the column values of results derived from query expressions in view definitions (when a viewed table is specified in the `FROM` clause)#6 • Store the column values of results derived from query expression bodies in `WITH` clauses (when a query name is specified in the `FROM` clause)#6	

No.	SQL statement for which work table is created		Purpose of work table	Columns of work table	Work table type
				• Store the column values of results derived from system-defined functions specified in table function derived tables (when a table function derived table is specified in the FROM clause) • Store the values resulting from the scalar function RANDOMROW (when the scalar function RANDOMROW is specified in the selection expression)	
2	GROUP BY clause is specified.	The grouping method is global hash grouping.#2	Retaining the grouping results. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the adb_sql_exe_hashtbl_area_size operand in the server definition or the client definition.	• Store the values resulting from the value expressions specified for grouped columns in the GROUP BY clause • Store the results of set functions • Store information the HADB server uses for hashing#9, #10	• Local work table#7 • Global work table#8
		The grouping method is local hash grouping#2	Sorting for grouping. This work table is used when there is insufficient hash grouping area. The size of the hash grouping area is specified in the adb_sql_exe_hashgrp_area_size operand in the server or client definition.	• Store the values resulting from the value expressions specified for grouped columns in the GROUP BY clause • Store the columns specified in the arguments of set functions • Store the results of set functions	Local work table
		The grouping method is sort grouping#2.	Sorting for grouping.		
3	SELECT DISTINCT is specified.	The method for processing SELECT DISTINCT is hash execution.#12	Retaining retrieval results. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the adb_sql_exe_hashtbl_area_size operand in the server definition or the client definition.	• Store the values of columns resulting from selection expressions • Store information HADB uses for hashing#9, #10	• Local work table#7 • Global work table#8
		The method for processing SELECT DISTINCT is work table execution.#12	Sorting retrieval results or eliminating duplicate retrieval results	• Store the values of columns resulting from selection expressions	Local work table
4	DISTINCT set function or inverse distribution function is specified.	The grouping method is global hash grouping.#2	Retaining the input values of set functions whose duplicates have been eliminated. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the adb_sql_exe_hashtbl_area_size operand in the server definition or the client definition.	• Store the values resulting from the value expressions specified for grouped columns in the GROUP BY clause • Store the values of columns specified in the arguments of the ALL set function	• Local work table#7 • Global work table#8

No.	SQL statement for which work table is created		Purpose of work table	Columns of work table	Work table type
		The grouping method is not global hash grouping.[#2]	Eliminating the duplicates of input values for set functions or sorting the input values for set functions	• Store the results of value expressions specified in the arguments of the DISTINCT set function • Store the results of value expressions specified for a sort key in the WITHIN group specification in an inverse distribution function • Store the results of value expressions specified in the arguments of the MEDIAN inverse distribution function • Store information the HADB server uses for hashing[#9, #10]	
5	Window functions are specified.		Sorting for obtaining the results of window functions	• Store the values of columns of a table specified in the FROM clause • Store the values resulting from the window function specified in the selection expression	Local work table
6	Multiple table references are specified in the FROM clause.	The table joining method is hash join.[#3]	Retaining the results of table references subject to table join processing. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the adb_sql_exe_hashtbl_area_size operand in the server definition or the client definition.	• Store the values of columns of a table specified in the FROM clause • Store information the HADB server uses for hashing[#9, #10]	• Local work table[#7] • Global work table[#8]
		The table joining method is nested loop join.[#3]	Retaining the results of table references subject to table join processing		Global work table
7	Derived tables are specified.		Retaining the results of query expression bodies corresponding to derived tables	• Store the values of columns resulting from a query expression body in derived tables[#6]	Global work table
8	Viewed tables are specified.		Retaining the results of query expressions corresponding to viewed tables	• Store the values of result columns derived from a query expression in view definitions[#6]	Global work table
9	WITH clause is specified.		Retaining the results of query expression bodies corresponding to query names	• Store the values of columns resulting from a query expression body in the WITH clause[#6]	Global work table
10	A table function derived table is specified.		Retaining the results of system-defined functions that derive table function derived tables.	• Store the values resulting from the system-defined function	Global work table
11	Joined tables are specified.		Retaining the results of joined tables.	• Store the values of columns in joined tables	Global work table

No.	SQL statement for which work table is created		Purpose of work table	Columns of work table	Work table type
			For details about the work tables used to obtain the results of joined tables, see *Multiple table references are specified in the FROM clause* in row 6.		
12	Subqueries are specified.	The subquery processing method is hash execution.[#4]	Retaining the results of subqueries. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition.	• Store the values of columns resulting from a selection expression in subqueries • Store the values of external reference columns contained in subqueries[#5] • Store the results of set functions contained in subqueries • Store information the HADB server uses for hashing[#9, #10]	• Local work table[#7] • Global work table[#8]
		The subquery processing method is not hash execution.[#4]	Retaining the results of subqueries	• Store the values of columns resulting from a selection expression in subqueries	Global work table
13	A set operation is specified.	The method for processing the set operation is hash execution.[#13]	Retaining the results of deduplication. This work table is used when a shortage occurs in the hash table area. The size of the hash table area is specified in the `adb_sql_exe_hashtbl_area_size` operand in the server definition or the client definition.	• Store the values of columns resulting from the selection expression specified in a query specification • Store information HADB uses for hashing[#9, #10]	• Local work table[#7] • Global work table[#8]
		The method for processing the set operation is work table execution.[#13]	Sorting retrieval results or eliminating duplicate retrieval results	• Store the values of columns resulting from the selection expression specified in a query specification	Local work table
14	A recursive query is specified.		Retaining the results of anchor members and recursive members	• Store the values of columns resulting from a recursive query	Global work table

#1

The row ID is a value that indicates a row's storage location. The data type of a row ID is `CHAR(16)`.

#2

For details about grouping methods, see 5.7 Grouping methods.

#3

For details about table joining methods, see 5.5 Table joining methods.

#4

For details about how to process subqueries, see 5.6 How to process subqueries.

#5

For details about external reference columns, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.

#6

Expansion of the internal derived table might result in this table being handled as the base table. For details about expansion of internal derived tables, see *Internal derived tables* in the manual *HADB SQL Reference*.

#7

A work table used to store data that the hash table area cannot accommodate due to insufficient space.

#8

A work table used to store data that cannot be processed because there is still insufficient hash table area after data has been stored in work tables.

#9

The data type of columns that store the information HADB uses for hashing is `INTEGER`.

#10

A column created in a work table used to store data that cannot be processed because there is still insufficient hash table area after data has been stored in work tables.

#11

Columns with a definition length of 128 or more bytes are not subject to retrieval. A column with a definition length of 128 or more bytes refers to any of the following columns:

- A column of `CHARACTER` type with a definition length of 128 or more bytes
- A column of `VARCHAR` type with a definition length of 128 or more bytes
- A column of `BINARY` type with a definition length of 128 or more bytes
- A column of `VARBINARY` type with a definition length of 128 or more bytes

#12

For details about the method for processing `SELECT DISTINCT`, see 5.9 Method for processing SELECT DISTINCT.

#13

For details about the method for processing the set operation, see 5.8 Methods for processing set operations.

You can find out whether work tables were created by checking the access path after the SQL statement has executed. For details about access paths, see the following sections:

- How to check access paths
  See 6.1.2 How to check access paths.
- Information displayed in access paths
  See (7) Work table creation information in 6.1.4 Information displayed in the tree view.

5.10.3 Number of work tables that are created

This subsection explains by way of examples the number of work tables that are created when an SQL statement is executed. Note that the number of work tables explained here is based on the SQL statements. The number of work tables that are actually created during execution of SQL statements is affected by other factors, including the maximum number of SQL processing real threads in the server definition or client definition (`adb_sql_exe_max_rthd_num` operand) and the number of data items to be manipulated.

(1) Example 1 (when the ORDER BY clause is specified)

Example SQL statement

```
SELECT "C1","C2","C3" FROM "T1" ORDER BY "C1" ASC
```

Explanation:

One work table is created for sort processing by the `ORDER BY` clause.

(2) Example 2 (when the GROUP BY clause is specified)

Example SQL statement

```
SELECT "C1","C2" FROM "T1" GROUP BY "C1","C2"
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  Two work tables are created for processing by the `GROUP BY` clause.
- When global hash grouping is not applied

  One work table is created for processing by the `GROUP BY` clause.

For details about grouping methods, see 5.7 Grouping methods.

(3) Example 3 (when SELECT DISTINCT is specified)

Example SQL statement

```
SELECT DISTINCT "C1","C2","C3" FROM "T1"
```

Explanation:

The number of work tables that are created depends on the applied method for processing `SELECT DISTINCT`.

- When hash execution is applied

  Two work tables are created for processing `SELECT DISTINCT`.
- When work table execution is applied

  One work table is created for processing `SELECT DISTINCT`.

For details about the method for processing `SELECT DISTINCT`, see 5.9 Method for processing SELECT DISTINCT.

(4) Example 4 (when the DISTINCT set function is specified)

Example SQL statement

```
SELECT COUNT(DISTINCT "C1") FROM "T1"
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  Two work tables are created for processing by the set function.

- When global hash grouping is not applied

  One work table is created for processing by the set function.

For details about grouping methods, see 5.7 Grouping methods.

(5) Example 5 (when the RANK window function is specified)

Example SQL statement

```
SELECT "C1",RANK() OVER (PARTITION BY "C2" ORDER BY "C3" ASC) FROM "T1"
```

Explanation:

One work table is created for the RANK window function processing.

(6) Example 6 (when the GROUP BY and ORDER BY clauses are both specified)

Example SQL statement

```
SELECT "C1","C2" FROM "T1" GROUP BY "C1","C2" ORDER BY "C1" ASC
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  A total of three work tables are created. Two are created for processing by the GROUP BY clause, and one is created for processing by the ORDER BY clause.

- When global hash grouping is not applied

  One work table is created for processing by the GROUP BY clause. Only one work table is created in this example because the GROUP BY and ORDER BY clauses only require sorting to take place once.

For details about grouping methods, see 5.7 Grouping methods.

(7) Example 7 (when the GROUP BY and ORDER BY clauses are both specified)

Example SQL statement

```
SELECT "C1","C2",COUNT(*) "DC1" FROM "T1"
     GROUP BY "C1","C2" ORDER BY "DC1" ASC
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  A total of three work tables are created. Two are created for processing by the GROUP BY clause, and one is created for processing by the ORDER BY clause.

- When global hash grouping is not applied

  A total of two work tables are created. One is created for processing by the GROUP BY clause, and one is created for processing by the ORDER BY clause.

For details about grouping methods, see 5.7 Grouping methods.

(8) Example 8 (when SELECT DISTINCT and the ORDER BY clause are specified)

Example SQL statement

```
SELECT DISTINCT "C1","C2","C3" FROM "T1" ORDER BY "C1" ASC
```

Explanation:

One work table is created for sort processing by SELECT DISTINCT.

In this example, only one work table is created because the number of sort processes required for SELECT DISTINCT and the ORDER BY clause is only one.

(9) Example 9 (when the GROUP BY clause and a DISTINCT set function are specified)

Example SQL statement

```
SELECT "C1",COUNT(DISTINCT "C2") FROM "T1" GROUP BY "C1"
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  A total of three work tables are created. Two are created for processing by the GROUP BY clause, and one is created for processing by the set function.

- When global hash grouping is not applied

  One work table is created for processing by the GROUP BY clause. Only one work table is created in this example because the GROUP BY clause and set function only require sorting to take place once.

For details about grouping methods, see 5.7 Grouping methods.

(10) Example 10 (when tables are joined)

Example SQL statement

```
SELECT "T1"."C1","T1"."C2","T2"."C1","T2"."C2" FROM "T1","T2"
    WHERE "T1"."C1"="T2"."C1"
```

Explanation:

Three work tables for join processing are created.

(11) Example 11 (multiple table references are specified in the FROM clause)

Example SQL statement

```
SELECT "DT1"."C1","DT2"."C1"
  FROM (SELECT COUNT("T1"."C1") FROM "T1") AS "DT1"("C1"),
       (SELECT COUNT("T2"."C1") FROM "T2") AS "DT2"("C1")
  WHERE "DT1"."C1">"DT2"."C1"
```

Explanation:

One work table that is used for table join processing is created.

(12) Example 12 (subquery is specified)

Example SQL statement

```
SELECT "T1"."C1","T1"."C2","T1"."C3" FROM "T1"
    WHERE "T1"."C1"=(SELECT "T2"."C1" FROM "T2"
                                    WHERE "T2"."C2"="T1"."C2")
```

Explanation:

The number of work tables that are created depends on whether hash execution is specified as the subquery processing method.

- When hash execution is applied

  Three work tables are created to process the subquery containing the external reference columns.

- When hash execution is not applied

  No work table is created to process the subquery containing the external reference columns.

For details about how to process subqueries, see 5.6 How to process subqueries.

(13) Example 13 (IN subqueries are specified)

Example SQL statement

```
SELECTf"C1","C2","C3" FROM "T1" WHERE "C1" IN (SELECT "C1" FROM "T2")
```

Explanation:

The number of work tables that are created depends on whether hash execution is specified as the subquery processing method.

- When hash execution is applied

  Three work tables are created to process the subquery specified in the `IN` predicate.

- When hash execution is not applied

  One work table is created to process the subquery specified in the `IN` predicate.

For details about how to process subqueries, see 5.6 How to process subqueries.

(14) Example 14 (quantified predicates are specified)

Example SQL statement

```
SELECT "C1","C2","C3" FROM "T1" WHERE "C1"=ANY(SELECT "C1" FROM "T2")
```

Explanation:

The number of work tables that are created depends on whether hash execution is specified as the subquery processing method.

- When hash execution is applied

  Three work tables are created to process the subquery specified in the quantified predicate.

- When hash execution is not applied

  One work table is created to process the subquery specified in the quantified predicate.

For details about how to process subqueries, see 5.6 How to process subqueries.

(15) Example 15 (EXISTS predicate is specified)

Example SQL statement

```
SELECT "T1"."C1","T1"."C2","T1"."C3" FROM "T1"
    WHERE EXISTS(SELECT * FROM "T2" WHERE "T2"."C2"="T1"."C2")
```

Explanation:

The number of work tables that are created depends on whether hash execution is specified as the subquery processing method.

- When hash execution is applied

  Three work tables are created to process the subquery specified in the `EXISTS` predicate.

- When hash execution is not applied

  No work table is created to process the subquery specified in the `EXISTS` predicate.

For details about how to process subqueries, see 5.6 How to process subqueries.

(16) Example 16 (table function derived table is specified)

Example SQL statement

```
SELECT "C1","C2","C3"
    FROM TABLE(ADB_CSVREAD(MULTISET (SELECT "FNAME" FROM "TFILE"),
                           'COMPRESSION_FORMAT=GZIP;
                            FIELD_NUM=1,2,3;'))
          AS "T1"("C1" INTEGER,"C2" CHAR(10),"C3" DATE)
```

Explanation:

One work table is created to store the results of the table subquery specified in the multiset value expression. For details about multiset value expressions, see *Specification format and rules for multiset value expressions* in the manual *HADB SQL Reference*.

(17) Example 17 (viewed tables are specified)

Example SQL statement

```
CREATE VIEW "VT1"("C1","C2")
    AS SELECT "T1"."C1","T2"."C1" FROM "T1","T2"
        WHERE "T1"."C2"<="T2"."C2"
SELECT * FROM "VT1" AS "XT1","VT1" AS "XT2"
    WHERE "XT1"."C1">"XT2"."C1"
```

Explanation:

One work table that is used in the viewed table processing is created.

(18) Example 18 (WITH clauses are specified)

Example SQL statement

```
WITH "QT1"("C1","C2") AS (SELECT "T1"."C1","T2"."C1" FROM "T1","T2"
                            WHERE "T1"."C2"<="T2"."C2")
SELECT * FROM "QT1" WHERE "C1"=(SELECT MAX("C1") FROM "QT1")
```

Explanation:

One work table that is used in the `WITH` clause processing is created.

(19) Example 19 (a set operation is specified)

Example SQL statement

```
SELECT "C1","C2" FROM "T1" UNION SELECT "C1","C2" FROM "T2"
```

Explanation:

Two work tables are created to process the set operation.

(20) Example 20 (a recursive query is specified)

Example SQL statement

```
WITH "QT1"("C1","C2")
  AS (SELECT "C1","C2" FROM "T1" WHERE "C2" BETWEEN 'AA' AND 'EE'
      UNION ALL
      SELECT "C1"+1,"C2" FROM "QT1" WHERE "C1"<=10)
SELECT * FROM "QT1"
```

Explanation:

In a recursive query, to retain the results of anchor and recursive members, two work tables that will be used to process the recursive query are created. For details about recursive queries, see *Query expression* in the manual *HADB SQL Reference*.

(21) Example 21 (a DISTINCT set function and GROUP BY clause are specified)

Example SQL statement

```
SELECT "C1",COUNT(DISTINCT "C2"),SUM("C3") FROM "T1"
    GROUP BY "C1"
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied

  A total of four work tables are created. Two are created for processing by the `GROUP BY` clause, and two are created for processing by the set function.

- When global hash grouping is not applied

  One work table is created for processing by the `GROUP BY` clause. In the case of this example, no work tables are created for processing by the set function because the sort processing by the `GROUP BY` clause and the sort processing by the set function are executed concurrently.

For details about grouping methods, see 5.7 Grouping methods.

(22) Example 22 (multiple DISTINCT set functions are specified)

Example SQL statement

```
SELECT COUNT(DISTINCT "C1"),COUNT(DISTINCT "C2") FROM "T1"
```

Explanation:

The number of work tables that are created depends on whether global hash grouping is selected as the grouping method.

- When global hash grouping is applied
  Three work tables are created for processing by the set functions.
- When global hash grouping is not applied
  Two work tables are created for processing by the set functions.

For details about grouping methods, see 5.7 Grouping methods.

5.11 Equivalent exchange of search conditions

HADB might convert a specified search condition so that the search condition can be evaluated more efficiently. This is called *equivalent exchange of search conditions*. When equivalent exchange is performed on search conditions, the indexes to be used during retrieval are determined based on the search conditions obtained after equivalent exchange has been applied. For details about the method for determining the index to be used during retrieval, see 5.2 B-tree indexes and text indexes used during execution of SQL statements and 5.3 Range indexes used during execution of SQL statements.

HADB performs equivalent exchange in the following order for the conditions specified in search conditions:

1. Equivalent exchange for `OR` conditions (removing from the `OR` conditions)#

   This is an equivalent exchange that removes a condition from the inside of an `OR` condition to the outside of the `OR` condition.

2. Equivalent exchange for `OR` conditions (converting to `IN` conditions)#

   This is an equivalent exchange that adds an `IN` condition created from an = condition in an `OR` condition to the outside of the `OR` condition.

3. Equivalent exchange for `OR` conditions (equivalent exchange to derived tables for which the `UNION ALL` set operation is specified)#

   This is an equivalent exchange from a search condition inside an `OR` condition to a derived table for which the `UNION ALL` set operation is specified.

4. Equivalent exchange for scalar operations

   This is an equivalent exchange that transposes scalar operations.

5. Equivalent exchange for an `IN` predicate

   This is an equivalent exchange from an `IN` predicate to the = conditions or <> conditions.

6. Equivalent exchange for a `HAVING` clause (converting to the `WHERE` clauses)

   This is an equivalent exchange from the search condition for a `HAVING` clause to the search condition for a `WHERE` clause.

7. Equivalent exchange for search conditions in SQL statements that specify derived queries (transposition to the `WHERE` clause of a derived query)

   This is an equivalent exchange that moves the search condition specified in the `WHERE` clause of an SQL statement in which a derived query is specified to the `WHERE` clause of the derived query.

#

The target of this equivalent exchange is the search condition in the `WHERE` clause.

The following subsections explain each of these types of equivalent exchange.

5.11.1 Equivalent exchange for OR conditions (removing from the OR conditions)

If the same condition is specified in an `OR` condition, equivalent exchange is performed in such a manner that the same condition is removed from the `OR` condition. When the same condition is removed from the `OR` condition, the search condition sometimes becomes more effective in narrowing down the retrieval range. This can also reduce the workload for condition evaluation because the number of identical conditions in an `OR` condition is reduced to one.

The OR conditions specified in the search condition in the WHERE clause, the ON search condition for joined tables, and the search condition in the HAVING clause are subject to this equivalent exchange.

When equivalent exchange has been performed on search conditions, the indexes to be used during retrieval are determined based on the search conditions obtained after equivalent exchange.

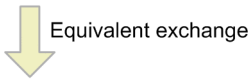
The following shows examples of equivalent exchange. In the examples, C1, C2, and C3 are column names.

(1) Examples of search conditions on which equivalent exchange is performed

(a) Example 1

■ Specified search condition

```
WHERE ("C1" = 100 AND "C2" = 'A')
   OR ("C1" = 100 AND "C3" > 20)
```



■ Search condition resulting from equivalent exchange

```
WHERE "C1" = 100
   AND ("C2" = 'A' OR "C3" > 20)
```

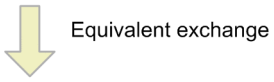
Explanation:

Because condition "C1" = 100 is specified on both sides of the OR condition, "C1" = 100 is removed from the OR condition.

(b) Example 2

■ Specified search condition

```
WHERE ("C1" < CURRENT_DATE AND "C2" = 'A')
   OR ("C1" < CURRENT_DATE AND "C3" > 20)
```



■ Search condition resulting from equivalent exchange

```
WHERE "C1" < CURRENT_DATE
   AND ("C2" = 'A' OR "C3" > 20)
```

Explanation:

Because condition "C1" < CURRENT_DATE is specified on both sides of the OR condition, "C1" < CURRENT_DATE is removed from the OR condition.

(c) Example 3

■ Specified search condition

```
WHERE ("T1"."C1" = "T2"."C1" AND "T1"."C2" = 'A')
   OR ("T1"."C1" = "T2"."C1" AND "T1"."C3" > 20)
```

■ Search condition resulting from equivalent exchange

```
WHERE "T1"."C1" = "T2"."C1"
   AND ("T1"."C2" = 'A' OR "T1"."C3" > 20)
```

Explanation:

Because condition `"T1"."C1" = "T2"."C1"` is specified on both sides of the `OR` condition, `"T1"."C1" = "T2"."C1"` is removed from the `OR` condition.

(d) Example 4

■ Specified search condition

```
WHERE ("C1" IS NULL AND "C2" = 'A')
   OR ("C1" IS NULL AND "C3" > 20)
```

■ Search condition resulting from equivalent exchange

```
WHERE "C1" IS NULL
   AND ("C2" = 'A' OR "C3" > 20)
```

Explanation:

Because condition `"C1" IS NULL` is specified on both sides of the `OR` condition, `"C1" IS NULL` is removed from the `OR` condition.

(e) Example 5

■ Specified search condition

```
WHERE ("C1" IN(100,200,300) AND "C2" = 'A')
   OR ("C1" IN(100,200,300) AND "C3" > 20)
```

■ Search condition resulting from equivalent exchange

```
WHERE "C1" IN(100,200,300)
   AND ("C2" = 'A' OR "C3" > 20)
```

Explanation:

Because condition `"C1" IN(100,200,300)` is specified on both sides of the `OR` condition, `"C1" IN (100,200,300)` is removed from the `OR` condition.

(f) Example 6

■ Specified search condition

```
WHERE ("C1" BETWEEN 100 AND 300 AND "C2" = 'A')
   OR ("C1" BETWEEN 100 AND 300 AND "C3" > 20)
```

Equivalent exchange

■ Search condition resulting from equivalent exchange

```
WHERE "C1" BETWEEN 100 AND 300
   AND ("C2" = 'A' OR "C3" > 20)
```

Explanation:

Because condition "C1" BETWEEN 100 AND 300 is specified on both sides of the OR condition, "C1" BETWEEN 100 AND 300 is removed from the OR condition.

(g) Example 7

■ Specified search condition

```
WHERE "C1" = 100 OR "C1" = 100 OR "C1" = 100
```

Equivalent exchange

■ Search condition resulting from equivalent exchange

```
WHERE "C1" = 100
```

Explanation:

Because the conditions specified in the OR conditions are all "C1" = 100, they are converted to a single = condition.

(h) Example 8

■ Specified search condition

```
WHERE "C1" <> 100 OR "C1" <> 100 OR "C1" <> 100
```

Equivalent exchange

■ Search condition resulting from equivalent exchange

```
WHERE "C1" <> 100
```

Explanation:

Because the conditions specified in the OR conditions are all "C1" <> 100, they are converted to a single <> condition.

(2) Examples of search conditions on which equivalent exchange is not performed

(a) Example 1

■ Specified search condition

```
WHERE ("T1"."C1" = "T2"."C1" AND "T1"."C2" = 'A')
   OR ("T2"."C1" = "T1"."C1" AND "T1"."C3" > 20)
```

Explanation:

Equivalent exchange is not performed because "T1"."C1" = "T2"."C1" and "T2"."C1" = "T1"."C1" are regarded as different conditions.

(b) Example 2

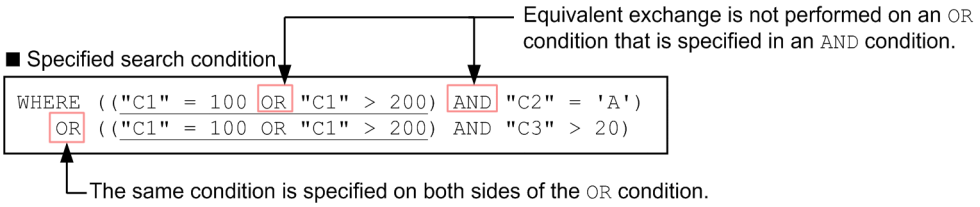
■ Specified search condition

```
WHERE NOT(("C1" = 100 AND "C2" = 'A')
       OR ("C1" = 100 AND "C3" > 20))
```

Explanation:

"C1" = 100 is specified on both sides of the OR condition, but no equivalent exchange is performed because this OR condition is specified inside a NOT condition.

(c) Example 3



Explanation:

"C1" = 100 OR "C1" > 200 is specified on both sides of the OR condition, but equivalent exchange is not performed on an OR condition specified inside an AND condition.

(3) Rules for equivalent exchange

1. If a comparison predicate is in any of the following formats, a condition inside the OR condition is removed from the OR condition:

   - *column-specification comparison-operator literal*

     Equivalent exchange is performed, even if the column specification and literal are specified in reverse order.

   - *column-specification comparison-operator datetime-information-acquisition-function* (or *user-information-acquisition-function*)

     Equivalent exchange is performed, even if the column specification and datetime information acquisition function (or user information acquisition function) are specified in reverse order.

   - *column-specification comparison-operator column-specification*

If the columns specified as the column specification are specified in reverse order, equivalent exchange is not performed because they are regarded as different conditions (see (a) Example 1 in (2) Examples of search conditions on which equivalent exchange is not performed).

2. If the NULL predicates are specified in either of the following formats, the condition in the OR condition is removed from the OR condition:
   - *column-specification* IS NULL
   - *column-specification* IS NOT NULL
3. If the IN predicates are specified in either of the following formats, the condition in the OR condition is removed from the OR condition:
   - *column-specification* IN (*value-expression*,...)

     Equivalent exchange is performed if only literals, datetime information acquisition functions, or user information acquisition functions are specified in the value expressions.
   - *column-specification* NOT IN (*value-expression*,...)

     Equivalent exchange is performed if only literals, datetime information acquisition functions, or user information acquisition functions are specified in the value expressions.
4. If the BETWEEN predicates are specified in either of the following formats, the condition in the OR condition is removed from the OR condition:
   - *column-specification* BETWEEN *value-expression-1* AND *value-expression-2*

     Equivalent exchange is performed if only literals, datetime information acquisition functions, or user information acquisition functions are specified in value expressions 1 and 2.
   - *column-specification* NOT BETWEEN *value-expression-1* AND *value-expression-2*

     Equivalent exchange is performed if only literals, datetime information acquisition functions, or user information acquisition functions are specified in value expressions 1 and 2.
5. Equivalent exchange is not performed in the following cases:
   - OR conditions specified in NOT conditions (see (b) Example 2 in (2) Examples of search conditions on which equivalent exchange is not performed)
   - OR conditions specified in AND conditions (see (c) Example 3 in (2) Examples of search conditions on which equivalent exchange is not performed)

5.11.2 Equivalent exchange for OR conditions (converting to IN conditions)

If multiple = conditions are specified for the same column in an OR condition, the following equivalent exchange is performed:

- The = conditions for the same column are converted to an IN condition.
- The = conditions for the same column are converted to an IN condition, which is added outside of the OR condition.

If the = conditions are converted to an IN condition and then added outside of the OR condition, the resulting condition can be effective in narrowing down the search range; however, the workload for condition evaluation might increase when an IN condition is added.

The OR conditions specified in the search condition in the WHERE clause, the ON search condition for joined tables, and the search condition in the HAVING clause are subject to this equivalent exchange.

If equivalent exchange has been performed on search conditions, the indexes to be used during retrieval are determined based on the search conditions obtained after equivalent exchange.

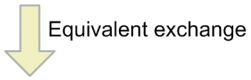
The following shows examples of equivalent exchange. In the examples, C1, C2, and C3 are column names.

(1) Examples of search conditions on which equivalent exchange is performed

(a) Example 1

■ Specified search condition

```
WHERE "C1" = 100 OR "C1" = 200 OR "C1" =300
```



■ Search condition resulting from equivalent exchange

```
WHERE "C1" IN (100,200,300)
```

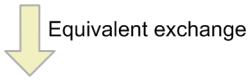
Explanation:

The OR conditions are converted to an IN condition because the conditions specified in the OR conditions are all = conditions for column C1.

(b) Example 2

■ Specified search condition

```
WHERE ("C1" = CURRENT_DATE AND "C2" = 'A')
   OR ("C1" = ? AND "C3" > 20)
```



■ Search condition resulting from equivalent exchange

```
WHERE "C1" IN (CURRENT_DATE,?)
   AND (("C1" = CURRENT_DATE AND "C2" = 'A')
   OR ("C1" = ? AND "C3" > 20))
```

Explanation:

The = condition for column C1 specified in the OR condition is converted to an IN condition, which is then added outside of the OR condition.

(2) Examples of search conditions on which equivalent exchange is not performed

(a) Example 1

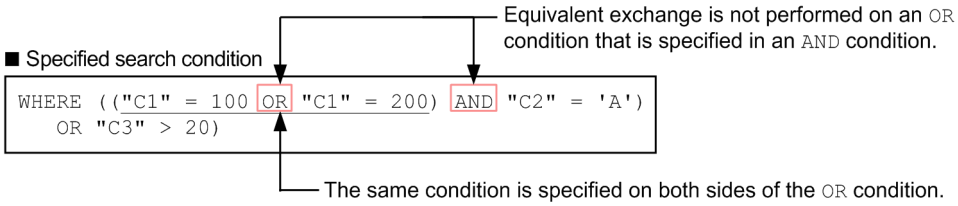
■ Specified search condition

```
WHERE NOT("C1" = 100 OR "C1" = 200 OR "C1" = 300)
```

Explanation:

The conditions specified in the OR conditions are all = conditions for column C1, but equivalent exchange is not performed because they are specified inside a NOT condition.

(b) Example 2



Explanation:

"C1" = 100 and "C1" = 200 are specified on either side of the OR condition, but equivalent exchange is not performed on an OR condition that is specified inside an AND condition.

(3) Rules for equivalent exchange

1. If a comparison predicate is in any of the formats shown below, an IN condition is added outside the OR condition. Note that if a column specified in the column specification is an external reference column, equivalent exchange is not performed. For details about external reference columns, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.
   - *column-specification = literal*

     Equivalent exchange is performed, even if the column specification and literal are specified in reverse order.
   - *column-specification = datetime-information-acquisition-function* (or *user-information-acquisition-function*)

     Equivalent exchange is performed, even if the column specification and datetime information acquisition function (or user information acquisition function) are specified in reverse order.
   - *column-specification = dynamic-parameter*

     Equivalent exchange is performed, even if the column specification and dynamic parameter are specified in reverse order.
2. Equivalent exchange is not performed in the following cases:
   - OR conditions specified in NOT conditions (see (a) Example 1 in (2) Examples of search conditions on which equivalent exchange is not performed)
   - OR conditions specified in AND conditions (see (b) Example 2 in (2) Examples of search conditions on which equivalent exchange is not performed)

5.11.3 Equivalent exchange for OR conditions (equivalent exchange to derived tables for which the UNION ALL set operation is specified)

A specified SQL statement might be subject to equivalent exchange if the SQL statement satisfies the following two conditions:

- A comma join or a joined table is specified in the FROM clause.
- The OR condition is specified in the WHERE clause.

The search conditions specified in the form of an OR condition are converted by equivalent exchange to derived tables for which the UNION ALL set operation is specified. The following explains this equivalent exchange in detail by using examples. In the examples, T1 and T2 are table names, and C1 is a column name.

■ **SQL statement before equivalent exchange (specified SQL statement)**

```
SELECT "T1"."C1" FROM "T1","T2"
  WHERE "T1"."C1" = "T2"."C1"
    AND ("T1"."C1" = 1 OR "T2"."C1" = 2)     ←1
```

In the preceding SQL statement, a comma join is specified in the FROM clause and an OR condition is specified in the WHERE clause (see the underscored portions). Therefore, the SQL statement is subject to equivalent exchange.

■ **SQL statement after equivalent exchange**

```
SELECT "DRVTBL"."C1"
  FROM (
        SELECT "T1"."C1" FROM "T1","T2"
          WHERE "T1"."C1" = "T2"."C1"
            AND "T1"."C1" = 1                ←2
        UNION ALL
        SELECT "T1"."C1" FROM "T1","T2"
          WHERE "T1"."C1" = "T2"."C1"
            AND "T2"."C1" = 2                ←3
            AND (CASE WHEN ("T1"."C1" = 1) THEN 1 ELSE 0 END) <> 1
        )AS DRVTBL("C1")
```

Set operation operands

As shown in the preceding example, the SQL statement is converted by equivalent exchange to derived tables for which the UNION ALL set operation is specified. In the SQL statement before equivalent exchange, search conditions were specified in the form of an OR condition (see 1 in the SQL statement before equivalent exchange). Now, in the SQL statement after equivalent exchange, these search conditions are specified as the search conditions of the query specifications in the set operation operands (see 2 and 3 in the SQL statement after equivalent exchange).

■ **Advantages of equivalent exchange**

For the SQL statement before equivalent exchange, because the WHERE clause contains an OR condition, table joining takes place before search conditions are evaluated.

For the SQL statement after equivalent exchange, however, because the WHERE clauses in the set operation operands contain no OR condition, table joining takes place after search conditions are evaluated. This might be able to reduce the number of input rows that are required for table joining (that is, the search performance might be improved).

If indexes are defined for columns "T1"."C1" and "T2"."C1", any of the indexes are used during a search. The indexes that will be used for a search are determined from the search conditions after equivalent exchange.

Note that, in some cases, the search time might become longer because query specifications and search conditions increase due to equivalent exchange.

(1) Example of equivalent exchange

The following shows examples of equivalent exchange. In the examples, T1, T2, and T3 are table names, and C1 is a column name.

Example 1: Case where two OR conditions are specified in the search conditions

■ SQL statement before equivalent exchange (specified SQL statement)

```
SELECT "T1"."C1",SUM("T2"."C1"),MIN("T3"."C1")
  FROM "T1" INNER JOIN ("T2" INNER JOIN "T3" ON "T2"."C1" = "T3"."C1") ON "T1"."C1" = "T2"."C1"
    WHERE "T1"."C1" = 1 OR "T2"."C1" = 10 OR "T3"."C1" = 100
    GROUP BY "T1"."C1"
```

Equivalent exchange

■ SQL statement after equivalent exchange

```
SELECT "DRVTBL"."C1",SUM("DRVTBL"."C2"),MIN("DRVTBL"."C3")
  FROM(
       SELECT "T1"."C1","T2"."C1","T3"."C1"
         FROM "T1" INNER JOIN ("T2" INNER JOIN "T3" ON "T2"."C1" = "T3"."C1") ON "T1"."C1" = "T2"."C1"
           WHERE "T1"."C1" = 1
       UNION ALL
       SELECT "T1"."C1","T2"."C1","T3"."C1"
         FROM "T1" INNER JOIN ("T2" INNER JOIN "T3" ON "T2"."C1" = "T3"."C1") ON "T1"."C1" = "T2"."C1"
           WHERE "T2"."C1" = 10
             AND (CASE WHEN ("T1"."C1" = 1) THEN 1 ELSE 0 END) <> 1
       UNION ALL
       SELECT "T1"."C1","T2"."C1","T3"."C1"
         FROM "T1" INNER JOIN ("T2" INNER JOIN "T3" ON "T2"."C1" = "T3"."C1") ON "T1"."C1" = "T2"."C1"
           WHERE "T3"."C1" = 100
             AND (CASE WHEN ("T1"."C1" = 1 OR "T2"."C1" = 10) THEN 1 ELSE 0 END) <> 1
       )AS DRVTBL("C1","C2","C3")
  GROUP BY "DRVTBL"."C1"
```

Set operation operand

Set operation operand

Set operation operand

Explanation:

- In the SQL statement before equivalent exchange, a joined table is specified in the FROM clause and OR conditions are specified in the WHERE clause (see the underscored portion in the preceding example). Therefore, the SQL statement is subject to equivalent exchange.
- The search conditions that were specified with OR conditions in the SQL statement before equivalent exchange are, after equivalent exchange, specified as search conditions of query specifications in separate set operation operands. If two OR conditions are specified as shown in the SQL statement before equivalent exchange, three set operation operands are generated after equivalent exchange.

Example 2: Case where AND and OR conditions are specified in the search conditions

■ SQL statement before equivalent exchange (specified SQL statement)

```
SELECT "T1"."C1",SUM("T2"."C2") FROM "T1","T2"
  WHERE "T1"."C1" = "T2"."C1" AND ("T1"."C2" = 1 OR "T2"."C2" = 10)
  GROUP BY "T1"."C1"
```

Equivalent exchange

■ SQL statement after equivalent exchange

```
SELECT "DRVTBL"."C1",SUM("DRVTBL"."C2")
  FROM (
       SELECT "T1"."C1","T2"."C2" FROM "T1","T2"
         WHERE "T1"."C1" = "T2"."C1"
           AND "T1"."C2" = 1
       UNION ALL
       SELECT "T1"."C1","T2"."C2" FROM "T1","T2"
         WHERE "T1"."C1" = "T2"."C1"
           AND "T2"."C2" = 10
           AND (CASE WHEN ("T1"."C2" = 1) THEN 1 ELSE 0 END) <> 1
       )AS DRVTBL("C1","C2")
  GROUP BY "DRVTBL"."C1"
```

Set operation operand

Set operation operand

Explanation:

- In the SQL statement before equivalent exchange, a comma join is specified in the FROM clause and an OR condition is specified in the WHERE clause (see the underscored portions). Therefore, the SQL statement is subject to equivalent exchange.
- The search conditions that were specified with OR conditions in the SQL statement before equivalent exchange are, after equivalent exchange, specified as search conditions of query specifications in separate set operation operands. The search condition that was specified on the left side of the AND condition in the SQL statement before equivalent exchange are, after equivalent exchange, specified as search conditions of query specifications in all set operation operands.

(2) Conditions for equivalent exchange to take place

For equivalent exchange to take place, all of the following conditions must be met.

■ Conditions for the server definition or client definition

All of the following conditions must be met:

- 0 is not specified for the adb_sql_exe_hashtbl_area_size operand in the server definition or client definition.
- 0 is not specified for the adb_sys_uthd_num operand in the server definition.
- 0 is not specified for the adb_sql_exe_max_rthd_num operand in the server definition or client definition.

■ Conditions for SQL statements

All of the following conditions must be met:

- The SQL statement is a retrieval SQL statement.
- An HADB server whose version is 04-03 or later has collected cost information from any of tables specified in the SQL statement.
- The SQL statement does not have a query specification that includes an external reference column.

■ Conditions for query specifications

All of the following conditions must be met:

1. Conditions for selection expressions
   - ROW is not specified in the selection expression.
2. Conditions for a table reference specified in the FROM clause
   - The number of tables specified in the selection expression does not exceed 16.
   - Comma joins are specified in the FROM clause.

     Note that this condition is met if only comma joins are specified or if both comma joins and joined tables are specified.

     The following shows the types of tables that can be specified. This condition is met even if the specifications of the following types of tables co-exist:

     • Base table (Note that equivalent exchange does not take place if an archivable multi-chunk table is specified.)

     • Joined table (Note that equivalent exchange does not take place if FULL OUTER JOIN is specified.)

     • Derived table (A table subquery, query name, or viewed table applies.)

■ Conditions for join conditions

All of the following conditions must be met:

- No OR condition is specified in the join specifications of joined tables.

- The join conditions of tables to which the columns specified in OR conditions belong satisfy the following condition:
  - At least one join condition in the *column-specification=column-specification* format exists between the tables to which the columns specified in OR conditions.
- When HADB performs equivalent exchange to derived tables for which the UNION ALL set operation is specified, a join condition for all join-target tables must exist in the search condition of the query specification in each set operation operand.

■ Conditions for the search conditions specified in the WHERE clause

All of the following conditions must be met:

1. Two or more conditions are not specified in OR conditions in the conditions specified by using the AND logical operator.

   Examples:

   - Example of a search condition that satisfies the conditions for equivalent exchange to take place

     ```
     condition-1 AND condition-2 AND (condition-3 OR condition-4 OR condition-5)
     ```

     The conditions for equivalent exchange to take place are satisfied because there is only one condition that includes OR conditions.

   - Example of a search condition that does not satisfy the conditions for equivalent exchange to take place

     ```
     (condition-1 OR condition-2) AND condition-3 AND (condition-4 OR condition-5)
     ```

     The conditions for equivalent exchange to take place are not satisfied because there are two conditions that include OR conditions.

   - If multiple OR conditions are nested, the conditions for equivalent exchange to take place are satisfied.
   - The number of tables specified in successive search conditions other than OR conditions is not taken into account (the specifications of these tables do not need to be join specifications).
   - An OR condition with NOT specified does not satisfy the conditions for equivalent exchange to take place.

2. Columns of joined tables are not specified in OR conditions.

   However, the conditions for equivalent exchange to take place are satisfied if a joined table is converted to a comma join.

3. No subquery is specified with an OR condition.
4. Scalar function RANDOM or RANDOM_NORMAL is not specified with an OR condition.
5. The number of OR logical operators does not exceed 15.
6. An OR condition includes a search condition for two or more tables.
7. The search condition in each set operation operand after equivalent exchange includes one or more search conditions in any of the following formats. In addition, logical operator OR or NOT is not specified for those conditions.

   - Comparison predicate

     ```
     • column-specification comparison-operator {literal|dynamic-parameter|datetim
     e-information-acquisition-function|user-information-acquisition-function}
     • {literal|dynamic-parameter|datetime-information-acquisition-function|user-i
     nformation-acquisition-function} comparison-operator column-specification
     ```

   - BETWEEN predicate

```
column-specification BETWEEN {literal|dynamic-parameter|datetime-information-
acquisition-function|user-information-acquisition-function}
          AND {literal|dynamic-parameter|datetime-information-acquisition-fu
nction|user-information-acquisition-function}
```

- IN predicate

```
column-specification IN ({literal|dynamic-parameter|datetime-information-acqu
isition-function|user-information-acquisition-function}
        [,{literal|dynamic-parameter|datetime-information-acquisition-functio
n|user-information-acquisition-function}]...)
```

- LIKE predicate

```
column-specification LIKE pattern-character-string [ESCAPE escape-character]
  pattern-character-string ::= {literal|dynamic-parameter|datetime-informatio
n-acquisition-function|user-information-acquisition-function}
  escape-character ::= {literal|dynamic-parameter}
```

- LIKE_REGEX predicate

```
column-specification LIKE_REGEX regular-expression-character-string [FLAG {I|
IGNORECASE}]
  regular-expression-character-string ::= literal
```

- NULL predicate

```
column-specification IS NULL
```

Note

If only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

■ Conditions for the locations of query specifications subject to equivalent exchange

If a query specification is included in the following locations, the HADB server checks whether to perform equivalent exchange:

- The query expression body in a query expression
- Scalar subquery
- Table subquery
- Set operation operands of a set operation

  If derived tables with UNION ALL specified are created when equivalent exchange is performed for an SQL statement, the maximum number of set operations might be exceeded. In such a case, however, equivalent exchange of the SQL statement continues.
- WITH list element

  Equivalent exchange is not performed for query specifications of recursive members.
- Query expression in the CREATE VIEW statement

5.11.4 Equivalent exchange for scalar operations

When one of the terms of a search condition specifies a scalar operation that contains a column specification, the scalar operation is transposed. That is, the search condition undergoes equivalent exchange so as to leave only a column specification. A scalar operation is transposed when all of the following conditions are met:

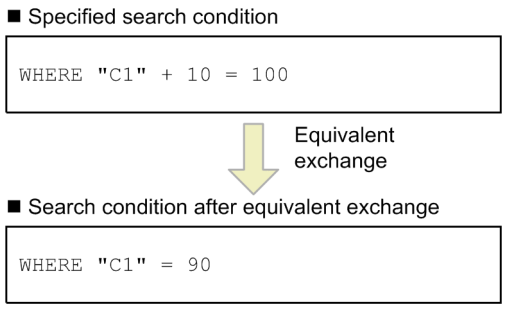
- The operand satisfies either of the following conditions:
  - The operator is an arithmetic operator (addition or subtraction) between a column specification and a literal.
  - The operator is a datetime operator between a column specification and a labeled duration.
- The data type of the column specification is `SMALLINT`, `INTEGER`, `TIME`, `DATE`, or `TIMESTAMP`.
- The scalar operation is not nested.

When a search condition is subjected to equivalent exchange, the indexes to be used during retrieval are determined based on the search conditions after equivalent exchange.

The following shows examples of equivalent exchange. In the examples, `C1` is a column name.

(1) Examples where equivalent exchange is performed

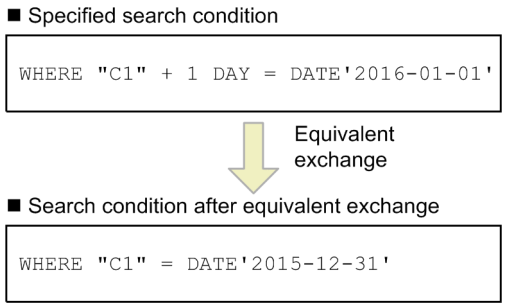
(a) Example 1



Explanation:

A term that includes a column specification has the scalar operation `+10`. By equivalent exchange, this scalar operation is moved to the right-hand side, leaving the condition with just a column specification. Because the format after equivalent exchange is column-specification comparison-operator literal, an index will be used during retrieval.

(b) Example 2



Explanation:

A term that includes a column specification has the scalar operation `+1 DAY`. By equivalent exchange, this scalar operation is moved to the right-hand side, leaving just a column specification in the condition. Because the format after equivalent exchange is column-specification comparison-operator literal, an index will be used during retrieval.

(c) Example 3

■ Specified search condition

```
WHERE "C1" + 10 BETWEEN 50 AND 100
```

Equivalent exchange

■ Search condition after equivalent exchange

```
WHERE "C1" BETWEEN 40 AND 90
```

Explanation:

A term that includes a column specification has the scalar operation `+10`. By equivalent exchange, this scalar operation is moved to the right-hand side, leaving just a column specification in the condition. Because the format after equivalent exchange is column-specification `BETWEEN` literal, an index will be used during retrieval.

(d) Example 4

■ Specified search condition

```
WHERE "C1" + 10 IN(100,200,300)
```

Equivalent exchange

■ Search condition after equivalent exchange

```
WHERE "C1" IN(90,190,290)
```

Explanation:

A term that includes a column specification has the scalar operation `+10`. By equivalent exchange, this scalar operation is moved to the right-hand side, leaving just a column specification in the condition. Because the format after equivalent exchange is column-specification `IN` literal, an index will be used during retrieval.

(2) Examples where equivalent exchange is not performed

(a) Example 1

■ Specified search condition

```
WHERE ("C1" + 10) + 50 = 100
```

Explanation:

Equivalent exchange does not take place because the scalar operation that includes the column specification is nested.

(3) Rules for equivalent exchange

1. When a comparison predicate is in any of the following formats, equivalent exchange is performed by transposing the scalar operation to leave only the column specification.
   - scalar-operation-containing-column-specification comparison-operator literal
   - literal comparison-operator scalar-operation-containing-column-specification

   Equivalent exchange does not occur if all of the following conditions are met:
   - The scalar operation that includes the column specification specifies `YEAR` or `MONTH` as a labeled duration.
   - The comparison operator of the comparison predicate is <>, `!=`, or `^=`.
2. When a `BETWEEN` predicate is used in any of the following formats, equivalent exchange is performed by transposing the scalar operation to leave only the column specification in the condition.
   - scalar-operation-including-column-specification `BETWEEN` literal `AND` literal
   - scalar-operation-including-column-specification `NOT BETWEEN` literal `AND` literal

   Equivalent exchange does not occur if all of the following conditions are met:
   - The scalar operation that includes the column specification specifies `YEAR` or `MONTH` as a labeled duration.
   - `NOT BETWEEN` is specified in the `BETWEEN` predicate.
3. When an `IN` predicate is used in any of the following formats, equivalent exchange is performed by transposing the scalar operation to leave only the column specification in the condition.
   - scalar-operation-including-column-specification `IN` (literal,...)
   - scalar-operation-including-column-specification `NOT IN` (literal,...)

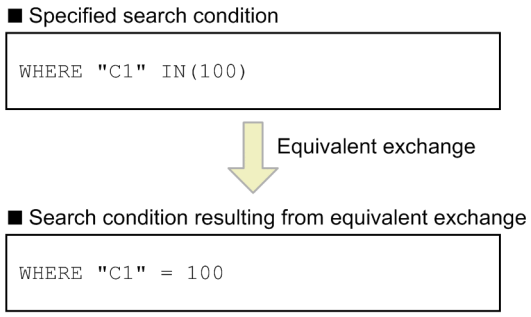
   Equivalent exchange will not take place if the scalar operation that includes the column specification specifies `YEAR` or `MONTH` as a labeled duration.

5.11.5 Equivalent exchange for an IN predicate

If only one comparison value is specified in the `IN` condition, equivalent exchange is performed on the specified search condition. If equivalent exchange has been performed on search conditions, the indexes to be used during retrieval are determined based on the search conditions obtained after equivalent exchange.

The following shows examples of equivalent exchange. In the examples, `C1` is a column name.

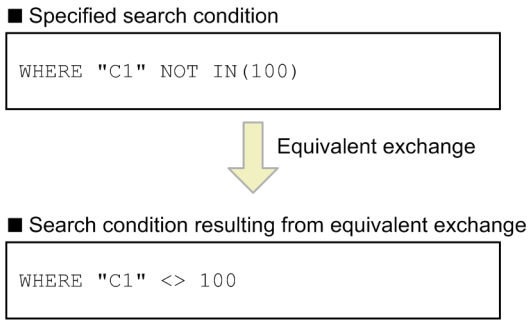
(1) Example 1



Explanation:

If only one comparison value is specified in the IN condition, the search condition is converted to the = condition.

(2) Example 2



Explanation:

If only one comparison value is specified in the NOT IN condition, the search condition is converted to the <> condition.

5.11.6 Equivalent exchange for a HAVING clause (converting to the WHERE clauses)

The search condition in a HAVING clause might be converted to a search condition in the WHERE clause. This equivalent exchange might enable unnecessary input rows to be deleted during grouping and indexes to be used for table retrieval processing.

The formats of conditions that are converted to a search condition in the WHERE clause are shown below. If equivalent exchange occurs for a search condition, the indexes to be used during retrieval processing are determined on the basis of the search condition obtained after the equivalent exchange.

Format of conditions subject to equivalent exchange

- Comparison predicate

```
• column-specification comparison-operator {literal|dynamic-parameter|datetime-inf
ormation-acquisition-function|user-information-acquisition-function}

• {literal|dynamic-parameter|datetime-information-acquisition-function|user-inform
ation-acquisition-function} comparison-operator column-specification
```

- BETWEEN predicate

```
column-specification [NOT] BETWEEN {literal|dynamic-parameter|datetime-information
-acquisition-function|user-information-acquisition-function}
                    AND {literal|dynamic-parameter|datetime-information-acquisitio
n-function|user-information-acquisition-function}
```

- IN predicate

```
column-specification [NOT] IN ({literal|dynamic-parameter|datetime-information-acq
uisition-function|user-information-acquisition-function}
                  [,{literal|dynamic-parameter|datetime-information-acquisition-fu
nction|user-information-acquisition-function}]...)
```

- LIKE predicate

```
column-specification [NOT] LIKE pattern-character-string [ESCAPE escape-character]
  pattern-character-string::={literal|dynamic-parameter|datetime-information-acqui
sition-function|user-information-acquisition-function}
  escape character ::={literal|dynamic-parameter}
```

- LIKE_REGEX predicate

```
column-specification [NOT] LIKE_REGEX regular-expression-character-string [FLAG {I
|IGNORECASE}]
  regular-expression-character-string::=literal
```

- NULL predicate

```
column-specification IS [NOT] NULL
```

Notes

- Conditions specified in an OR condition of a logical operation are not subject to this equivalent exchange.

  However, if the following equivalent exchange is applied to the OR condition of the logical operation , after the following equivalent exchange is applied, equivalent exchange related to the HAVING clause will take place.

  • Equivalent exchange that moves a condition from inside the OR condition to outside the OR condition

  • Equivalent exchange that adds the IN condition created from an = condition in the OR condition to outside the OR condition

  For details about equivalent exchange for the OR condition, see 5.11.1 Equivalent exchange for OR conditions (removing from the OR conditions) and 5.11.2 Equivalent exchange for OR conditions (converting to IN conditions).
- Conditions specified in NOT in logical operations and conditions containing subqueries are not subject to this equivalent exchange.
- If column-specification columns are external reference columns, this equivalent exchange is not applied. For details about external reference columns, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.
- If only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about scalar operations equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

5.11.7 Equivalent exchange for search conditions in SQL statements that specify derived queries (transposition to the WHERE clause of a derived query)

Equivalent exchange is performed to move a search condition specified in the WHERE clause of an SQL statement that specifies a derived query to the WHERE clause of the derived query. For details about derived queries, see *Derived queries and derived query names* in the manual *HADB SQL Reference*.

The following shows examples of equivalent exchange. In the examples, C1, C2, and C3 are column names.

(1) Examples where equivalent exchange is performed

(a) Example 1 (when derived query is a query specification)

■ Specified SQL statement

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT DISTINCT "C1","C2","C3"
          FROM "T1") "D1"("C1","C2","C3")
    WHERE "D1"."C1"<10
```

■ SQL statement after equivalent exchange

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT DISTINCT "C1","C2","C3"
          FROM "T1" WHERE "T1"."C1"<10) "D1"("C1","C2","C3")
```

Explanation:

The column specification on the left side of the `WHERE` clause of the SQL statement that specifies the derived query is a derived column (`"D1"."C1"`). This column is derived based on the selection expression (`"T1"."C1"`) of the query specification in the derived query. Because the conditions for applying equivalent exchange are satisfied, equivalent exchange is applied to the search condition in the SQL statement that specifies the derived query.

(b) Example 2 (when derived query is a query expression)

■ Specified SQL statement

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT "C1","C2","C3" FROM "T1"
        UNION DISTINCT
        SELECT "C1","C2","C3" FROM "T2") "D1"("C1","C2","C3")
    WHERE "D1"."C1"<10
```

■ SQL statement after equivalent exchange

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT "C1","C2","C3" FROM "T1"
          WHERE "T1"."C1"<10
        UNION DISTINCT
        SELECT "C1","C2","C3" FROM "T2"
          WHERE "T2"."C1"<10) "D1"("C1","C2","C3")
```

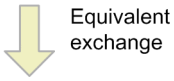
Explanation:

The column specification on the left side of the `WHERE` clause of the SQL statement that specifies the derived query is a derived column (`"D1"."C1"`). This column is derived based on the selection expression (`"T1"."C1","T2"."C1"`) of the query expression in the derived query. Because the conditions for applying equivalent exchange are satisfied, equivalent exchange is applied to the search condition in the SQL statement that specifies the derived query.

(c) Example 3 (when derived query is a query specification)

■ Specified SQL statement

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT DISTINCT "C1","C2","C3" FROM "T1") "D1"("C1","C2","C3")
    WHERE "D1"."C1"<10 OR ("D1"."C2">100 AND "D1"."C3"=200)
```



■ SQL statement after equivalent exchange

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT DISTINCT "C1","C2","C3" FROM "T1"
          WHERE "T1"."C1"<10 OR ("T1"."C2">100 AND "T1"."C3"=200)
        ) D1"("C1","C2","C3")
```

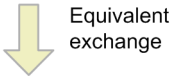
Explanation:

The column specification in the `OR` condition of the logical operation specified in the `WHERE` clause of the SQL statement that specifies the derived query is a derived column (`"D1"."C1","D1"."C2","D1"."C3"`). This column is derived based on the selection expression (`"T1"."C1","T1"."C2","T1"."C3"`) of the query specification in the derived query. Because the `OR` condition of the logical operation satisfies the conditions for applying equivalent exchange, equivalent exchange is applied to the search condition in the SQL statement that specifies the derived query.

(d) Example 4 (when derived query is a query expression)

■ Specified SQL statement

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT "C1","C2","C3" FROM "T1"
        UNION DISTINCT
        SELECT "C1","C2","C3" FROM "T2"
        ) "D1"("C1","C2","C3")
    WHERE "D1"."C1"<10 OR ("D1"."C2">100 AND "D1"."C3"=200)
```



■ SQL statement after equivalent exchange

```
SELECT "C1","C1","C2","C3"
  FROM (SELECT "C1","C2","C3" FROM "T1"
          WHERE "T1"."C1"<10 OR ("T1"."C2">100 AND "T1"."C3"=200)
        UNION DISTINCT
        SELECT "C1","C2","C3" FROM "T2"
          WHERE "T2"."C1"<10 OR ("T2"."C2">100 AND "T2"."C3"=200)
        ) "D1"("C1","C2","C3")
```

Explanation:

The column specification in the `OR` condition of the logical operation specified in the `WHERE` clause of the SQL statement that specifies the derived query is a derived column (`"D1"."C1","D1"."C2","D1"."C3"`). This column is derived based on the selection expression (`"T1"."C1","T1"."C2","T1"."C3","T2"."C1","T2"."C2","T2"."C3"`) of the query expression in the derived query. Because the `OR` condition of the logical operation satisfies the conditions for applying equivalent exchange, equivalent exchange is applied to the search condition in the SQL statement that specifies the derived query.

(2) Format of conditions subject to equivalent exchange

The following shows the format of conditions that are subject to equivalent exchange:

- `Comparison` predicate

```
▪ column-specification comparison-operator {literal|dynamic-parameter|datetime-inf
ormation-acquisition-function|user-information-acquisition-function}
▪ {literal|dynamic-parameter|datetime-information-acquisition-function|user-inform
ation-acquisition-function} comparison-operator column-specification
▪ CONTAINS (column-specification,search-condition-expression-string) > 0
```

- `BETWEEN` predicate

```
column-specification [NOT] BETWEEN {literal|dynamic-parameter|datetime-information
-acquisition-function|user-information-acquisition-function}
                    AND {literal|dynamic-parameter|datetime-information-acquisition
-function|user-information-acquisition-function}
```

- `IN` predicate

```
column-specification [NOT] IN ({literal|dynamic-parameter|datetime-information-acq
uisition-function|user-information-acquisition-function}
                 [,{literal|dynamic-parameter|datetime-information-acquisition-func
tion|user-information-acquisition-function}]...)
```

- `LIKE` predicate

```
column-specification [NOT] LIKE pattern-character-string [ESCAPE escape-character]
  pattern-character-string::={literal|dynamic-parameter|datetime-information-acqui
sition-function|user-information-acquisition-function}
  escape-character::={literal|dynamic-parameter}
```

- `LIKE_REGEX` predicate

```
column-specification [NOT] LIKE_REGEX regular-expression-character-string [FLAG {I
|IGNORECASE}]
  regular-expression-character-string::=literal
```

- `NULL` predicate

```
column-specification IS [NOT] NULL
```

(3) Notes

- Equivalent exchange will be performed if the selection expression of the derived query on which the derived column specified in the column specification of the search condition is based is a column specification.
- Equivalent exchange is not performed if a window function is specified in the derived query.
- Equivalent exchange is not performed if the derived query is expanded into the SQL statement that specifies the derived query. For details about the expansion of derived queries, see *Internal derived tables* in the manual *HADB SQL Reference*.
- Equivalent exchange is not performed for the following types of derived queries:
  - The derived query specifies the name of a viewed table, and the same viewed table name is specified more than once in the SQL statement.
  - The derived query specifies the name of a query specified as a `WITH` list element, and the same query name is specified more than once in the SQL statement.

- Multiple WITH list elements are specified, and the derived query specifies a query name that is already specified as a WITH list element.
- The derived query is specified in a joined table other than a joined table for which only INNER JOIN is specified its join type.
- A table value constructor is specified as the derived query.
- The derived query is a recursive query.

- When all of the following conditions are satisfied, equivalent exchange is applied to the search condition in the SQL statement that specifies the derived query:
  - A condition specified as an OR condition in a logical operation meets the format requirements for a search condition that is subject to equivalent exchange for search conditions in SQL statements that specify derived queries
  - All column specifications point to columns in the same derived table
- When applying the following equivalent exchange to the OR condition of a logical operation, the following equivalent exchange is applied first. Then, equivalent exchange for search conditions in SQL statements that specify derived queries is applied.
  - Equivalent exchange that moves a condition from inside the OR condition to outside the OR condition
  - Equivalent exchange that adds the IN condition created from an = condition in the OR condition to the outside of the OR condition

  For details about equivalent exchange described earlier, see 5.11.1 Equivalent exchange for OR conditions (removing from the OR conditions) and 5.11.2 Equivalent exchange for OR conditions (converting to IN conditions).
- Conditions specified in NOT conditions in logical operations and conditions that contain subqueries are not subject to equivalent exchange.
- Equivalent exchange is not performed if the columns specified by column-specification are external reference columns. For details about external reference columns, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.
- If only literals are specified in a scalar operation, that scalar operation might be treated as a literal. For details about the conditions under which scalar operations are equivalent to literals, see *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

5.12 Considerations when searching an archivable multi-chunk table

This section explains the matters you need to consider when searching an archivable multi-chunk table.

The examples in this section use a table in which the archive range column contains datetime data in `DATE` format.

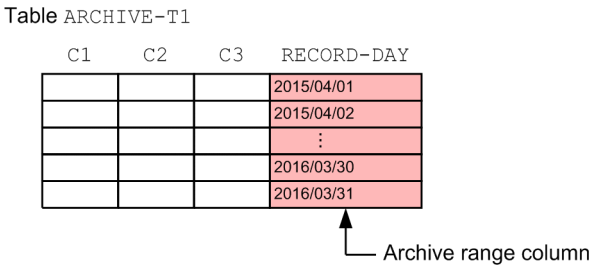
> **Note**
>
> Make sure that you read *Chunk archiving function (compressing data in a chunk)* in the *HADB Setup and Operation Guide* before reading this section.

5.12.1 Tips for searching an archivable multi-chunk table

This subsection explains by way of examples the process of searching an archivable multi-chunk table.

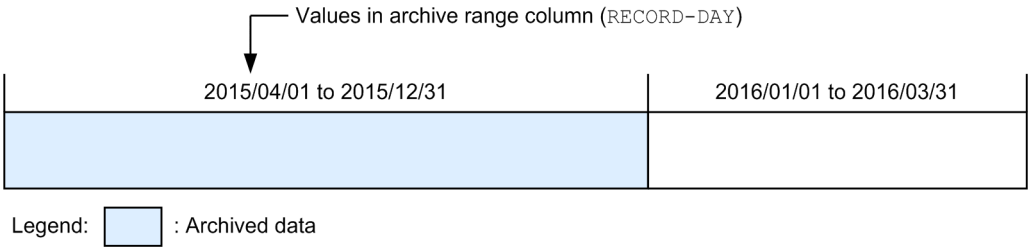
The definition of the archivable multi-chunk table used in these examples and the archived state of the data are as follows:

▪ Definition of archivable multi-chunk table



▪ Archived state of data

- The database stores data from April 2015 to March 2016.
- The data from April to December 2015 is archived.
- The data from January to March 2016 is not archived.



(1) Basic concept when searching archivable multi-chunk tables

When searching an archivable multi-chunk table, you need to narrow the scope of the search by specifying the datetime information for the archive range column as a search condition.

You also need to be aware of whether the data you are searching for is archived. Search processing can take longer when searching archived data.

(2) Specifying search conditions

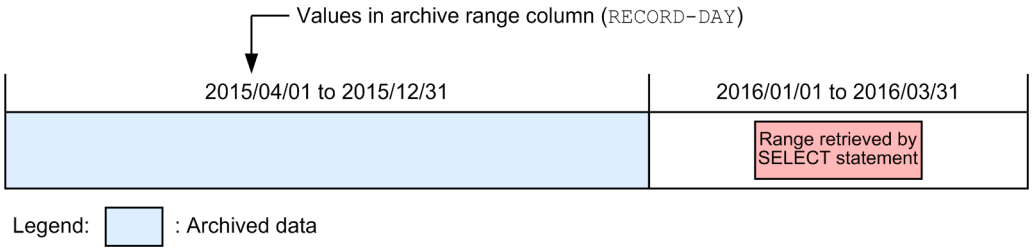
The following are the key considerations when specifying search conditions to search an archivable multi-chunk table:

- Narrow down the search range by specifying a condition that specifies the archive range column as a search condition in a `WHERE` clause.

  Example:

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2016/02/01' AND DATE'2016/02/29'
```

  You must specify the underlined portion. This narrows the search range using the datetime information for the archive range column.



  Note that restrictions apply to the predicates you can specify. For details, see 5.12.2 Using the datetime information of the archive range column to narrow the search range.

- Narrow the search range further by adding `AND` conditions.

  Example:

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2016/02/01' AND DATE'2016/02/29'
    AND "C1"='P001'
    AND "C2"=100
```

- We recommend that you specify literals as the comparison conditions for the archive range column.

  Example: Example of recommended specification

```
WHERE "RECORD-DAY" BETWEEN DATE'2016/01/01' AND DATE'2016/03/31'
WHERE "RECORD-DAY" >= DATE'2016/02/01'
```

(3) When searching unarchived data

When searching unarchived data, specify search conditions a way that limits the search range to the unarchived data. In this example, the unarchived data is data from January 2016.

Example:

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" >= DATE'2016/01/01'
    AND "C1"='P001'
    AND "C2"=100
```

The search condition specified in this example restricts the search range to data from January 1st, 2016.

(4) When searching archived data

When searching archived data, the datetime information specified for the archive range column in the search condition preferably references as narrow a range as possible. In this example, the archived data is data from April 2015 to December 2015.

Example:

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2015/10/01' AND DATE'2015/10/05'
    AND "C1"='P001'
    AND "C2"=100
```

Narrow the search range as much as possible by specifying the underlined portion. This reduces the number of archive files that need to be read. The search time increases in proportion to the number of archive files HADB has to read.

5.12.2 Using the datetime information of the archive range column to narrow the search range

When searching an archivable multi-chunk table, you need to narrow the scope of the search by using the datetime information for the archive range column when specifying the search conditions.

Important

If you do not comply with the rules described here, searches will target all archived data and take longer as a result. The KFAA51121-W message will be output if you run a search that targets all archived data. In this situation, amend the SQL statement, and then narrow the search range by using the datetime information of the archive range column.

(1) Rules for specifying search conditions

The datetime information of the archive range column is used to narrow the search range under the following conditions:

- A condition that specifies the archive range column is specified as a search condition in a WHERE clause.
- In the search condition that specifies the archive range column, only a comparison predicate, IN predicate, or BETWEEN predicate is specified.
- The comparison predicate, IN predicate, or BETWEEN predicate is specified in a way that meets the conditions described in (2) Comparison predicate specification rules and the subsequent subsections.
- There is no NOT condition in the search condition that specifies the archive range column.
- There is no OR condition in the search condition that specifies the archive range column (a search condition that specifies the archive range column is not specified on either side of an OR condition).

  Example where search range is not narrowed:

```
SELECT * FROM "ARCHIVE-T1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/01' AND DATE'2016/01/10'
     OR "RECORD-DAY" BETWEEN DATE'2016/02/01' AND DATE'2016/02/10'
```

  Example where search range is narrowed:

```
SELECT * FROM "ARCHIVE-T1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/01' AND DATE'2016/01/10'
  AND ("C1"='P001' OR "C2"='S002')
```

Important

Search conditions specified in DELETE and UPDATE statements are also subject to the specification rules described here. DELETE and UPDATE statements that do not comply with these specification rules generate errors.

For details about the rules for DELETE statements that delete rows in archivable multi-chunk tables, see *Rules* under *Specification format and rules for the DELETE statement* in the manual *HADB SQL Reference*.

For details about the rules for UPDATE statements that update rows in archivable multi-chunk tables, see *Rules* under *Specification format and rules for the UPDATE statement* in the manual *HADB SQL Reference*.

(2) Comparison predicate specification rules

When you specify a comparison predicate that complies with the following specification rules, the search range is narrowed using the datetime information of the archive range column.

(a) Specification rules and example of recommended specification

Specification format of comparison predicate

```
comparison-predicate::=comparison-operand-1 comparison-operator comparison-operand
-2
```

Specification rules

- =, <, <=, >=, or > is specified as the comparison operator.
- One of the comparison operands specifies the archive range column (as a single column specification).
- The opposite comparison operand specifies a value specification.

Note

We recommend that you specify a literal in the value specification.

Example of recommended specification

```
SELECT * FROM "ARCHIVE-T1" WHERE "RECORD-DAY" >= DATE'2016/01/01'
```

Examples of deprecated specification

```
SELECT * FROM "ARCHIVE-T1" WHERE "RECORD-DAY" = ?
SELECT * FROM "ARCHIVE-T1" WHERE "RECORD-DAY" >= CURRENT_DATE
```

Although these examples result in narrowing of the search range, use of non-literals is not recommended.

Important

Specifying a literal in the value specification results in faster narrowing of the search range than if a non-literal were specified.

(b) Specification examples where search range is not narrowed

Example 1

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" - 10 DAY > DATE'2016/02/10'
```

In this example, the search range is not narrowed because a comparison operand specifies a datetime operation that uses the archive range column. That is, it is not a single column specification. In this case, you can narrow the search range by amending the SELECT statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" > DATE'2016/02/10' + 10 DAY
```

In this example, the archive range column is specified as a single column specification in the comparison operand on one side. The comparison operand on the other side specifies a value expression equivalent to a literal.

Example 2

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" > CURRENT_DATE - 1 YEAR
```

In this example, the search range is not narrowed because the comparison operand does not specify a value specification. In this case, you can narrow the search range by amending the SELECT statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" > DATE'2016/02/10' - 1 YEAR
```

As in this example, specify CURRENT_DATE explicitly as a literal. The search range will then be narrowed because the comparison operand on the right side is a value expression equivalent to a literal.

For details about value expressions equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

(3) IN predicate specification rules

When you specify an IN predicate that complies with the following specification rules, the search range is narrowed using the datetime information of the archive range column.

(a) Specification rules and example of recommended specification

Specification format of IN predicate

```
IN predicate::=value-expression-1 [IS] [NOT] IN {(value-expression-2[,value-expres
sion-3]...)|table-subquery}
```

Specification rules

- *value-expression-1* specifies the archive range column (as a single column specification).
- *value-expression-2* and subsequent value expressions specify value specifications.
- There is no table subquery specified in the IN predicate.
- NOT is not specified in the IN predicate.

Note

We recommend that you specify literals for the value specifications specified in *value-expression-2* onward.

Example of recommended specification

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" IN (DATE'2016/01/01',DATE'2016/02/01')
```

Example of deprecated specification

```
SELECT * FROM "ARCHIVE-T1" WHERE "RECORD-DAY" IN (?,?)
```

Although this example results in narrowing of the search range, use of non-literals is not recommended.

Important

Specifying a literal in the value specification results in faster narrowing of the search range than if a non-literal were specified.

(b) Specification examples where search range is not narrowed

Example 1

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" IN (CURRENT_DATE,
                           CURRENT_DATE - 7 DAY,
                           CURRENT_DATE - 14 DAY)
```

In this example, the search range is not narrowed because a condition other than a value specification is specified in *value-expression-2* and later in an `IN` predicate. In this case, you can narrow the search range by amending the `SELECT` statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" IN (DATE'2016/02/10',
                           DATE'2016/02/10' - 7 DAY,
                           DATE'2016/02/10' - 14 DAY)
```

As in this example, specify `CURRENT_DATE` explicitly as a literal. The search range will then be narrowed because the value expressions specified in *value-expression-2* and later in the `IN` predicate are equivalent to literals.

For details about value expressions equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

Example 2

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" IN (SELECT "SALES_DATE" FROM "SALESLIST"
                              WHERE "USERID" = 'U001')
```

In this example, the search range is not narrowed because a table subquery is specified in an `IN` predicate. In this case, you can narrow the search range by amending the `SELECT` statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" IN (SELECT "SALES_DATE" FROM "SALESLIST"
                           WHERE "USERID" = 'U001')
      AND "RECORD-DAY" > DATE'2015/10/01'
```

As in this example, add a search condition that results in narrowing of the search range.

(4) BETWEEN predicate specification rules

When you specify a BETWEEN predicate that complies with the following specification rules, the search range is narrowed using the datetime information of the archive range column.

(a) Specification rules and example of recommended specification

Specification format of BETWEEN predicate

```
BETWEEN predicate::=value-expression-1 [NOT] BETWEEN value-expression-2 AND value-
expression-3
```

Specification rules

- *value-expression-1* specifies the archive range column (as a single column specification).
- *value-expression-2* and *value-expression-3* specify value specifications.
- NOT is not specified in the BETWEEN predicate.

> **Note**
>
> We recommend that you specify literals for the value specifications in *value-expression-2* and *value-expression-3*.

Example of recommended specification

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2016/01/01' AND DATE'2016/01/10'
```

Example of deprecated specification

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN ? AND ?
```

Although this example results in narrowing of the search range, use of non-literals is not recommended.

> **Important**
>
> Specifying a literal in the value specification results in faster narrowing of the search range than if a non-literal were specified.

(b) Specification examples where search range is not narrowed

Example 1

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN CURRENT_DATE - 2 YEAR
                           AND CURRENT_DATE - 1 YEAR
```

In this example, the search range is not narrowed because a condition other than a value specification is specified as *value-expression-2* or *value-expression-3* of the BETWEEN predicate. In this case, you can narrow the search range by amending the SELECT statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2016/02/10' - 2 YEAR
                           AND DATE'2016/02/10' - 1 YEAR
```

As in this example, specify CURRENT_DATE explicitly as a literal. The search range will then be narrowed because *value-expression-2* or *value-expression-3* in the BETWEEN predicate is a value expression that is equivalent to a literal.

For details about value expressions equivalent to literals, see the table *Conditions under which value expressions are equivalent to literals* under *Rules* in *Specification format and rules for value expressions* in the manual *HADB SQL Reference*.

Example 2

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" NOT BETWEEN DATE'2016/01/01' AND DATE'2016/01/31'
```

In this example, the search range is not narrowed because NOT is specified in a BETWEEN predicate. In this case, you can narrow the search range by amending the SELECT statement as follows:

Amended example

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2015/12/01' AND DATE'2015/12/31'
      AND "RECORD-DAY" BETWEEN DATE'2016/02/01' AND DATE'2016/02/29'
```

As in this example, do not specify NOT in a BETWEEN predicate. Specify multiple BETWEEN predicates with an AND condition.

5.12.3 Notes about specifying JOIN (joined table)

When you specify a joined table, depending on how the joined table is specified, the search range might not be narrowed by the datetime information of the archive range column. The following explains how to specify joined tables in such a way that the search range is narrowed.

In the examples, ARCHIVE-T1 represents the archivable multi-chunk table, and RECORD-DAY represents the archive range column.

(1) Example 1 (LEFT OUTER JOIN)

If you specify the archivable multi-chunk table in the table reference on the right side of LEFT OUTER JOIN, the search range is not narrowed using the datetime information of the archive range column.

- **Example where search range is not narrowed (before)**

```
SELECT "T1"."C1" FROM "T1"
                 LEFT OUTER JOIN "ADBUSER01"."ARCHIVE-T1" AS "DT"
                 ON "T1"."C1" = "DT"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

If you specify the archivable multi-chunk table in the table reference on the right side of LEFT OUTER JOIN as underlined, the search range is not narrowed using the datetime information of the archive range column. In this case, searches will target all archived data and might take longer as a result.

To avoid this issue, modify the SQL statement as follows:

▪ Example where search range is narrowed (after)

```
SELECT "T1"."C1" FROM "T1"
                 LEFT OUTER JOIN (SELECT * FROM "ADBUSER01"."ARCHIVE-T1"
                                    WHERE "RECORD-DAY"
                                      BETWEEN DATE'2016/01/15' AND DATE'2016/02/15
'
                                    AND "C1"='P001') AS "DT"
                 ON "T1"."C1" = "DT"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

As underlined, the search condition that specifies the archive range column is replaced with an explicitly specified derived table. When the SQL statement is amended in this way, the search range is narrowed using the datetime information of the archive range column.

> **Note**
>
> No particular action is required if you specify the archivable multi-chunk table in the table reference on the left side of LEFT OUTER JOIN. That is, you do not need to modify the SQL statement.

(2) Example 2 (RIGHT OUTER JOIN)

If you specify the archivable multi-chunk table in the table reference on the left side of RIGHT OUTER JOIN, the search range is not narrowed using the datetime information of the archive range column.

▪ Example where search range is not narrowed (before)

```
SELECT "T1"."C1" FROM "ADBUSER01"."ARCHIVE-T1" AS "DT"
                 RIGHT OUTER JOIN "T1"
                 ON "DT"."C1" = "T1"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

If you specify the archivable multi-chunk table in the table reference on the left side of RIGHT OUTER JOIN as underlined, the search range is not narrowed using the datetime information of the archive range column. In this case, searches will target all archived data and might take longer as a result.

To avoid this issue, modify the SQL statement as follows:

▪ Example where search range is narrowed (after)

```
SELECT "T1"."C1" FROM (SELECT * FROM "ADBUSER01"."ARCHIVE-T1"
                         WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016
/02/15'
                         AND "C1"='P001') AS "DT"
                 RIGHT OUTER JOIN "T1"
                 ON "DT"."C1" = "T1"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

As underlined, the search condition that specifies the archive range column is replaced with an explicitly specified derived table. When the SQL statement is amended in this way, the search range is narrowed using the datetime information of the archive range column.

> **Note**
>
> No particular action is required if you specify the archivable multi-chunk table in the table reference on the right side of `RIGHT OUTER JOIN`. That is, you do not need to modify the SQL statement.

(3) Example 3 (FULL OUTER JOIN)

If you specify the archivable multi-chunk table in the table reference on the left or right side of `FULL OUTER JOIN`, the search range is not narrowed using the datetime information of the archive range column.

▪ **Example where search range is not narrowed (before)**

```
SELECT "T1"."C1" FROM "ADBUSER01"."ARCHIVE-T1" AS "DT"
                 FULL OUTER JOIN "T1"
                 ON "DT"."C1" = "T1"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

If, as underlined, you specify the archivable multi-chunk table in the table reference on the left or right side of `FULL OUTER JOIN`, the search range is not narrowed using the datetime information of the archive range column. In this case, searches will target all archived data and might take longer as a result.

To avoid this issue, modify the SQL statement as follows:

▪ **Example where search range is narrowed (after)**

```
SELECT "T1"."C1" FROM (SELECT * FROM "ADBUSER01"."ARCHIVE-T1"
                          WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016
/02/15'
                          AND "C1"='P001') AS "DT"
                 FULL OUTER JOIN "T1"
                 ON "DT"."C1" = "T1"."C1"
  WHERE "RECORD-DAY" BETWEEN DATE'2016/01/15' AND DATE'2016/02/15'
  AND "C1"='P001'
```

Explanation:

As underlined, the search condition that specifies the archive range column is replaced with an explicitly specified derived table. When the SQL statement is amended in this way, the search range is narrowed using the datetime information of the archive range column.

5.12.4 Equivalent exchange of SQL statements that search archivable multi-chunk tables

When all of the following conditions are met, the HADB server uses equivalent exchange to automatically transform SQL statements that search archivable multi-chunk tables.

- When searching archived data
- A condition that specifies the archive range column is specified as a search condition in a `WHERE` clause.

- The search condition specified in the WHERE clause complies with the rules explained in 5.12.2 Using the datetime information of the archive range column to narrow the search range.

Note

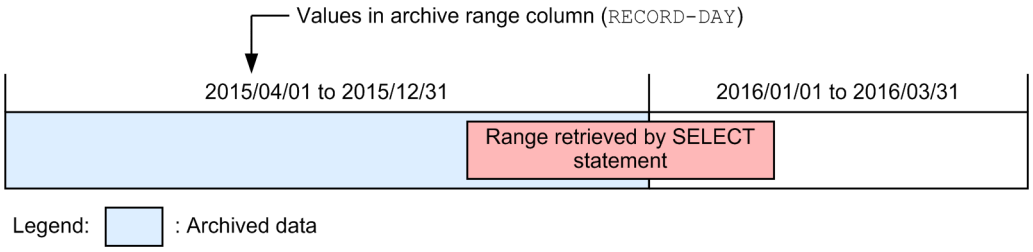
- The SQL statement after equivalent exchange is output to the access path information.
- The rules for SQL statements are applied to the SQL statement after equivalent exchange.

The following shows an example of equivalent exchange.

Example of specified SELECT statement

```
SELECT * FROM "ARCHIVE-T1"
    WHERE "RECORD-DAY" BETWEEN DATE'2015/11/01' AND DATE'2016/01/31'
    AND "C1"='P001'
```

Search range used by this SELECT statement



Example of SELECT statement after equivalent exchange

```
SELECT * FROM
  (SELECT * FROM "ARCHIVE-T1"                                    ...1
   UNION ALL                                                     ...2
   SELECT * FROM                                                 ...3
     TABLE (ADB_CSVREAD(MULTISET(                                ...4
       SELECT "ARCHIVE_FILE_NAME" FROM
                 "HADB"."LOCATION_TABLE_00020191" AS "LOC"
                ,"HADB"."STATUS_CHUNKS" AS "SCK"
         WHERE "RANGE_MAX" >= DATE'2015/11/01'                   ...5
           AND "RANGE_MIN" <= DATE'2016/01/31'                   ...5
           AND "SCK"."TABLE_SCHEMA" = 'ADBUSER01'
           AND "SCK"."TABLE_NAME" = 'ARCHIVE-T1'
           AND "SCK"."CHUNK_ID" = "LOC"."CHUNK_ID"
           AND "SCK"."CHUNK_STATUS" IS NULL
       )
       ,'omitted')
     ) AS "TBLFUNC_00020191" ("C1" VARCHAR(10), "C2" INT, "RECORD_DAY" DATE)    ...
6
  )
    WHERE "RECORD_DAY" BETWEEN DATE'2015/11/01' AND DATE'2016/01/31'
    AND "C1"='P001'
```

Explanation:

The SQL statement in this example searches archived data and data that is not archived. In this example, the SQL statement is rewritten into a query that searches archived data and a query that searches unarchived data, and the union of these two queries is determined by a UNION ALL operator.

1. A query that searches unarchived data
2. Determines the union (UNION ALL) of the queries labeled 1 and 3

3. A query that searches archived data
4. This part is converted to an ADB_CSVREAD function that reads the archive files that store the archived data.
5. The search condition specified in the WHERE clause is rewritten as a search condition that searches the location table.#
6. TBLFUNC_00020191 is the correlation name of the table function derived table. Correlation names are determined according to the following conventions:

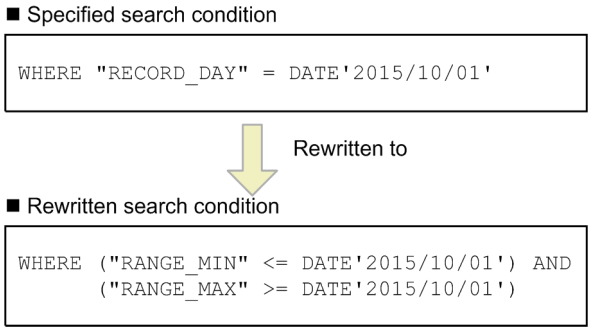
   TBLFUNC_nnnnnnnn

   nnnnnnnn: An 8 character string (0 to 9 and A to F) obtained by converting the table ID of the archivable multi-chunk table to hexadecimal

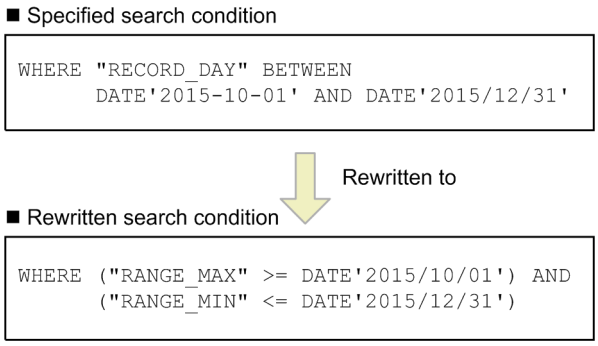
#

Examples of rewriting the search condition as a search condition that searches the location table are as follows:

Example 1:



Example 2:



- RANGE_MAX

  A column in the location table. RANGE_MAX is the maximum value in the archive range column for each archive file.

- RANGE_MIN

  A column in the location table. RANGE_MIN is the minimum value in the archive range column for each archive file.

Note

The internal derived tables generated by equivalent exchange of SQL statements that search archivable multi-chunk tables are not subject to expansion. For details about the expansion of internal derived tables, see *Internal derived tables* in the manual *HADB SQL Reference*.

5.13 Expanding internal derived tables

HADB automatically changes query expressions containing internal derived tables and efficiently evaluates the internal derived tables. Changing query expressions containing internal derived tables is called expanding internal derived tables. For details about the rules for expanding internal derived tables, see the topic *Internal derived tables* in the manual *HADB SQL Reference*.

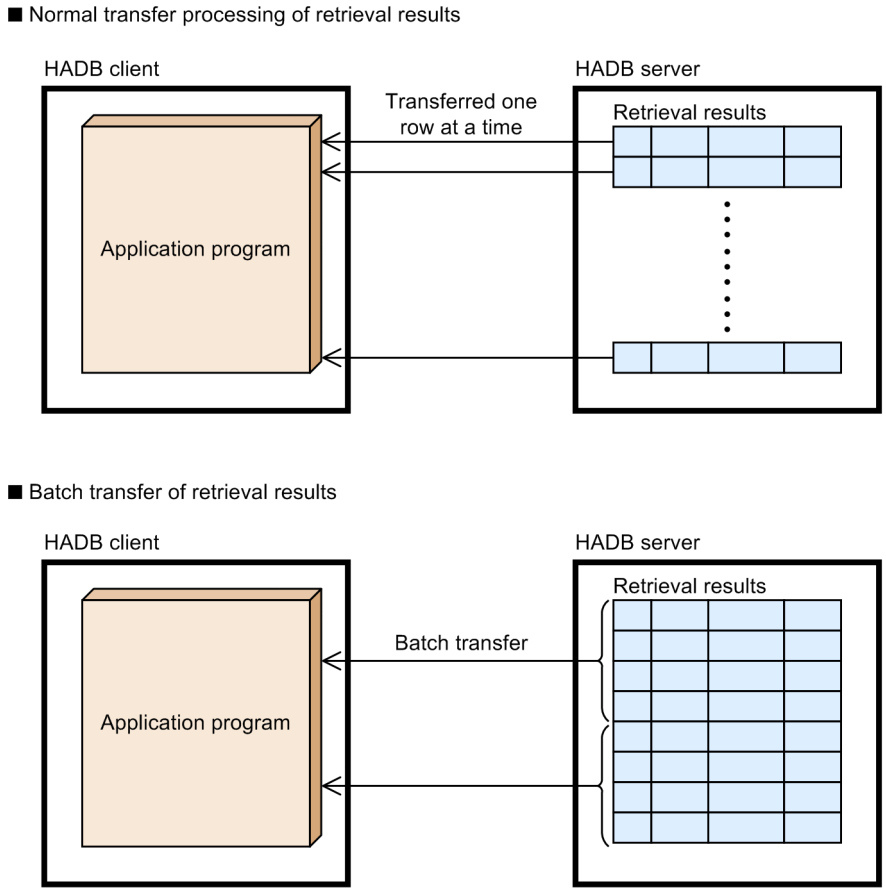
The priority order and selection rules for indexes that are used during retrieval are applied to the query expressions obtained after internal derived tables have been expanded. For details about the index to be used during retrieval, see 5.2 B-tree indexes and text indexes used during execution of SQL statements and 5.3 Range indexes used during execution of SQL statements.

5.14 Improving performance by batch transfer of retrieval results

You can transfer multiple rows of retrieval results in the batch mode from the HADB server to the HADB client. This feature is useful when you retrieve a large amount of data.

The following figure provides an overview of batch transfer of retrieval results.

Figure 5-21: Overview of batch transfer of retrieval results



Specify a setting to transfer retrieval results in the batch mode. The following are the possible settings:

- If you use a JDBC driver:
  Specify the number of rows to be transferred in the batch mode by using one of the following methods:
  - `adb_clt_fetch_size` in the system properties, user properties, or URL connection properties
  - `setFetchSize` method of the `Statement` object
  - `setFetchSize` method of the `ResultSet` object
- If you use ODBC drivers or CLI functions:
  Specify the number of rows to be transferred in the batch mode in the `adb_clt_fetch_size` operand in the client definition.

If the appropriate setting above is specified and rows are retrieved, the retrieval results are transferred in the batch mode.

Notes

- As the number of rows to be transferred in the batch mode increases, the amount of memory used by the HADB server and HADB client also increases. Therefore, if you will be performing batch transfer, re-evaluate the

memory requirements. For details about the memory requirements for HADB servers, see *Determining the memory requirement during normal operation* in the *HADB Setup and Operation Guide*. For details about the memory requirements for HADB clients, see C. Estimating the Memory Requirements for an HADB Client

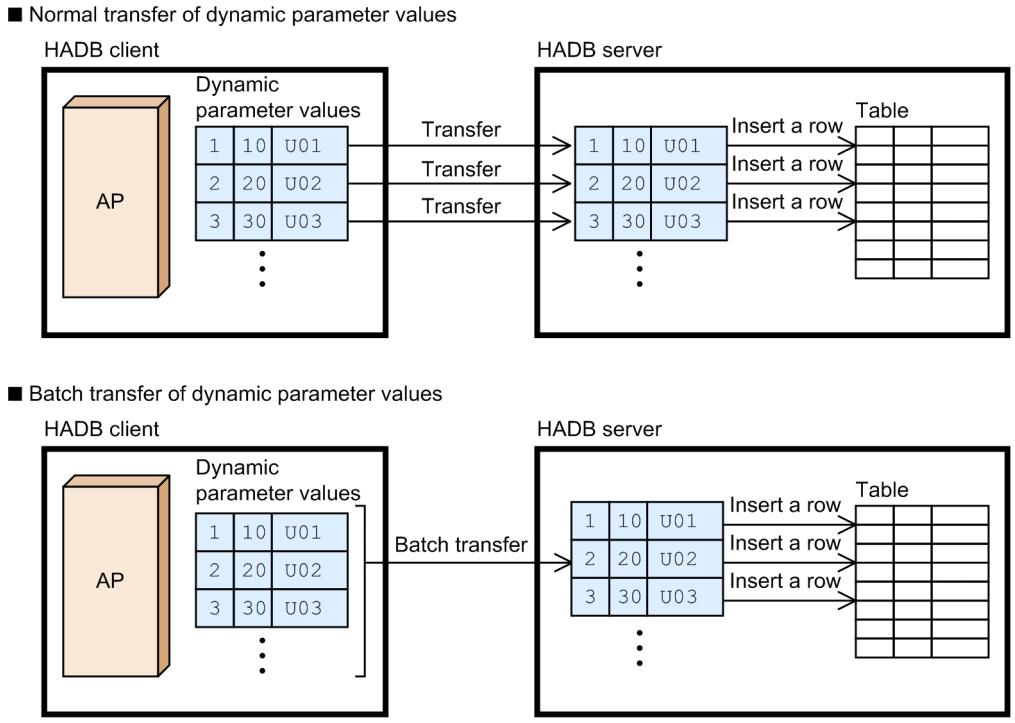
- If an error occurs during batch transfer, retrieval results buffered on the HADB server are discarded and only error information is returned to the HADB client.
- If an update SQL statement is run during retrieval using a cursor, the result of the update operation might be applied to the retrieval results, depending on the timing. However, the result of the update operation is never applied to the retrieval results during retrieval using batch transfer. This is because the HADB server is not accessed by an HADB client on which retrieval results remain.

5.15 Batch transfer of dynamic parameter values

Multiple dynamic parameter values can be transferred from an HADB client to the HADB server in the batch mode. This function is useful for accessing the HADB server from an HADB client and then using dynamic parameters to add, update, and delete data.

The following figure provides an overview of batch transfer of dynamic parameter values.

Figure 5-22: Overview of batch transfer of dynamic parameter values



Explanation:

When multiple rows are inserted (`INSERT` statement) by using dynamic parameter values as row insertion values, the dynamic parameter values are normally transferred to the HADB server one set at a time (where a set consists of the dynamic parameter values to be inserted into one row).

In the case of batch transfer of dynamic parameter values, two or more sets of dynamic parameter values are transferred together in the batch mode. Because this method reduces the communication overhead, it also reduces the application program's execution time.

The figure above shows an example of row insertion (`INSERT` statement), but batch transfer of dynamic parameter values is also applicable to the `UPDATE` and `DELETE` statements.

If a JDBC driver is used, dynamic parameter values are transferred in the batch mode when SQL statements are executed by using the `executeBatch` method or `executeLargeBatch` method of the `PreparedStatement` class.

If CLI functions are used, dynamic parameter values are transferred in the batch mode when the SQL statements are executed in the following order:

1. Preprocess the `INSERT`, `UPDATE`, or `DELETE` statement.
2. Use `a_rdb_SQLBindArrayParams()` to perform batch binding of the dynamic parameters.
3. Use the statement handle preprocessed in 1 to execute the SQL statement by `a_rdb_SQLExecute()`.

If an ODBC driver is used, batch transfer of dynamic parameter values is not available.

Notes

- If an error occurs during execution of an SQL statement and the processing is rolled back, the number of results rows is discarded and only the error information is returned to the HADB client.
- You must re-estimate the memory requirements for the HADB servers and clients, because batch transfer of dynamic parameter values requires memory resources. For details about the memory requirements for HADB servers, see *Determining the memory requirement during normal operation* in the *HADB Setup and Operation Guide*. For details about the memory requirements for HADB clients, see C. Estimating the Memory Requirements for an HADB Client
- If you perform batch transfer of dynamic parameter values for an SQL statement with the scalar function `RANDOMCURSOR` specified, the scalar function `RANDOMCURSOR` generates a pseudorandom number for each set of dynamic parameters.

6 Tuning Application Programs

This chapter explains how to use access paths.

6.1 How to use access paths (how to use SQL statement execution plans)

This section explains the information that is output as access paths (SQL statement execution plans) and how to use that information.

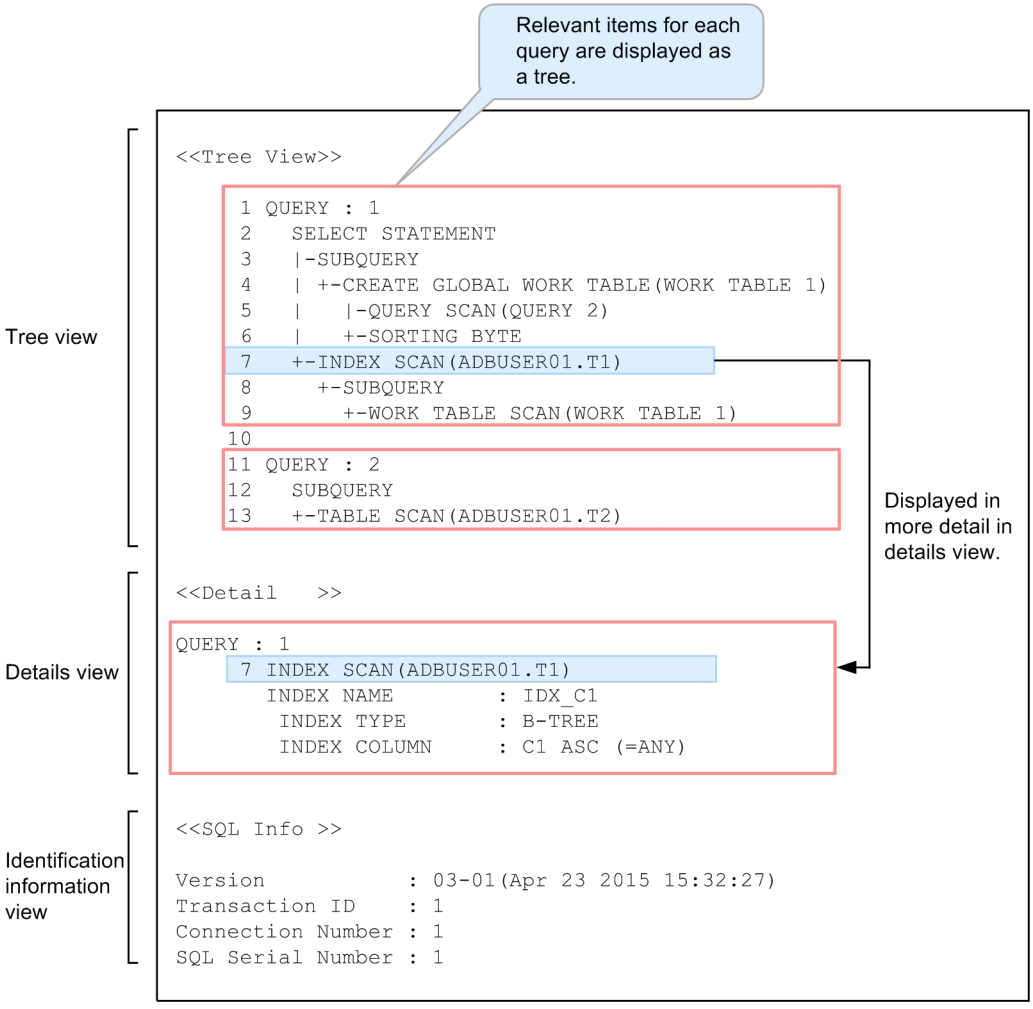
6.1.1 About access paths

An access path is the execution plan that is used when HADB executes an SQL statement. By checking the access paths, you can determine how the SQL statements to be executed will be processed by HADB.

(1) Example of displayed access path information

The following figure shows an example of displayed access path information.

Figure 6-1: Example of displayed access path information



Explanation:

Access path information is displayed in three views: tree view, details view, and identification information view.

- Tree view

  The information displayed in tree view includes table search methods, table joining methods, and information about work tables. Related items are displayed in tree format.

Access path information is displayed for each query. Each query is assigned a query tree number.

`<<Tree View>>` is displayed as the header for the tree view section.

- Details view

  The details view displays detailed information about the items in tree view. The tree row numbers in the details view correspond to the tree row numbers in tree view. In this example, details view displays detailed information about the item assigned tree row number 7 in tree view.

  `<<Detail >>` is displayed as the header for the details view section.

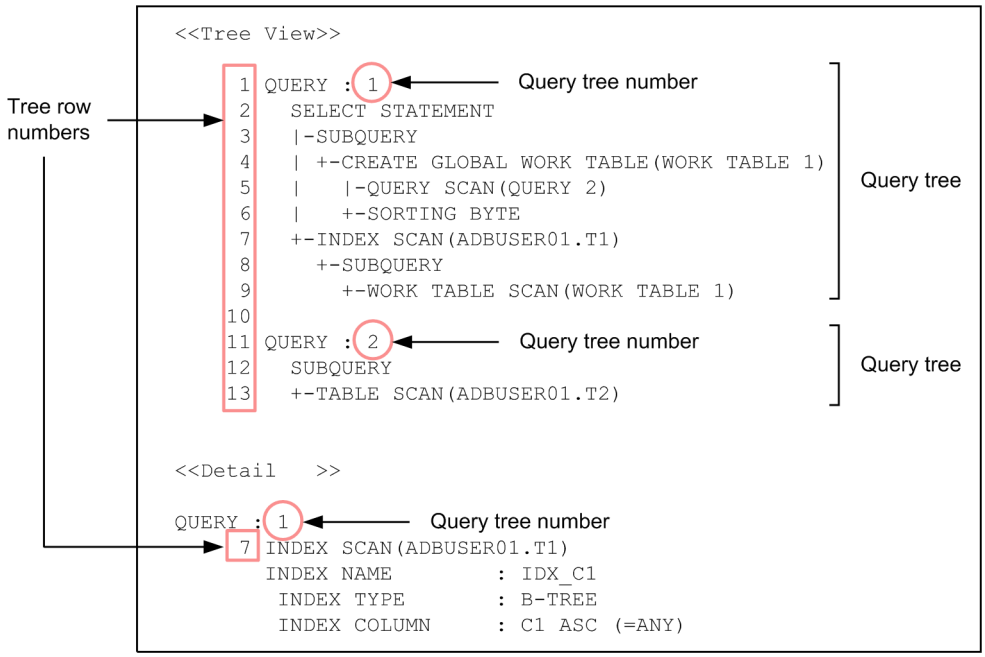
- Identification information view

  The identification information view displays SQL statement identification information. SQL statement identification information is information used to identify the SQL statement for which HADB acquired access path statistical information. Based on the information in the identification information view, you can identify the correlation between the output access path information and the output access path statistical information.

  `<<SQL Info >>` is displayed as the header for the identification information view section.

  For details about access path statistical information, see *Examples of output of and output items for access path statistical information* in the *HADB Setup and Operation Guide*.

■ Query trees, query tree numbers, and tree row numbers



Explanation:

- Query tree

  Access path information is output in a form of a query tree for each query (query specification and table value constructor).

- Query tree number

  Each query tree is assigned a number (called a query tree number) for identification.

  A query tree corresponding to a subquery is assigned a query tree number of 2 or larger. Note, however, that a query tree number of 1 or larger is assigned to a query tree that corresponds to a subquery if the following conditions are met:

  - `VALUES` (in which a subquery is specified) is specified in the `INSERT` statement.
  - A subquery is specified in the `PURGE CHUNK` statement.

Note

- Whether query tree numbers are output in exact ascending order (1, 2, ...) depends on the SQL statement to be run.
- A query tree number of 0 is output for query trees that are not contained in the queries in the SQL statement.

- Tree row number
  A number assigned to each row in the tree.

(2) SQL statements for which access paths are output

Access path information is output for the following SQL statements:

- SELECT
- UPDATE
- INSERT
  Access path information is output in the following circumstances:
  - A query expression body is specified in the INSERT statement.
  - VALUES is specified in the INSERT statement, and a subquery is specified.
- DELETE
- PURGE CHUNK
  Access path information is output when a subquery is specified in the PURGE CHUNK statement.

Access path information is output when you run these SQL statements.

6.1.2 How to check access paths

There are two ways to check access paths:

- Check access path information by executing the #SET OPT REPORT subcommand of the adbsql command
- Check access path information output in SQL trace information

(1) How to check access path information by running #SET OPT REPORT

You can display access path information by running the #SET OPT REPORT subcommand of the adbsql command. To display access path information:

Procedure

1. Run the adbsql command.
2. Run the #SET OPT REPORT subcommand of the adbsql command.

```
#SET OPT REPORT ON TYPE=PATH EXEC=PREPARE;
```

> **Note**
>
> In this example, EXEC=PREPARE is specified because the intention is to display the access path information without actually running the SQL statement. If you want to run the SQL statement and display the access path information, do not specify EXEC=PREPARE.

3. Run the SQL statement.

```
SELECT "C1" FROM "T1" WHERE "C1">10;
```

When you run this SQL statement, access path information is displayed as follows:

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-KEY SCAN(ADBUSER01.T1)


<<Detail   >>

QUERY : 1
     3 KEY SCAN(ADBUSER01.T1)
       INDEX NAME        : IDX_C1
        INDEX TYPE       : B-TREE
        INDEX COLUMN     : C1 ASC (>)


<<SQL Info >>

Version           : 03-01(Aug  5 2015 09:32:34)
Transaction ID    : 6197
Connection Number : 3
SQL Serial Number : 1
```

In this example, the table and B-tree index are defined as follows:

```
CREATE TABLE "T1"("C1" INT,"C2" DEC,"C3" CHAR(10)) IN "DBAREA01"
CREATE INDEX "IDX_C1" ON "T1"("C1") IN "DBAREA02" EMPTY
CREATE INDEX "IDX_C3" ON "T1"("C3") IN "DBAREA02" EMPTY
```

(2) How to check access path information output in SQL trace information

Use a text editor to view SQL trace information output in an SQL trace file. SQL trace files have the following file names:

- $ADBDIR/spool/adbsqltrc01.log to adbsqltrc08.log

The following is an example of access path information output as SQL trace information:

Example:

```
[SQL]
SELECT * FROM "T1" WHERE "C1"=? AND "C2"=? AND "C3"=?

[access path]
<<Tree View>>
```

```
1 QUERY : 1
2   SELECT STATEMENT
3   +-TABLE SCAN(T1)
```

For details about SQL trace information, see *Running SQL tracing* in the *HADB Setup and Operation Guide*.

6.1.3 Examples of access paths

(1) Example 1

Table B-tree index definitions

```
CREATE TABLE "T1"("C1" INT,"C2" DEC,"C3" CHAR(10)) IN "DBAREA01"
CREATE INDEX "IDX_C1" ON "T1"("C1") IN "DBAREA02" EMPTY
CREATE INDEX "IDX_C3" ON "T1"("C3") IN "DBAREA02" EMPTY
CREATE TABLE "T2"("C1" INT,"C2" DEC,"C3" CHAR(10)) IN "DBAREA01"
```

Executed SQL statement

```
SELECT * FROM "T1","T2"
    WHERE "T1"."C1"="T2"."C2"
```

Example of access paths

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT                        ...1
     3   +-NESTED LOOP JOIN                      ...2
     4     |-TABLE SCAN(ADBUSER01.T2)            ...3
     5     +-INDEX SCAN(ADBUSER01.T1) -ORDER     ...4


<<Detail   >>

QUERY : 1
     5 INDEX SCAN(ADBUSER01.T1)                  ...5
       INDEX NAME        : IDX_C1               ...6
        INDEX TYPE       : B-TREE               ...7
        INDEX COLUMN     : C1 ASC (=)           ...8


<<SQL Info >>

Version           : 03-01(Aug  5 2015 09:32:34)
Transaction ID    : 6197
Connection Number : 3
SQL Serial Number : 2
```

Explanation:

1. The `SELECT` statement is executed.
2. In the table join processing, a nested loop join is executed.
3. In table `T2` retrieval processing, a table scan is performed.
4. In table `T1` retrieval processing, an index scan is performed. `-ORDER` indicates that sequential execution, not out-of-order execution, is applied.
5. Detailed information about the index scan on table `T1` is displayed.

6. B-tree index `IDX_C1` is used during the index scan.
7. The index type is displayed. `B-TREE` means that this is a B-tree index.
8. Information about B-tree index `IDX_C1` is displayed:
   - `C1`: Indexed column of B-tree index `IDX_C1`
   - `ASC`: Key value sort order (ascending)
   - `(=)` : Range search condition specification

(2) Example 2

Table definition

```
CREATE TABLE "T1"("C1" INT,"C2" DEC,"C3" CHAR(10)) IN "DBAREA01"
CREATE TABLE "T2"("C1" INT,"C2" DEC,"C3" CHAR(10)) IN "DBAREA01"
```

Executed SQL statement

```
SELECT * FROM "T1"
     WHERE "C1"=ANY(SELECT "C2" FROM "T2" WHERE "C1"="T1"."C2")
```

Example of access paths

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT                                ...1
     3   +-TABLE SCAN(ADBUSER01.T1)                      ...2
     4     +-SUBQUERY LOOP                               ...3
     5       |-CREATE LOCAL WORK TABLE(WORK TABLE 1)     ...4
     6       | +-QUERY SCAN(QUERY 2)                     ...5
     7       +-WORK TABLE SCAN(WORK TABLE 1)             ...6
     8
     9 QUERY : 2                                         ...7
    10   SUBQUERY LOOP                                   ...8
    11   +-TABLE SCAN(ADBUSER01.T2)                      ...9


<<SQL Info >>

Version           : 03-01(Aug  5 2015 09:32:34)
Transaction ID    : 6197
Connection Number : 3
SQL Serial Number : 3
```

Explanation:

1. The `SELECT` statement is executed.
2. In table `T1` retrieval processing, a table scan is executed.
3. Nested loops work table execution or nested loops row value execution is used to process the subquery specified in the quantified predicate (`ANY`).
4. In the subquery processing, a local work table is created.
5. A query scan is performed. The query scan `QUERY : 2` displayed in tree row number 9 is performed.
6. In the subquery processing, the work table is scanned.
7. Detailed information about the subquery is displayed.
8. In the subquery processing, nested loops work table execution or nested loops row value execution is used.

9. In the table T2 retrieval processing, a table scan is performed.

6.1.4 Information displayed in the tree view

The tree view displays information for each query, such as the table retrieval method, table joining method, and subquery processing method, in tree format.

(1) SQL statements executed

One of the following is displayed:

- SELECT STATEMENT

  A SELECT statement is to be executed.
- UPDATE STATEMENT

  An UPDATE statement is to be executed.
- INSERT STATEMENT

  An INSERT statement is to be executed.
- DELETE STATEMENT

  A DELETE statement is to be executed.
- PURGE CHUNK STATEMENT

  PURGE CHUNK statement is to be executed.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

A SELECT statement is executed.

(2) Subquery processing methods

One of the following is displayed:

- SUBQUERY

  A subquery processing method other than nested loop execution or hash execution is applied.
- SUBQUERY LOOP

  Nested loops work table execution or nested loops row value execution is applied as the subquery processing.
- SUBQUERY HASH

  Hash execution is applied as the subquery processing method.

  If SUBQUERY HASH is followed by FILTER, a hash filter is applied during hash execution.

For details about how to process subqueries, see 5.6 How to process subqueries.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-SUBQUERY HASH -FILTER
     4     |-QUERY SCAN(QUERY 2)
     5     +-TABLE SCAN(ADBUSER01.T1)
     6
     7 QUERY : 2
     8   SUBQUERY HASH
     9   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

Hash execution is applied as the subquery processing method.

Because `SUBQUERY HASH` is followed by `FILTER` in tree row number 3, a hash filter is applied during hash execution.

(3) Specification of derived tables

The following information is displayed:

- `DERIVED TABLE`(*correlation-name*)

  One of the following is specified:

  - A derived table
  - A viewed table (if a correlation name is specified)
  - A query name (if a correlation name is specified)
- `DERIVED TABLE`(*query-name*)

  A query name is specified (without a correlation name).
- `DERIVED TABLE`(*table-identifier*)

  A viewed table is specified (without a correlation name)

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-DERIVED TABLE(D1)
     4     +-QUERY SCAN(QUERY 2)
     5
     6 QUERY : 2
     7   DERIVED TABLE(D1)
     8   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     9   | |-TABLE SCAN(ADBUSER01.T2)
    10   | |-SORTING BYTE
    11   | +-LIMIT 10
    12   |-WORK TABLE SCAN(WORK TABLE 1)
    13   +-LIMIT 10
```

Explanation:

A derived table is specified in the `SELECT` statement. The correlation name is displayed in parentheses.

(4) Specification of set operations

The following information is displayed:

- SET OPERATION

  A set operation is specified.

> **Note**
>
> The preceding information is also displayed in the following circumstances:
>
> - FULL OUTER JOIN is specified for joining tables
> - An archivable multi-chunk table is specified
>
> When retrieving an archivable multi-chunk table, equivalent exchange of the SQL statement might result in it being automatically rewritten as an SQL statement that specifies a set operation. For details, see 5.12.4 Equivalent exchange of SQL statements that search archivable multi-chunk tables.
> - Equivalent exchange related to OR conditions (equivalent exchange to a derived table for which the UNION ALL set operation is specified) is applied
>
> If a comma join or joined table is specified in the FROM clause and an OR condition is specified in the WHERE clause, the SQL statement might be automatically rewritten as a result of equivalent exchange to derived tables for which the UNION ALL set operation is specified. For details, see 5.11.3 Equivalent exchange for OR conditions (equivalent exchange to derived tables for which the UNION ALL set operation is specified).

Output example

```
<<Tree View>>

     1 QUERY : 0
     2   SELECT STATEMENT
     3   +-SET OPERATION
     4     |-QUERY SCAN(QUERY 1)
     5     +-QUERY SCAN(QUERY 2)
     6
     7 QUERY : 1
     8   QUERY
     9   +-TABLE SCAN(ADBUSER01.T1)
    10
    11 QUERY : 2
    12   QUERY
    13   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

A set operation is specified in the SELECT statement.

Note that SET OPERATION might be followed by the following item:

- RECURSIVE

  This item indicates that a recursive query will be run.

Output example

```
<<Tree View>>
```

```
 1 QUERY : 3
 2   SELECT STATEMENT
 3   +-DERIVED TABLE(REC)
 4     +-SET OPERATION -RECURSIVE
 5       |-QUERY SCAN(QUERY 1)
 6       +-QUERY SCAN(QUERY 2)
 7
 8 QUERY : 1
 9   QUERY
10   |-CREATE GLOBAL WORK TABLE(WORK TABLE 1)
11   | +-TABLE SCAN(ADBUSER01.T1)
12   +-WORK TABLE SCAN(WORK TABLE 1)
13
14 QUERY : 2
15   QUERY
16   |-CREATE GLOBAL WORK TABLE(WORK TABLE 2)
17   | +-WORK TABLE SCAN(WORK TABLE 2)
18   +-WORK TABLE SCAN(WORK TABLE 2)
```

Explanation:

An item indicating that a recursive query will be run is included.

After `SET OPERATION`, the query information displayed first pertains to anchor members. The query information displayed next pertains to recursive members. In the preceding output example, `QUERY 1` is the query information about anchor members, and `QUERY 2` is the query information about recursive members.

(5) Set operation method specification

The following information is displayed:

- `SPECIFIC`

  Set operation method specification is enabled.

  For details about set operation method specifications, see *Specification format and rules for query expressions* in the manual *HADB SQL Reference*.

Output example

```
<<Tree View>>

 1 QUERY : 0
 2   SELECT STATEMENT
 3   +-SET OPERATION -SPECIFIC
 4     |-QUERY SCAN(QUERY 1)
 5     +-QUERY SCAN(QUERY 2)
 6
 7 QUERY : 1
 8   QUERY
 9   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
10   | |-TABLE SCAN(ADBUSER01.T1)
11   | +-SORTING BYTE -DISTINCT
12   +-WORK TABLE SCAN(WORK TABLE 1)
13
14 QUERY : 2
15   QUERY
16   |-CREATE LOCAL WORK TABLE(WORK TABLE 2)
17   | |-TABLE SCAN(ADBUSER01.T2)
18   | +-SORTING BYTE -DISTINCT
19   +-WORK TABLE SCAN(WORK TABLE 2)
```

Explanation:

The set operation method specification specified in the `SELECT` statement is enabled.

(6) Query types

The following information is displayed:

- QUERY

  A query expression body other than a subquery or a derived table is specified.

> **Note**
>
> If an archivable multi-chunk table is specified as the table to be updated by an UPDATE statement or the table to be deleted by a DELETE statement, a query that retrieves the location table or system table (STATUS_CHUNKS) might be displayed.

Output example

```
<<Tree View>>

     1 QUERY : 0
     2   INSERT STATEMENT
     3   +-QUERY SCAN(QUERY 1)
     4
     5 QUERY : 1
     6   QUERY
     7   +-TABLE SCAN(ADBUSER01.T2) -ORDER
```

Explanation:

A query expression body is specified in the INSERT statement.

(7) Work table creation information

One of the following is displayed:

- CREATE GLOBAL WORK TABLE (WORK TABLE *work-table-number*)

  A global work table is created.

- CREATE LOCAL WORK TABLE (WORK TABLE *work-table-number*)

  A local work table is created.

A unique work table number is assigned to each work table.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-SUBQUERY
     4   | +-CREATE GLOBAL WORK TABLE(WORK TABLE 1)
     5   |   |-QUERY SCAN(QUERY 2)
     6   |   +-SORTING BYTE
     7   +-INDEX SCAN(ADBUSER01.T1)
     8     +-SUBQUERY
     9       +-WORK TABLE SCAN(WORK TABLE 1)
    10
    11 QUERY : 2
    12   SUBQUERY
    13   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

A global work table is created in the subquery processing.

(8) Subquery processing method specification

The following information is displayed:

- SPECIFIC

  Subquery processing method specification is enabled.

  For details about subquery processing method specifications, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-SUBQUERY -SPECIFIC
     4   | +-CREATE GLOBAL WORK TABLE(WORK TABLE 1)
     5   |   +-QUERY SCAN(QUERY 2)
     6   +-TABLE SCAN(ADBUSER01.T1(A))
     7     +-SUBQUERY
     8       +-WORK TABLE SCAN(WORK TABLE 1)
     9
    10 QUERY : 2
    11   SUBQUERY
    12   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

The subquery processing method specification specified in the SELECT statement is enabled.

(9) Subquery processing delegation specification

The following information is displayed:

- DELEGATION

  Subquery processing delegation specification is enabled.

  For details about subquery processing delegation specifications, see *Specification format and rules for subqueries* in the manual *HADB SQL Reference*.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-TABLE SCAN(ADBUSER01.T1(A))
     4     +-SUBQUERY LOOP -DELEGATION -USING CACHE
     5       +-QUERY SCAN(QUERY 2)
     6
     7 QUERY : 2
     8   SUBQUERY LOOP
     9   |-SUBQUERY LOOP -SPECIFIC -DELEGATION
    10   | +-QUERY SCAN(QUERY 3)
    11   +-TABLE SCAN(ADBUSER01.T1(B))
    12
    13 QUERY : 3
```

```
    14   SUBQUERY LOOP
    15   +-TABLE SCAN(ADBUSER01.T1(C))
```

Explanation:

The subquery processing delegation method specification specified in the SELECT statement is enabled.

(10) Subquery cache usage information

The following information is displayed:

- USING CACHE

  Cache is used to store the results of a subquery. Cache might be used when nested loops row value execution is used to process the subquery.

For details about nested loops row value execution, see (2) Nested loops row value execution in 5.6.3 Methods for processing subqueries that contain an external reference column.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-TABLE SCAN(ADBUSER01.T1)
     4     +-SUBQUERY LOOP -USING CACHE
     5       +-QUERY SCAN(QUERY 2)
     6
     7 QUERY : 2
     8   SUBQUERY LOOP
     9   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

Nested loops row value execution is used to process the subquery, and the results of the subquery are stored using cache.

(11) Specification of table function derived tables

The following information is displayed:

- TABLE FUNCTION DERIVED TABLE(*correlation-name*)

  A table function derived table is specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-TABLE FUNCTION DERIVED TABLE(T4)
     4     +-TABLE SCAN(ADBUSER01.T4)
```

Explanation:

A table function derived table is specified in the SELECT statement. The information in parentheses is the correlation name.

(12) Processing method for duplicate removal

The following information is displayed:

- GLOBAL HASH UNIQUE

  Duplication in the retrieval results will be eliminated by using one of the following methods:

  - Hash execution for the method for processing the set operation

    For details about hash execution for the method for processing the set operation, see 5.8.1 Hash execution.
  - Hash execution for the method for processing SELECT DISTINCT

    For details about hash execution for the method for processing SELECT DISTINCT, see 5.9.1 Hash execution.
  - Global hash grouping for the grouping method

    For details about global hash grouping for the grouping method, see (2) Global hash grouping in 5.7.1 Hash grouping.

  GLOBAL HASH UNIQUE is also displayed if work tables might be created because some rows cannot be processed in the hash table area.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-KEY SCAN(ADBUSER01.T1)
     4   |-GLOBAL HASH UNIQUE
     5   +-GROUPING
```

Explanation:

Global hash grouping is used for the grouping method to eliminate duplicate retrieval results.

(13) Grouping methods

One of the following is displayed:

- GROUPING

  Grouping that does not use work tables is performed.
- SORT GROUPING

  Sort grouping is performed.
- GLOBAL HASH GROUPING

  Global hash grouping is performed. If there are rows that cannot be processed in the hash table area, work tables might be created.
- LOCAL HASH GROUPING

  Local hash grouping is performed. If there are rows that cannot be processed in the hash grouping area, work tables might be created.

For details about grouping methods, see 5.7 Grouping methods.

Output example

```
<<Tree View>>

     1 QUERY : 1
```

```
     2   SELECT STATEMENT
     3   |-TABLE SCAN(ADBUSER01.T1)
     4   +-GLOBAL HASH GROUPING
```

Explanation:

Global hash grouping is performed during the grouping method.

Following the grouping method information, the following item might be displayed:

- SPECIFIC

  The grouping method specification specified in the GROUP BY clause is enabled.

  Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4   | |-TABLE SCAN(ADBUSER01.T1)
     5   | +-SORTING BYTE
     6   |-WORK TABLE SCAN(WORK TABLE 1)
     7   +-LOCAL HASH GROUPING -SPECIFIC
```

  Explanation:

  The grouping method specification specified in the GROUP BY clause is enabled and LOCAL HASH GROUPING is applied as the grouping method. For details about the grouping method specification, see the topic *Specification format and rules for GROUP BY clauses* in the manual *HADB SQL Reference*.

- INDEX

  The grouping method that uses the characteristics of B-tree indexes is performed.

  Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-KEY SCAN(ADBUSER01.T1)
     4   +-GROUPING -INDEX
```

  Explanation:

  The grouping method that uses the characteristics of B-tree indexes is performed for table T1.

- COLUMN

  The grouping method that uses the characteristics of column store tables is performed.

  Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-TABLE SCAN(ADBUSER01.T1) -COLUMN STORE
     4   +-GROUPING -COLUMN
```

  Explanation:

  The grouping method that uses the characteristics of column store tables is performed for table T1.

- GROUPING SET

  Grouping processing is performed multiple times.

Output example

```
<<Tree View>>
     1 QUERY : 1
     2 SELECT STATEMENT
     3 |-DERIVED TABLE(##DRVTBL_0000000001)
     4 | |-TABLE SCAN(ADBUSER01.T1)
     5 | +-GLOBAL HASH GROUPING -GROUPING SET
     6 +-GLOBAL HASH GROUPING
```

Explanation:

Grouping processing is performed multiple times by global hash grouping.

> **Note**
>
> `GROUPING SET` (grouping set information) might be output if multiple `DISTINCT` set functions are specified with different arguments.

(14) HAVING clause specification

The following information is displayed:

- `HAVING`

  The `HAVING` clause is specified. This information might also be displayed if a derived table is expanded even though the `HAVING` clause is not specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-TABLE SCAN(ADBUSER01.T1)
     4   |-GLOBAL HASH GROUPING
     5   +-HAVING
```

Explanation:

The `HAVING` clause is specified in the `SELECT` statement.

(15) Sort processing

The following information is displayed:

- `SORTING {BYTE|ISO}`
  - `BYTE`: Sorts by bytecode.
  - `ISO`: Sorts by sort code (ISO/IEC 14651:2011 compliance).

  Sort processing is performed according to the `ORDER BY` clause.

  Note that this information might not be displayed even though the `ORDER BY` clause is specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
```

```
     3   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4   | |-TABLE SCAN(ADBUSER01.T1)
     5   | +-SORTING BYTE
     6   +-WORK TABLE SCAN(WORK TABLE 1)
```

Explanation:

Sort processing is performed according to the specified `ORDER BY` clause.

(16) Information about duplicate removal

The following information is displayed:

- `DISTINCT`

  This item indicates that duplicate removal will be performed.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2 SELECT STATEMENT
     3 |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4 | |-KEY SCAN(ADBUSER01.T1)
     5 | +-SORTING BYTE -DISTINCT
     6 +-WORK TABLE SCAN(WORK TABLE 1)
```

Explanation:

The item indicating that duplicate removal will be performed during sort processing is output.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2 SELECT STATEMENT
     3 |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4 | |-KEY SCAN(ADBUSER01.T1) -DISTINCT
     5 | +-SORTING BYTE -DISTINCT
     6 +-WORK TABLE SCAN(WORK TABLE 1)
```

Explanation:

Items indicating that duplicate removal will be performed during key scan processing and sort processing are included.

(17) SELECT deduplication method specification

The following information is displayed:

- `SPECIFIC`

  `SELECT` deduplication method specification is enabled.

  For details about the `SELECT` deduplication method specification, see *Specification format and rules for query specifications* in the manual *HADB SQL Reference*.

  Note that if `DISTINCT` described in (16) Information about duplicate removal is not output, `SPECIFIC` is not output even if the `SELECT` deduplication method specification is specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4   | |-TABLE SCAN(ADBUSER01.T1)
     5   | +-SORTING BYTE -SPECIFIC -DISTINCT
     6   +-WORK TABLE SCAN(WORK TABLE 1)
```

Explanation:

The SELECT deduplication method specification specified in the SELECT statement is enabled.

(18) LIMIT clause specification

The following information is displayed:

- LIMIT [{*offset*|? PARAMETER},]{*row_count*|? PARAMETER}
  - *offset*

    A LIMIT clause specifying the offset for the first row to be returned is specified.

    If the literal 0 is specified for the offset row count, no offset for the first row to be returned is displayed.
  - *row_count*

    A LIMIT clause specifying the maximum number of rows to be returned is specified.
  - ? PARAMETER

    A LIMIT clause containing dynamic parameters for both or either of the offset row count and limit row count is specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4   | |-TABLE SCAN(ADBUSER01.T1)
     5   | |-SORTING BYTE
     6   | +-LIMIT ? PARAMETER,5
     7   |-WORK TABLE SCAN(WORK TABLE 1)
     8   +-LIMIT ? PARAMETER,5
```

Explanation:

A LIMIT clause containing a dynamic parameter for the offset row count and 5 for the limit row count is specified.

(19) Specification of window functions

The following information is displayed:

- WINDOW

  A window function is specified.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-CREATE LOCAL WORK TABLE(WORK TABLE 1)
     4   | |-TABLE SCAN(ADBUSER01.T1)
     5   | +-SORTING BYTE
     6   |-WORK TABLE SCAN(WORK TABLE 1)
     7   +-WINDOW
```

Explanation:

A window function is executed according to its specification.

(20) Table retrieval method

One of the following is displayed:

- `TABLE SCAN`

  A table scan is performed in the table retrieval processing.

- `INDEX SCAN(`*schema-name*`.`*table-identifier*`(`*query-name-or-correlation-name*`))`

  An index scan is performed in the table retrieval processing. If there is a query name or a correlation name, it is displayed.

- `KEY SCAN`

  A key scan is performed in the table retrieval processing.

For details about table scans, index scans, and key scans, see 5.1 How to retrieve tables.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-INDEX SCAN(ADBUSER01.T1)
```

Explanation:

An index scan is performed in the table `T1` retrieval processing.

(21) Table-data storage format

The following information is displayed:

- `COLUMN STORE`

  The table-data storage format is column store format.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-TABLE SCAN(ADBUSER01.T1) -COLUMN STORE
```

Explanation:

The table-data storage format of table T1 is column store format.

(22) Sequential execution

The following information is displayed:

- ORDER

  Sequential execution, not out-of-order execution, is applied.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-NESTED LOOP JOIN
     4     |-TABLE SCAN(ADBUSER01.T2)
     5     +-INDEX SCAN(ADBUSER01.T1) -ORDER
```

Explanation:

In the table T1 retrieval processing, an index scan is performed using the sequential execution method.

(23) Index specification

One of the following is displayed:

- SPECIFIC

  The index specification is enabled.

- SPECIFIC DISABLED

  The index specification is disabled.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-INDEX SCAN(ADBUSER01.T1) -SPECIFIC
```

Explanation:

An index scan is performed in the table T1 retrieval processing with an index specification specified in the SELECT statement enabled.

(24) Collecting cost information

The following information is displayed:

- USING COST

  Cost information is collected for a table or index.

Output example

```
<<Tree View>>

     1 QUERY : 1
```

```
     2   SELECT STATEMENT
     3   +-TABLE SCAN(ADBUSER01.T3) -USING COST
```

Explanation:

Cost information was collected for the table T3.

(25) Work table scan

The following information is displayed:

- WORK TABLE SCAN(WORK TABLE *work-table-number*)

  A work table is scanned. A unique work table number is assigned to each work table.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   |-SUBQUERY
     4   | +-CREATE GLOBAL WORK TABLE(WORK TABLE 1)
     5   |   |-QUERY SCAN(QUERY 2)
     6   |   +-SORTING BYTE
     7   +-INDEX SCAN(ADBUSER01.T1)
     8     +-SUBQUERY
     9       +-WORK TABLE SCAN(WORK TABLE 1)
    10
    11 QUERY : 2
    12   SUBQUERY
    13   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

A work table is scanned in the subquery processing.

Note that you might encounter circumstances in which the row IDs of a table specified in a FROM clause are stored in a work table created when executing an SQL statement that specifies an ORDER BY clause. In this case, data might be retrieved from data pages using these row IDs immediately after the work table is retrieved. For details about the purposes and columns of work tables, see 5.10 Considerations when executing an SQL statement that creates work tables.

(26) Query scan

The following information is displayed:

- QUERY SCAN(QUERY *query-tree-number*)

  A query scan is performed.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-SUBQUERY HASH
     4     |-QUERY SCAN(QUERY 2)
     5     +-TABLE SCAN(ADBUSER01.T1)
     6
     7 QUERY : 2
```

```
     8   SUBQUERY HASH
     9   +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

A query scan is performed.

In this example, the query scan indicated as `QUERY : 2` displayed in tree row number 7 is performed.

Example of the executed SELECT statement

```
SELECT * FROM "T1" WHERE "C1"=(SELECT "C2" FROM "T2" WHERE "C1"="T1"."C1")
```

(27) Table joining methods

One of the following is displayed:

- `NESTED LOOP JOIN`

  A nested loop join is performed in the table join processing.

- `HASH JOIN`

  A hash join is performed in the table join processing.

  If `HASH JOIN` is followed by `FILTER`, a hash filter is applied during hash join.

  If there is an = condition that compares columns in two tables, a hash join might be performed. If there are rows that cannot be processed in the hash table area, work tables might be created.

For details about table joining methods, see 5.5 Table joining methods.

Output example (NESTED LOOP JOIN)

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-NESTED LOOP JOIN
     4     |-TABLE SCAN(ADBUSER01.T2)
     5     +-INDEX SCAN(ADBUSER01.T1) -ORDER
```

Explanation:

- A nested loop join is performed to join tables `T1` and `T2`.
- If the table joining method is nested loop join, the information is displayed under `NESTED LOOP JOIN` in order, beginning with the outer table. In this example, the information in tree row number 4 is for the outer table and the information in tree row number 5 is for the inner table.

Output example (HASH JOIN)

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-HASH JOIN -FILTER
     4     |-TABLE SCAN(ADBUSER01.T1)
     5     +-TABLE SCAN(ADBUSER01.T2)
```

Explanation:

- A hash join is performed to join tables `T1` and `T2`.

- If the table joining method is hash join, information for the outer table and the inner table for hash join is displayed in this order under HASH JOIN. In this example, the information in tree row number 4 is for the outer table and the information in tree row number 5 is for the inner table.
- Because HASH JOIN is followed by FILTER in tree row number 3, a hash filter is applied during hash join.

(28) Join method specification

Either of the following is displayed:

- SPECIFIC

  Join method specification is enabled.
- SPECIFIC DISABLED

  Join method specification is disabled.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-HASH JOIN -SPECIFIC
     4     |-TABLE SCAN(ADBUSER01.T1(A))
     5     +-TABLE SCAN(ADBUSER01.T2(B))
```

Explanation:

The join method specification specified in the SELECT statement is enabled, and a hash join was executed in the join processing for the table.

(29) Scan information for table value constructors

The following information is displayed:

- TABLE VALUE CONSTRUCTOR SCAN

  Table value constructors are scanned.

Output example

```
<<Tree View>>

     1 QUERY : 1
     2   SELECT STATEMENT
     3   +-NESTED LOOP JOIN
     4     |-TABLE SCAN(ADBUSER01.T1)
     5     +-DERIVED TABLE(DT)
     6     +-TABLE VALUE CONSTRUCTOR SCAN
```

Explanation:

Table value constructors are scanned.

6.1.5 Information displayed in the details view

The following information is displayed in the details view:

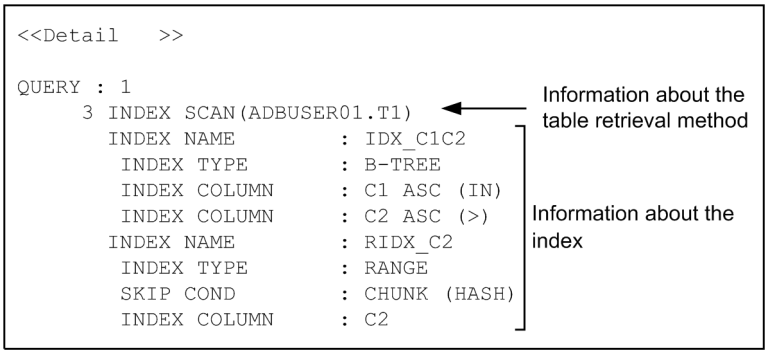
- Information about the table retrieval methods and indexes
- Information about the table joining methods
- Information about set operations
- Information about table function derived tables
- Information about subqueries
- Information about the grouping

Note that the details view is displayed only when there is detailed information for the query that is being displayed in the query tree.

(1) Information related to table retrieval methods and indexes

The following shows an example of the output format for information related to table retrieval methods and indexes.

Output example



(a) Information related to table retrieval methods

One of the following is displayed as the information related to table retrieval methods:

- `TABLE SCAN`

  A table scan is performed in the table retrieval processing.
- `INDEX SCAN` (*schema-name*`.`*table-identifier* (*query-name-or-correlation-name*))

  An index scan is performed in the table retrieval processing. If there is a query name or a correlation name, it is displayed.
- `KEY SCAN`

  A key scan is performed in the table retrieval processing.

For details about table scans, index scans, and key scans, see 5.1 How to retrieve tables.

Output example

```
<<Detail   >>

QUERY : 1
     3 INDEX SCAN(ADBUSER01.T1)
       INDEX NAME         : IDX_C1C2
        INDEX TYPE        : B-TREE
        INDEX COLUMN      : C1 ASC (IN)
        INDEX COLUMN      : C2 ASC (>)
       INDEX NAME         : RIDX_C2
```

```
        INDEX TYPE        : RANGE
        SKIP COND         : CHUNK (HASH)
        INDEX COLUMN      : C2
```

Explanation:

An index scan is performed in the table T1 retrieval processing.

(b) Information related to indexes

The following shows an output format for the information related to indexes.

Output format (B-tree index)

```
INDEX NAME         : B-tree-index-name (uniqueness-constraint-information)
 INDEX TYPE        : index-type
 INDEX COLUMN      : indexed-column-name key-value-sort-order (range-search-conditi
on)
```

- *uniqueness-constraint-information*

  If this B-tree index is a unique index, uniqueness constraint information is displayed. One of the following is displayed:

  UNIQUE: This unique index does not violate the uniqueness constraint.

  UNIQUE INVALID: This unique index violates the uniqueness constraint.

- *index-type*

  For a B-tree index, B-TREE is displayed as the index type.

- *key-value-sort-order*

  Displays the sort order for the key values of the B-tree index that was specified when the B-tree index was defined. One of the following is displayed:

  ASC: The key values are sorted in ascending order.

  DESC: The key values are sorted in descending order.

- *range-search-condition*

  Displays one of the following as the range search condition:

  =, <, <=, >, >=, =ANY, BETWEEN ({<,<|<,<=|<=,<|<=,<=}), IN, LIKE, IS NULL

Rules for search condition output

- If no range search condition is specified for an indexed column, none is displayed.
- If IN *table-subquery* or *quantified-predicate*=SOME is specified as the range search condition, =ANY is displayed.
- If the left side of the comparison predicate is not a single column specification, the HADB server performs equivalent exchange of the search condition. The comparison operator of the comparison predicate after equivalent exchange is output as the search condition.

  Example:

  Search condition specified in WHERE clause of SELECT statement

  WHERE 10 ≤ C1

  Search condition after equivalent exchange by HADB server

  WHERE C1 ≥ 10

  Information output as search condition in access path information

  INDEX COLUMN: C1 ASC (≥)

Output format (text index)

```
INDEX NAME          : text-index-name
 INDEX TYPE         : index-type
 INDEX COLUMN       : indexed-column-name
```

- *index-type*

  For a text index, TEXT is displayed as the index type.

Output format (range index)

```
INDEX NAME          : range-index-name
 INDEX TYPE         : index-type
 SKIP COND          : range-index-condition-type
 INDEX COLUMN       : indexed-column-name
```

- *index-type*

  For a range index, RANGE is displayed as the index type.

- *range-index-condition-type*

  Displays one of the following as the range index condition to be used:

  - CHUNK: The chunk skip condition is used.
  - SEGMENT: The segment skip condition is used.
  - CHUNK AND SEGMENT: The chunk skip condition and the segment skip condition are both used.

  When a hash join is performed, (HASH) is output if a range index that is defined for a column that is to be matched with a hash table is used. When hash execution is performed as the processing method for a subquery, (HASH) is output if a range index that is defined for a column that is to be matched with a hash table is used.

  Output example

  ```
  SKIP COND          : CHUNK (HASH)
  ```

  For details about a hash join, see 5.5.2 About hash join.

  For details about hash execution as the processing method for a subquery, see either (4) Hash execution in 5.6.1 Methods for processing subqueries that do not contain an external reference column or (3) Hash execution in 5.6.3 Methods for processing subqueries that contain an external reference column.

Output example

```
<<Detail   >>

QUERY : 1
     3 INDEX SCAN(ADBUSER01.T1)
       INDEX NAME          : IDX_C1C2          ...1
        INDEX TYPE         : B-TREE            ...2
        INDEX COLUMN       : C1 ASC (IN)       ...3
        INDEX COLUMN       : C2 ASC (>)        ...3
       INDEX NAME          : RIDX_C2           ...4
        INDEX TYPE         : RANGE             ...5
        SKIP COND          : CHUNK (HASH)      ...6
        INDEX COLUMN       : C2                ...7
```

Explanation:

1. Name of the index to be used
2. Type of the index displayed by INDEX NAME in 1 In this example, IDX_C1C2 is a B-tree index because B-TREE is displayed.

   The information displayed in 1 and 2 indicates that an index scan is performed using B-tree index IDX_C1C2.

3. Information about B-tree index IDX_C1C2:

   C1, C2: Indexed column

   ASC: Key value sort order

   (IN), (>): Range search condition specification

4. Name of the index to be used

5. Type of the index displayed by INDEX NAME in 4 In this example, RIDX_C2 is a range index because RANGE is displayed.

6. Type of range index condition to be used

   CHUNK: Indicates that range index RIDX_C2 is used for a chunk skip condition.

   (HASH): When a hash join is performed, (HASH) indicates that the range index RIDX_C2 that is defined for a column that is to be matched with a hash table is used. When hash execution is performed as the processing method for a subquery, (HASH) indicates that the range index RIDX_C2 that is defined for a column that is to be matched with a hash table is used.

7. Indexed column of range index RIDX_C2

Note

- Information about the B-tree index is displayed when an index scan or a key scan is performed for table retrieval.

  If there are multiple indexed columns, information is displayed for each indexed column. In such a case, the information is displayed in the order the indexed columns were defined in the CREATE INDEX statement.

- Information about a text index is displayed when an index scan is implemented as the table retrieval method.

- Information about a range index is displayed whenever there is a range index condition.

(2) Information related to table joining methods

The following shows an example of the output format for information related to table joining methods.

Example of the output format

```
<<Detail   >>

QUERY : 1
     3 HASH JOIN                              ← Table joining method
       JOIN TYPE          : INNER JOIN        ← Join type
       BUILD COLUMN       : ADBUSER01.T1.C3             ← Hash retrieval information
       PROBE COLUMN       : ADBUSER01.T2.C3             ← Hash retrieval information
```

(a) Table joining method

One of the following is displayed as the table joining method:

- NESTED LOOP JOIN

  A nested loop join is performed in the table join processing.

- HASH JOIN

  A hash join is performed in the table join processing.

If there is an = condition that compares columns in two tables, a hash join might be performed. If there are rows that cannot be processed in the hash table area, work tables might be created.

For details about table joining methods, see 5.5 Table joining methods.

Output example

```
<<Detail   >>

QUERY : 1
     3 HASH JOIN
       JOIN TYPE          : INNER JOIN
       BUILD COLUMN       : ADBUSER01.T1.C3
       PROBE COLUMN       : ADBUSER01.T2.C3
```

Explanation:

A hash join is applied as the table joining method.

(b) Join type (JOIN TYPE)

One of the following is displayed as `JOIN TYPE`:

- `CROSS JOIN`

  A cross join is specified.
- `INNER JOIN`

  An inner join is specified.
- `LEFT OUTER JOIN`

  An outer join performed by a `LEFT OUTER JOIN` is specified.
- `RIGHT OUTER JOIN`

  An outer join performed by a `RIGHT OUTER JOIN` is specified.
- `FULL OUTER JOIN(LEFT)`

  An outer join performed by a `FULL OUTER JOIN` is specified.
- `FULL OUTER JOIN(RIGHT)`

  An outer join performed by a `FULL OUTER JOIN` is specified.

For details about each join method, see the topic *Specification format and rules for joined tables* in the manual *HADB SQL Reference*.

Output example

```
<<Detail   >>

QUERY : 1
     3 HASH JOIN
       JOIN TYPE          : INNER JOIN
       BUILD COLUMN       : ADBUSER01.T1.C3
       PROBE COLUMN       : ADBUSER01.T2.C3
```

Explanation:

An inner join is performed.

Important

INNER JOIN or CROSS JOIN might be converted to a comma join during execution of an SQL statement. In this case, the join type (JOIN TYPE) is not displayed. For details about a comma join, see *Explanation of specification format* in *Specification format and rules for FROM clauses* in the manual *HADB SQL Reference*.

The output rules are as follows:

- For an SQL statement that creates an internal derived table, the results of expanding the internal derived table are output as the table name and correlation name of the outer table columns and inner table columns for the hash join. For details about internal derived tables and the internal derived table expansion rules, see *Internal derived tables* in the manual *HADB SQL Reference*.

(c) Hash retrieval information (BUILD COLUMN and PROBE COLUMN)

When hash join is performed, the following information is displayed as hash retrieval information:

- BUILD COLUMN

  Information about the joined columns in the outer table
- PROBE COLUMN

  Information about the joined columns in the inner table

Output example

```
<<Detail   >>

QUERY : 1
     3 HASH JOIN
       BUILD COLUMN      : ADBUSER01.T1.C1(CREATE FILTER 1)     ...1
       PROBE COLUMN      : ADBUSER01.T2.C1(USE FILTER 1)        ...2
```

Explanation:

1. Column name ADBUSER01.T1.C1 of the joined column in the outer table during hash join is displayed.
2. Column name ADBUSER01.T2.C1 of the joined column in the inner table during hash join is displayed.

If a hash filter is applied during hash join, hash filter information is displayed in the underlined portion.

The output rules are as follows:

- For BUILD COLUMN or PROBE COLUMN, the column name is displayed in one of the following formats. Output of a correlation name has the highest priority among those column names. If a column name cannot be displayed, three asterisks (***) are displayed.
  - *table-name*.*column-name*
  - *query-name*.*column-name*
  - *correlation-name*.*column-name*
- If a hash filter is applied during hash join, (CREATE FILTER XXXXX) is displayed for BUILD COLUMN. XXXXX is the number of the hash filter created based on the column value for the displayed column name.

  (USE FILTER XXXXX) is displayed for PROBE COLUMN. XXXXX is the number of the hash filter to be used.
- For an SQL statement that creates an internal derived table, the results of expanding the internal derived table are output as the table name and correlation name of the outer table columns and inner table columns for the hash join.

For details about internal derived tables and the internal derived table expansion rules, see *Internal derived tables* in the manual *HADB SQL Reference*.

(3) Information related to set operations

In information related to set operations, the type of set operation is output in the following format:

SET OPERATION TYPE:set-operation-type

One of the following is output as the set operation type:

- `UNION ALL`
  The set operation `UNION ALL` is specified.
- `UNION DISTINCT`
  The set operation `UNION DISTINCT` is specified.
- `EXCEPT ALL`
  The set operation `EXCEPT ALL` is specified.
- `EXCEPT DISTINCT`
  The set operation `EXCEPT DISTINCT` is specified.
- `INTERSECT ALL`
  The set operation `INTERSECT ALL` is specified.
- `INTERSECT DISTINCT`
  The set operation `INTERSECT DISTINCT` is specified.

The following is an example of the output format of information related to set operations:

Output example

```
<<Detail   >>

QUERY : 0
     3 SET OPERATION
       SET OPERATION TYPE : UNION ALL
```

Explanation:

The set operation `UNION ALL` is specified.

Notes

- When a set operation that specifies `ALL` and one that specifies `DISTINCT` are specified consecutively in an SQL statement, access path information might be output that interprets the `ALL` set operation as a `DISTINCT` set operation. In this case, the set operations that are consecutively specified will appear combined in the output shown in (4) Specification of set operations in 6.1.4 Information displayed in the tree view. The corresponding information related to the set operations will also be combined.
- If both of the following conditions are met, access path information might be output that interprets the `UNION DISTINCT` set operation as a `UNION ALL` set operation.
  - Set operations that are specified consecutively in an SQL statement contain set operations with `UNION`, `UNION ALL`, and `UNION DISTINCT` specified.
  - Hash execution is applied as the method for processing the set operation.

In this case, the set operations that are consecutively specified will appear combined in the output shown in (4) Specification of set operations in 6.1.4 Information displayed in the tree view. The corresponding information related to the set operations will also be combined.

(4) Information about table function derived tables

In the information about table function derived tables, the names of system-defined functions to be executed are output in the following format:

FUNCTION NAME : schema-name.system-defined-function-name

Either of the following is output as the system-defined function:

- ADB_AUDITREAD

  The ADB_AUDITREAD function is specified.

- ADB_CSVREAD

  The ADB_CSVREAD function is specified.

The following is an example of the output format of the information about table function derived tables:

Output example

```
<<Detail   >>

QUERY : 1
     3 TABLE FUNCTION DERIVED TABLE(T5)
       FUNCTION NAME : MASTER.ADB_AUDITREAD
```

Explanation:

The ADB_AUDITREAD function is specified.

(5) Information about subqueries

If hash execution is applied during subquery processing, hash retrieval information is output in information about subqueries. The following information is output as hash retrieval information:

- BUILD COLUMN

  If hash execution is applied as the method for processing subqueries that do not contain external reference columns, information about the column resulting from the subquery is output.

  If hash execution is applied as the method for processing subqueries that contain external reference columns, information about the column to be compared with the external reference column specified in the subquery is output.

- PROBE COLUMN

  If hash execution is applied as the method for processing subqueries that do not contain external reference columns, column information to be compared with the result of the subquery specified in the search condition is output.

  If hash execution is applied as the method for processing subqueries that contain external reference columns, information about the external reference column specified in the subquery is output.

Output example

```
<<Detail   >>

QUERY : 1
     3 SUBQUERY HASH
```

```
        BUILD COLUMN      : ADBUSER01.T2.C1(CREATE FILTER 1)     ...1
        PROBE COLUMN      : ADBUSER01.T1.C1(USE FILTER 1)        ...2
```

Explanation:

In the preceding example, hash execution is applied as the method for processing subqueries that do not contain external reference columns.

1. Column name `ADBUSER01.T2.C1` of the column resulting from the subquery is output.
2. Column name `ADBUSER01.T1.C1` of the column to be compared with the result of the subquery specified in the search condition is output.

If a hash filter is applied during hash execution, hash filter information is output in the underlined portion.

The output rules are as follows:

- For `BUILD COLUMN` or `PROBE COLUMN`, the column name is displayed in one of the following formats. Output of a correlation name has the highest priority among those column names. If a column name cannot be displayed, three asterisks (`***`) are displayed.
  - *table-name*.*column-name*
  - *query-name*.*column-name*
  - *correlation-name*.*column-name*
- If a hash filter is applied during hash execution, (`CREATE FILTER XXXXX`) is displayed for `BUILD COLUMN`. `XXXXX` is the number of the hash filter created based on the column value for the displayed column name.

  (`USE FILTER XXXXX`) is displayed for `PROBE COLUMN`. `XXXXX` is the number of the hash filter to be used.
- When an internal derived table is expanded by using an SQL statement that creates an internal derived table, the results of expanding the internal derived table are output as the table name and correlation name output for `BUILD COLUMN` and `PROBE COLUMN`. For details about internal derived tables and the internal derived table expansion rules, see *Internal derived tables* in the manual *HADB SQL Reference*.

(6) Information about the grouping

When the grouping processing is performed multiple times, information about the grouping (grouping set information) is output in the following format:

```
GROUPING SET : {table-name|query-name|correlation-name}.grouping-column-name-1-for-gr
ouping-process-1
               {table-name|query-name|correlation-name}.grouping-column-name-2-for-gr
ouping-process-1
                              :
GROUPING SET : {table-name|query-name|correlation-name}.grouping-column-name-1-for-gr
ouping-process-2
               {table-name|query-name|correlation-name}.grouping-column-name-2-for-gr
ouping-process-2
                              :
GROUPING SET : {table-name|query-name|correlation-name}.grouping-column-name-1-for-gr
ouping-process-N
               {table-name|query-name|correlation-name}.grouping-column-name-2-for-gr
ouping-process-N
                              :
```

The following shows an example of the output format of the information about the grouping.

Example of the output format

```
<<Detail   >>
QUERY : 1
     5 GLOBAL HASH GROUPING
       GROUPING SET : ADBUSER01.T1.C1
                      ADBUSER01.T1.C2
                      ADBUSER01.T1.C3
       GROUPING SET : ADBUSER01.T2.C1
                      ADBUSER01.T2.C2
       GROUPING SET : ADBUSER01.T3.C1
                      ADBUSER01.T3.C3
```

Explanation:

A grouping column name is output for each grouping.

If no grouping column name can be output, three asterisks (`***`) are output.

The output rules are as follows:

- When an internal derived table is expanded by using an SQL statement that creates an internal derived table, the results of expanding the internal derived table are output as the table name and correlation name that are output in the information about the grouping. For details about internal derived tables and the internal derived table expansion rules, see *Internal derived tables* in the manual *HADB SQL Reference*.

6.1.6 Information output in identification information view (SQL statement identification information)

The identification information view displays SQL statement identification information that is used to identify the SQL statement for which HADB acquired access path statistical information. Based on the information displayed in this view, you can identify the correlation between the access path information and the access path statistical information.

For details about access path statistical information, see *Examples of output of and output items for access path statistical information* in the *HADB Setup and Operation Guide*.

The following is an example of the information displayed in the identification information view.

Output example

```
<<SQL Info >>

Version           : 03-01(Apr 23 2015 15:32:27)
Transaction ID    : 1
Connection Number : 1
SQL Serial Number : 1
```

Explanation:

- `Version`

  The version of the HADB server that ran the SQL statement for which the access path statistical information was acquired.

  The information in parentheses is additional version information.

- `Transaction ID`

  The transaction ID of the SQL statement for which the access path statistical information was acquired.

- `Connection Number`

  The connection sequence number of the SQL statement for which the access path statistical information was acquired.

- `SQL Serial Number`

  The SQL statement sequence number of the SQL statement for which the access path statistical information was acquired.

Note

This information corresponds to the following information in the SQL statement execution information output before the access path statistical information. The header name for the information is in parentheses.

- Transaction ID (`tran_id`)
- Connection sequence number (`con_num`)
- SQL statement sequence number (`sql_serial_num`)

For details about SQL statement execution information, see *SQL statement execution information* under *Information that is output as SQL trace information* in the *HADB Setup and Operation Guide*.

6.1.7 Information displayed for an access path (alphabetical order)

The following table lists in alphabetical order and describes the information that is displayed for an access path.

Table 6-1: Information displayed for an access path (alphabetical order)

First letter	Output information	Description	Classification
B	BUILD COLUMN	If hash join is applied as the table joining method, information about the joined columns in the outer table is output. If a hash filter is applied, hash filter information is also output.	Hash retrieval information (BUILD COLUMN and PROBE COLUMN)
		If hash execution is applied as the method for processing subqueries that do not contain external reference columns, information about the column resulting from the subquery is output. If hash execution is applied as the method for processing subqueries that contain external reference columns, information about the column to be compared with the external reference column specified in the subquery is output. If a hash filter is applied, hash filter information is also output.	Information about subqueries
C	CHUNK	The chunk skip condition is used.	Information related to indexes
	CHUNK AND SEGMENT	The chunk skip condition and the segment skip condition are both used.	
	COLUMN	The grouping method that uses the characteristics of column store tables is performed.	Grouping methods
	COLUMN STORE	The table-data storage format is column store format.	Table-data storage format

First letter	Output information	Description	Classification
	CREATE FILTER	The number of the hash filter created based on the column value for the column name shown in BUILD COLUMN is output.	Hash retrieval information (BUILD COLUMN and PROBE COLUMN)
			Information about subqueries
	CREATE GLOBAL WORK TABLE (WORK TABLE *work-table-number*)	A global work table is created.	Work table creation information
	CREATE LOCAL WORK TABLE (WORK TABLE *work-table-number*)	A local work table is created. A unique work table number is assigned to each work table.	
	CROSS JOIN	A cross join is specified.	Information related to table joining methods
D	DELEGATION	Subquery processing delegation specification is enabled.	Subquery processing delegation specification
	DELETE STATEMENT	A DELETE statement is executed.	SQL statements executed
	DERIVED TABLE (*correlation-name, query-name, or table-identifier)*)	A derived table, viewed table, or query name is specified.	Specification of derived tables
	DISTINCT	This item indicates that duplicate removal will be performed.	Information about duplicate removal
F	FILTER	A hash filter is applied during hash execution for the subquery processing method.	Subquery processing methods
		A hash filter is applied during hash join for the table joining method.	Table joining method
	FULL OUTER JOIN (LEFT) FULL OUTER JOIN (RIGHT)	An outer join performed by a FULL OUTER JOIN is specified.	Information related to table joining methods
	FUNCTION NAME	The type of system-defined function that is executed.	Information about table function derived tables
G	GLOBAL HASH GROUPING	Global hash grouping is executed. Work tables might be created if there are rows that cannot be processed in the hash table area.	Grouping methods
	GLOBAL HASH UNIQUE	Duplication in the retrieval results has been eliminated by using one of the following methods: • Hash execution for the method for processing the set operation • Hash execution for the method for processing SELECT DISTINCT • Global hash grouping for the grouping method Work tables are created if there are rows that cannot be processed in the hash table area.	Processing method for duplicate removal
	GROUPING	Grouping that does not use work tables is performed.	Grouping methods
	GROUPING SET	Grouping processing is performed multiple times.	• Grouping methods • Information about the grouping
H	(HASH)	When a hash join is performed, (HASH) indicates that the range index that is defined for a column that is to be matched with a hash table is used. When hash	Information related to indexes

First letter	Output information	Description	Classification
		execution is performed as the processing method for a subquery, (HASH) indicates that the range index that is defined for a column that is to be matched with a hash table is used.	
	HASH JOIN	A hash join is performed in the table join processing. If there is an = condition that compares columns in two tables, a hash join might be performed. If there are rows that cannot be processed in the hash table area, work tables might be created.	• Table joining method (tree view) • Table joining method (details view)
	HAVING	The HAVING clause is specified. This item might also be displayed when a derived table is expanded even though the HAVING clause is not specified.	HAVING clause specification
I	INDEX	The grouping method that uses the characteristics of B-tree indexes is performed.	Grouping methods
	INDEX COLUMN	Information related to indexed columns. For B-tree indexes, the name of the indexed column, the key value sort order, and the search condition are output. For text indexes and range indexes, the name of the indexed column is output.	Information related to indexes
	INDEX NAME	Indicates the index name.	
	INDEX SCAN (*schema-name*.*table-identifier* (*query-name-or-correlation-name*))	An index scan is performed in the table retrieval processing. If there is a query name or a correlation name, it is displayed.	• Table retrieval method (tree view) • Table retrieval method (details view)
	INDEX TYPE	The index type (B-tree index, text index, or range index).	Information related to indexes
	INNER JOIN	An inner join is specified.	Information related to table joining methods
	INSERT STATEMENT	An INSERT statement is executed.	SQL statements executed
J	JOIN TYPE	Join type	Information related to table joining methods
K	KEY SCAN	A key scan is performed in the table retrieval processing.	• Table retrieval method (tree view) • Table retrieval method (details view)
L	LIMIT *offset*,*row_count*	The LIMIT clause is specified. If the offset row count is not specified, the offset row count is not displayed. If dynamic parameters are specified for either or both of the offset row count and limit row count, ? PARAMETER is displayed.	LIMIT clause specification
	LEFT OUTER JOIN	An outer join performed by a LEFT OUTER JOIN is specified.	Information related to table joining methods
	LOCAL HASH GROUPING	Local hash grouping is performed. If there are rows that cannot be processed in the hash grouping area, work tables might be created.	Grouping methods
N	NESTED LOOP JOIN	A nested loop join is performed in the table join processing.	• Table joining methods (tree view)

First letter	Output information	Description	Classification
			• Table joining methods (details view)
O	ORDER	Sequential execution, not out-of-order execution, is applied.	Sequential execution
P	PROBE COLUMN	If hash join is applied as the table joining method, information about the joined columns in the inner table is output. If a hash filter is applied, hash filter information is also output.	Hash retrieval information (BUILD COLUMN and PROBE COLUMN)
		If hash execution is applied as the method for processing subqueries that do not contain external reference columns, column information to be compared with the result of the subquery specified in the search condition is output. If hash execution is applied as the method for processing subqueries that contain external reference columns, information about the external reference column specified in the subquery is output. If a hash filter is applied, hash filter information is also output.	Information about subqueries
	PURGE CHUNK STATEMENT	A PURGE CHUNK statement is executed.	SQL statements executed
Q	QUERY	A query other than a subquery or a derived table is specified.	Query types
	QUERY SCAN (QUERY *query-tree-number*)	A query scan is performed.	Query scan
R	RECURSIVE	This item indicates that a recursive query will be run.	Specification of set operations
	RIGHT OUTER JOIN	An outer join performed by a RIGHT OUTER JOIN is specified.	Information related to table joining methods
S	SEGMENT	The segment skip condition is used.	Information related to indexes
	SELECT STATEMENT	A SELECT statement is executed.	SQL statements executed
	SET OPERATION	A set operation is specified.	Specification of set operations
	SET OPERATION TYPE	The type of set operation that is executed.	Information related to set operations
	SKIP COND	Type of range index condition.	Information related to indexes
	SORT GROUPING	Sort grouping is performed.	Grouping methods
	SORTING {BYTE|ISO}	Sort processing by the ORDER BY clause is performed. This information might not be displayed even though the ORDER BY clause is specified.	Sort processing
	SPECIFIC	Set operation method specification is enabled.	Set operation method specification
		Subquery processing method specification is enabled.	Subquery processing method specification

First letter	Output information	Description	Classification
		The grouping method specification specified in the GROUP BY clause is enabled.	Grouping methods
		SELECT deduplication method specification is enabled.	SELECT deduplication method specification
		The index specification is enabled.	Index specification
		Join method specification is enabled.	Join method specification
	SPECIFIC DISABLED	The index specification is disabled.	Index specification
		The join method specification is disabled.	Join method specification
	SUBQUERY	A subquery processing method other than nested loop execution or hash execution is applied.	Subquery processing methods
	SUBQUERY HASH	Hash execution is applied as the subquery processing method.	
	SUBQUERY LOOP	Nested loop execution is applied as the subquery processing method.	
T	TABLE FUNCTION DERIVED TABLE (correlation name)	A table function derived table is specified.	Specification of table function derived table
	TABLE SCAN	A table scan is performed in the table retrieval processing.	• Table retrieval method (tree view) • Table retrieval method (details view)
	TABLE VALUE CONSTRUCTOR SCAN	Table value constructors are retrieved.	Information retrieved for table value constructors
U	UNIQUE	This unique index does not violate the uniqueness constraint.	Information related to indexes
	UNIQUE INVALID	This unique index violates the uniqueness constraint.	
	UPDATE STATEMENT	An UPDATE statement is executed.	SQL statements executed
	USE FILTER	The number of the hash filter to be used is output.	Hash retrieval information (BUILD COLUMN and PROBE COLUMN)
			Information about subqueries
	USING CACHE	Cache is used to store the results of a subquery.	Subquery cache usage information
	USING COST	Cost information is collected in relation to a table or index.	Collecting cost information
W	WINDOW	A window function is specified.	Specification of window functions
	WORK TABLE SCAN (WORK TABLE *work-table-number*)	A work table is scanned. A unique work table number is assigned to each work table.	Work table scan

7 Creating Application Programs

This chapter explains how to create application programs that use the JDBC driver. For details about how to set up an environment for the JDBC driver, see 3. Setting Up an Environment for the JDBC Driver.

7.1 JDBC driver provided by HADB

This section explains the scope of the JDBC standard with which the JDBC driver provided by HADB is compliant. This section also explains the package name and directory structure of the JAR file.

7.1.1 Scope of JDBC standards compliance

HADB is implemented with the Type 4 JDBC driver. The following table shows the scope of the JDBC standards with which the HADB JDBC driver is compliant.

Table 7-1: Scope of the JDBC standards with which the HADB JDBC driver is compliant

No.	JDBC standard	Function	Compliant
1	The JDBC™ API, Version 1.20 (JDBC 1.2 API)	`Driver` interface	Y
2		`Connection` interface	Y
3		`Statement` interface	Y
4		`PreparedStatement` interface	Y
5		`CallableStatement` interface	N
6		`ResultSet` interface	Y
7		`DatabaseMetaData` interface	Y
8		`ResultSetMetaData` interface	Y
9		`Blob` interface	N
10		`Array` interface	N
11		`SQLException` interface	Y
12		`SQLWarning` interface	Y
13	The JDBC™ 2.1 API, Version 1.1 (the JDBC 2.1 Core API)	Result set extensions	L[#1]
14		Batch updating	Y
15		Support for persistent Java objects	N
16		Addition of JDBC SQL data types	N
17		Custom mapping of data types	N
18	The JDBC 2.0 Standard Extension API, Version 1.0 (the JDBC 2.0 Optional Package API)	JNDI	Y
19		Connection pool	Y
20		Distributed transactions (JTA support)	N
21		Row sets	N
22	JDBC™ 3.0 Specification (JDBC 3.0 API)	Save points	N
23		Enhancement of the connection pool function	N
24		Parameter metadata	Y
25		Automatic generation keys	N

No.	JDBC standard	Function	Compliant
26		Concurrent opening of multiple result sets	N
27		Use of parameter names in `CallableStatement`	N
28		Holdable cursors	Y
29		`BOOLEAN` type	N
30		Data manipulation in the `Blob` class	N
31		Reference type	N
32		`DATALINK` and `URL` types	N
33		JCA-related architectures	N
34		API for adding database metadata	Y
35	JDBC™ 4.0 Specification (JDBC 4.0 API)	Automatic loading of `java.sql.Driver`	Y
36		`ROWID` data type	N
37		National character data type	N
38		Enhancement of `BLOB` and `CLOB` functions	N
39		XML support	N
40		Wrapper pattern	Y
41		SQL exception extensions	Y
42		Connection management	L[#2]
43		Addition of scalar functions	L[#3]
44		API for adding database metadata	Y
45	JDBC™ 4.1 Specification (JDBC 4.1 API)	`try-with-resources` statement	Y
46		Java data type of conversion target of `getObject` method	L[#4]
47		Acquisition of parent logger	N
48		Schema specification	N
49		Closing and timing out physical connections	N
50		Closing `Statement` objects when dependent objects are closed	Y
51		API that adds database metadata	Y
52	JDBC™ 4.2 Specification (JDBC 4.2 API)	`REF CURSOR`	N
53		`SQLType` interface	N
54		`JDBCType` enumerator (`Enum`)	N
55		Large update counts	Y
56		API that adds database metadata	Y

Legend:

Y: Supported

L: Limited support

N: Not supported

#1

Only the scroll function is supported.

#2

Only `Connection#isValid()` and `Statement#isPoolable()` are supported.

#3

Only `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, and `EXTRACT` are supported.

#4

A `getObject` method of the `ResultSet` interface. This method only supports conversion to some Java data types. For details, see 8.5.43 getObject(int columnIndex,Class<T> type).

Hereafter, the term *JDBC driver* refers to the Type 4 JDBC driver.

7.1.2 Package name and directory structure of the JAR file

The JAR file's package name and directory structure are as follows:

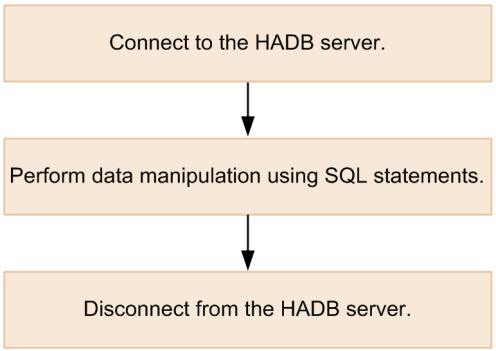
- Package name: `com.hitachi.hadb.jdbc`
- Directory structure: `com/hitachi/hadb/jdbc/`

> **Note**
>
> *HADB* and *hadb* are acronyms for Hitachi Advanced Database.

7.2 Basic procedure for application program processing

The following figure shows the basic procedure for using an application program to manipulate a database.

Figure 7-1: Basic procedure for using an application program to manipulate a database



Explanation:

1. Use the `getConnection` method of the `DriverManager` or `DataSource` class to connect to the HADB server. For details about how to connect to the HADB server, see 7.3 How to connect to the HADB server.
2. Execute SQL statements to manipulate data. If you execute a `SELECT` statement, see 7.4 Retrieving data (executing the SELECT statement).

   If you execute an `INSERT`, `UPDATE`, or `DELETE` statement, see 7.5 Adding, updating, or deleting data (executing the INSERT, UPDATE, or DELETE statement).
3. Use the `close` method of the `Connection` object to disconnect from the HADB server.

The JDBC API packages are explained in detail beginning from 8. The JDBC 1.2 API.

■ Notes about creating multithreaded applications

When all of the following conditions are met, a wait condition might arise due to serialization.

- The application is designed to use multiple `Statement` objects created from the same `Connection` object in different threads.
- SQL statements are executed concurrently using these `Statement` objects.

Because this wait condition occurs before the SQL statements are executed, it is not included in the timer monitoring time for the SQL execution processing. This means that a timeout error might not occur even if the timer monitoring time specified in the `adb_clt_rpc_sql_wait_time` property or by the `setQueryTimeout` method is exceeded.

7.3 How to connect to the HADB server

You must first connect to the HADB server to access the HADB database. There are two ways to connect to the HADB server:

- Use the `getConnection` method of the `DriverManager` class.
- Use the `getConnection` method of the `DataSource` class.

Note that you must have the `CONNECT` privilege to execute the `getConnection` method.

The following subsections provide details about both methods.

7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server

Connect to the HADB server by executing the `getConnection` method of the `DriverManager` class. Before the `getConnection` method is executed, the `Driver` class is automatically registered on the Java Virtual Machine (JVM). (You can also manually perform registration. For details, see (1) How to register the Driver class into the Java Virtual Machine (JVM).) You can then execute the `getConnection` method to connect to the HADB server.

The following subsections explain the steps.

(1) How to register the Driver class into the Java Virtual Machine (JVM)

Register the `Driver` class into the Java Virtual Machine (JVM). Note that when you register the `Driver` class into the Java Virtual Machine (JVM), you need a driver name (*package-name*`.`*class-name*). The package name and class name of the JDBC driver are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `HADBDriver`

> **Note**
>
> *HADB* and *hadb* are acronyms for Hitachi Advanced Database.

There are three ways to register the `Driver` class, as shown below.

■ Method 1 (using the `forName` method of the `Class` class)

Execute the `forName` method of the `Class` class within the application as follows:

```
Class.forName("com.hitachi.hadb.jdbc.HADBDriver");
```

■ Method 2 (using the system properties)

Specify the following value in the Java Virtual Machine's (JVM) system property (`jdbc.drivers`):

```
System.setProperty("jdbc.drivers", "com.hitachi.hadb.jdbc.HADBDriver");
```

■ Method 3 (using the operation settings file for the Java Virtual Machine)

This method is applicable only to Java Applets.

Specify the information provided below in the *JAVA_HOME*\.hotjava\properties file (*JAVA_HOME* depends on the Java execution environment). If you register multiple JDBC drivers, delimit then with the colon (:).

```
jdbc.drivers="com.hitachi.hadb.jdbc.HADBDriver"
```

(2) Connecting to the HADB server with the getConnection method

Connect to the HADB server by executing the `getConnection` method of the `DriverManager` class. When a connection to the HADB server is established successfully, the JDBC driver returns a reference to a `Connection` class instance as the result of executing the method. If connection establishment with the HADB server fails, as in the following cases, the method throws an `SQLException`:

- Required connection information is not specified in the arguments.
- Specified connection information is invalid.
- Connection cannot be established (for example, because the HADB server at the connection destination has not been started).

The `getConnection` method is provided in the following three formats, each with its own set of arguments (`url`, `user`, `password`, and `info`) that specify information about the connection to the HADB server:

- `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 following subsections explain the values to be specified in these arguments.

(a) Values to be specified in the url argument (specifying the URL for the connection)

You specify in the `url` argument the URL to be used for the connection. The following shows the URL specification format:

```
jdbc:hadb[://[host][:port]/[?property=value[&property=value]...]]
```

> **Note**
>
> *HADB* and *hadb* are acronyms for Hitachi Advanced Database.

Examples of URL specification

- Example 1: Omitting the property

  ```
  jdbc:hadb://localhost:23650/
  ```

- Example 2: Specifying one property

  ```
  jdbc:hadb://localhost:23650/?adb_clt_ap_name=AP001
  ```

- Example 3: Specifying multiple properties

  ```
  jdbc:hadb://localhost:23650/?methodtrace=ON&tracenum=600
  &sqlwarningkeep=FALSE&user=ADBUSER01&password=password01&adb_clt_ap_name=AP001
  ```

URL specification rules

- Spaces are not allowed within an item or between items in the `URL` argument.
- Each item name is case sensitive.
- Specification of an item enclosed in square brackets (`[ ]`) is optional.
- Specify a question mark (`?`) before specifying the first property (`property`) and use the ampersand (`&`) as the delimiter between properties.
- If the same property is specified more than once, the first value specified takes effect.
- Ampersands (`&`) cannot be used as property values. If a password contains an ampersand, use another connection method because the `password` property cannot be specified in a URL. For details about passwords in HADB, see the topic *Password specification rules* in the *HADB Setup and Operation Guide*.
- When an invalid value is specified in a property in a URL, `SQLException` will not be thrown if the correct value is specified for a user property of the same name.

Explanation of each URL item

- `jdbc:hadb`

  This item consists of the protocol name and subprotocol name. You must specify this item. This item is case sensitive.

- `host`

  Specifies the host name of the HADB server at the connection destination. This host name is used for communication between the HADB client and the HADB server.

  You can use other methods of specifying the HADB server's host name. For details about other specification methods and priorities, see 7.3.3 Connection information priorities.

  If the `getConnection` method is executed with no host name specified, the method throws an `SQLException`.

  In a cold standby configuration, specify the alias IP address used for communication between the HADB server and HADB client.

  Multi-node function:

  When you use the multi-node function, specify the alias IP address that is used for communication between the HADB server and the HADB client.

- `port`

  Specifies the port number of the HADB server that is used for communication between the HADB client and the HADB server.

  You can use other methods of specifying the HADB server's port number. For details about other specification methods and priorities, see 7.3.3 Connection information priorities.

  If the `getConnection` method is executed with no port number specified, the method throws an `SQLException`.

- `property=value`

  Specifies a property (`property`) and a value (`value`) for that property.

  The following table lists and describes the properties that can be specified in the `url` argument.

Table 7-2: Properties that can be specified in the url argument

No.	Property name	Description
1	`user`	Specifies the authorization identifier to be used to connect to the HADB server. For the naming rules for authorization identifiers, see the topic *Specifying names* in the manual *HADB SQL Reference*.

No.	Property name	Description
		You can use other methods of specifying the authorization identifier to be used to connect to the HADB server. For the specification priorities, see 7.3.3 Connection information priorities. If the `getConnection` method is executed with no authorization identifier specified, the method throws an `SQLException`.
2	`password`	Specifies a password for the authorization identifier that is to be used to connect to the HADB server.
3	`encodelang`	Specifies the conversion character set to be used for character encoding conversion processing when the `String` class is used to transfer data with the HADB server. Select a supported conversion character set from the list of *Supported encodings* in *Internationalization support* in the *JavaTM Platform, Standard Edition* JDK document. If this specification is omitted, the character encoding will be converted using the supported conversion character set indicated in Table 7-15: Names of the character sets supported for the HADB server's character encoding. Note that Java Virtual Machine's (JVM) default conversion character set is used to convert the following values: • Application identifiers (such as those specified with the `adb_clt_ap_name` user property) • Authorization identifiers or passwords (such as those specified with the `getConnection` method) Specify this property only if you want to use a character set other than the supported conversion character set shown in Table 7-15: Names of the character sets supported for the HADB server's character encoding. If you already use the supported conversion character set indicated in Table 7-15: Names of the character sets supported for the HADB server's character encoding for conversion, you do not need to specify this property.
4	`methodtrace`	Specifies whether JDBC interface method traces are to be obtained. `ON`: Obtain JDBC interface method traces. `OFF`: Do not obtain JDBC interface method traces. For details about the JDBC interface method traces, see 7.7.1 JDBC interface method traces. If any other value is specified, the JDBC driver throws an `SQLException`. If this specification is omitted, `OFF` is assumed. If no valid log writer is specified with the `setLogWriter` method, JDBC interface method traces are not obtained, even if `ON` is specified.
5	`tracenum`	Specifies the number of entries for a JDBC interface method trace, in the range from 10 to 1,000. The default value is `500`. This property value takes effect when both of the following conditions are satisfied: • A valid log writer is specified with the `setLogWriter` method. • `ON` is specified in `methodtrace`. Even if this property value is to take effect, the JDBC driver throws an `SQLException` if the specified value is not within the range of 10 to 1,000.
6	`sqlwarningkeep`	Specifies whether warning information returned from the HADB server is to be retained. `TRUE`: Retain warning information. `FALSE`: Do not retain warning information. If this specification is omitted, `TRUE` is assumed. If any other value is specified, the JDBC driver throws an `SQLException`. For details about the warning information retention level for the `Connection` object, see 8.9.1 Creating an SQLWarning object.

No.	Property name	Description
7	adb_clt_rpc_con_wait_time	Specifies the maximum amount of time to wait for HADB server connection processing to be completed. Functionally, this property is the same as the adb_clt_rpc_con_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_con_wait_time operand in the client definition.
8	adb_clt_rpc_sql_wait_time	Specifies the following wait times: • How long a HADB client waits for the HADB server to respond to a processing request. • How long to wait to secure processing real threads if a shortage occurs when multiple SELECT statements are executed concurrently in the same connection. Functionally, this property is the same as the adb_clt_rpc_sql_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_sql_wait_time operand in the client definition.
9	adb_clt_ap_name	Specifies the identification information (application identifier) for the application program that is to connect to the HADB server. Because application identifiers are converted to the Java Virtual Machine's (JVM) default conversion character set, we recommend that you use a name consisting of only single-byte alphanumeric characters that do not depend on the conversion character set. You can use other methods of specifying the application identifier. For the specification priorities, see 7.3.3 Connection information priorities. If the application program has connected to the HADB server without its application identifier having been specified anywhere, ******** will be set as the application identifier. Functionally, this property is the same as the adb_clt_ap_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_ap_name operand in the client definition.
10	adb_clt_group_name	Specifies the name of the client group to which the application belongs. Functionally, this property is the same as the adb_clt_group_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_group_name operand in the client definition.
11	adb_clt_fetch_size	Specifies the maximum number of rows that are to be sent as retrieval results from the HADB server to the HADB client by a single FETCH process. Functionally, this property is the same as the adb_clt_fetch_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_fetch_size operand in the client definition.
12	adb_dbbuff_wrktbl_clt_blk_num	Specifies the number of local work table buffer pages. Functionally, this property is the same as the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition.
13	adb_sql_exe_max_rthd_num	Specifies the maximum number of SQL processing real threads. Functionally, this property is the same as the adb_sql_exe_max_rthd_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_max_rthd_num operand in the client definition.
14	adb_sql_exe_hashgrp_area_size	Specifies the size (in kilobytes) of the hash grouping area. Functionally, this property is the same as the adb_sql_exe_hashgrp_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashgrp_area_size operand in the client definition.

No.	Property name	Description	
15	adb_sql_exe_hashtbl_area_size	Specifies the size (in megabytes) of the hash table area. Functionally, this property is the same as the adb_sql_exe_hashtbl_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashtbl_area_size operand in the client definition.	
16	adb_sql_exe_hashflt_area_size	Specifies the size (in megabytes) of the hash filter area. Functionally, this property is the same as the adb_sql_exe_hashflt_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashflt_area_size operand in the client definition.	
17	adb_sql_prep_delrsvd_use_srvdef	Specifies whether reserved words are to be unregistered if specified as such in the adb_sql_prep_delrsvd_words operand in the server definition. Functionally, this property is the same as the adb_sql_prep_delrsvd_use_srvdef operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_prep_delrsvd_use_srvdef operand in the client definition.	
18	adb_clt_trn_iso_lv	Specifies the transaction isolation level. Functionally, this property is the same as the adb_clt_trn_iso_lv operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_iso_lv operand in the client definition.	
19	adb_clt_trn_access_mode	Specifies the transaction access mode. Functionally, this property is the same as the adb_clt_trn_access_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_access_mode operand in the client definition.	
20	adb_clt_sql_text_out	Specifies whether SQL statements issued by the HADB client are to be output to the client message log files and the server message log files. Functionally, this property is the same as the adb_clt_sql_text_out operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_text_out operand in the client definition.	
21	adb_clt_sql_order_mode	Specifies the sort order for character string data in a SELECT statement in which the ORDER BY clause is specified. Functionally, this property is the same as the adb_clt_sql_order_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_order_mode operand in the client definition.	
22	adb_sql_prep_dec_div_rs_prior	Specify the minimum scaling value of the result of a division operation (arithmetic operation) specified in an SQL statement when the data type of the result is DECIMAL. Functionally, this property is the same as the adb_sql_prep_dec_div_rs_prior operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_prep_dec_div_rs_prior operand in the client definition.	
23	adb_jdbc_exc_trc_out_path	Specifies the absolute path of the directory to which exception trace logs are to be output.	For details about these properties and their permitted values, see (b) Setup for acquisition of the exception trace log (setting properties) in (1) Methods to be acquired and setup for log acquisition in 7.7.2 Exception trace log.
24	adb_jdbc_info_max	Specifies the maximum number of information items to be output to one file.	
25	adb_jdbc_cache_info_max	Specifies the maximum number of information items to be stored in memory.	

No.	Property name	Description	
26	adb_jdbc_trc_out_lv	Specifies the trace acquisition level.	

Note

You can use other methods of specifying these properties. For details about other specification methods and priorities, see 7.3.3 Connection information priorities.

Note

The property names that are specified in the connection URL were changed in HADB 03-00, as shown below. The previous property names are still supported, but if you have upgraded your HADB to version 03-00 or later, we recommend that you change the property names.

No.	Property name before change (property name used in HADB versions earlier than 03-00)	Property name after change (property name used in HADB version 03-00 or later)
1	apname	adb_clt_ap_name
2	extrcoutpath	adb_jdbc_exc_trc_out_path
3	extrcinfomax	adb_jdbc_info_max
4	extrccacheinfomax	adb_jdbc_cache_info_max
5	extrcoutlv	adb_jdbc_trc_out_lv

(b) Value to be specified in the user argument (specifying the authorization identifier)

The `user` argument specifies the authorization identifier that is used to connect to the HADB server.

For the naming rules for authorization identifiers, see the topic *Specifying names* in the manual *HADB SQL Reference*.

You can use other methods of specifying the authorization identifier used to connect to the HADB server. For the specification priorities, see 7.3.3 Connection information priorities.

If the `getConnection` method is executed with no authorization identifier specified, the method throws an `SQLException`.

If `null` is specified, the JDBC driver assumes that specification of an authorization identifier was omitted.

If a character string with a length of zero is specified, the JDBC driver throws an `SQLException`.

(c) Value to be specified in the password argument (specifying the password)

The `password` argument specifies a password for the authorization identifier that is to be used to connect to the HADB server.

If `null` or a character string with a length of zero is specified, the JDBC driver assumes that specification of a password was omitted.

(d) Values to be specified in the info argument (specifying the user properties)

The following table lists and describes the information that can be specified in the `info` argument (information that can be specified in user properties).

Table 7-3: Information that can be specified in the `info` argument (information that can be specified in user properties)

No.	Property name	Description
1	`user`	Specifies the authorization identifier to be used to connect to the HADB server. For the naming rules for authorization identifiers, see the topic *Specifying names* in the manual *HADB SQL Reference*. • You can use other methods of specifying the authorization identifier to be used to connect to the HADB server. For the specification priorities, see 7.3.3 Connection information priorities. • If the `getConnection` method is executed with no authorization identifier specified, the method throws an `SQLException`. • If `null` is specified, the JDBC driver assumes that specification of an authorization identifier was omitted. • If a character string with a length of zero is specified, the JDBC driver throws an `SQLException`.
2	`password`	Specifies a password for the authorization identifier being used to connect to the HADB server. If `null` is specified, or if a character string with a length of zero is specified, the JDBC driver assumes that no password is specified.
3	`encodelang`	Specifies the conversion character set to be used for character encoding conversion processing when the `String` class is used to transfer data with the HADB server. For details about this property and its permitted values, see Table 7-2: Properties that can be specified in the url argument. If the specified conversion character set name is not supported by Java Virtual Machine (JVM), the JDBC driver throws an `SQLException` when a connection is established with the HADB server. If this specification is omitted, the specification of `encodelang` for the connection URL is applied.
4	`methodtrace`	Specifies whether JDBC interface method traces are to be obtained. For details about this property and its permitted values, see Table 7-2: Properties that can be specified in the url argument. If this specification is omitted, the specification of `methodtrace` for the connection URL is applied.
5	`tracenum`	Specifies the number of JDBC interface method trace entries. For details about this property and its permitted values, see Table 7-2: Properties that can be specified in the url argument. If this specification is omitted, the specification of `tracenum` for the connection URL is applied.
6	`sqlwarningkeep`	Specifies whether warning information returned from the HADB server is to be retained. For details about this property and its permitted values, see Table 7-2: Properties that can be specified in the url argument. If this specification is omitted, the specification of `sqlwarningkeep` for the connection URL is applied.
7	`adb_clt_rpc_srv_host`	Specifies the host name of the HADB server at the connection destination. Functionally, this property is the same as the `adb_clt_rpc_srv_host` operand in the client definition. For details about this property and its permitted values, see the description of the `adb_clt_rpc_srv_host` operand in the client definition.
8	`adb_clt_rpc_srv_port`	Specifies the port number of the HADB server that is used for communication between the HADB client and the HADB server. Functionally, this property is the same as the `adb_clt_rpc_srv_port` operand in the client definition. For details about this property and its permitted values, see the description of the `adb_clt_rpc_srv_port` operand in the client definition.

No.	Property name	Description
9	adb_clt_rpc_con_wait_time	Specifies the maximum amount of time to wait for HADB server connection processing to be completed. Functionally, this property is the same as the adb_clt_rpc_con_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_con_wait_time operand in the client definition.
10	adb_clt_rpc_sql_wait_time	Specifies the following wait times: • How long a HADB client waits for the HADB server to respond to a processing request. • How long to wait to secure processing real threads if a shortage occurs when multiple SELECT statements are executed concurrently in the same connection. Functionally, this property is the same as the adb_clt_rpc_sql_wait_time operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_rpc_sql_wait_time operand in the client definition.
11	adb_clt_ap_name	Specifies the identification information (application identifier) for the application program that is to connect to the HADB server. Because application identifiers are converted to Java Virtual Machine's (JVM) default conversion character set, we recommend that you use a name consisting of only single-byte alphanumeric characters that do not depend on the conversion character set. You can use other methods of specifying the application identifier. For the specification priorities, see 7.3.3 Connection information priorities. If the application program has connected to the HADB server without its application identifier having been specified anywhere, ******** will be set as the application identifier. Functionally, this property is the same as the adb_clt_ap_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_ap_name operand in the client definition.
12	adb_clt_group_name	Specifies the name of the client group to which the application belongs. Functionally, this property is the same as the adb_clt_group_name operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_group_name operand in the client definition.
13	adb_clt_fetch_size	Specifies the maximum number of rows that are to be sent as retrieval results from the HADB server to the HADB client by a single FETCH process. Functionally, this property is the same as the adb_clt_fetch_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_fetch_size operand in the client definition.
14	adb_dbbuff_wrktbl_clt_blk_num	Specifies the number of local work table buffer pages. Functionally, this property is the same as the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_dbbuff_wrktbl_clt_blk_num operand in the client definition.
15	adb_sql_exe_max_rthd_num	Specifies the maximum number of SQL processing real threads. Functionally, this property is the same as the adb_sql_exe_max_rthd_num operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_max_rthd_num operand in the client definition.
16	adb_sql_exe_hashgrp_area_size	Specifies the size (in kilobytes) of the hash grouping area. Functionally, this property is the same as the adb_sql_exe_hashgrp_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashgrp_area_size operand in the client definition.

No.	Property name	Description	
17	adb_sql_exe_hashtbl_area_size	Specifies the size (in megabytes) of the hash table area. Functionally, this property is the same as the adb_sql_exe_hashtbl_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashtbl_area_size operand in the client definition.	
18	adb_sql_exe_hashflt_area_size	Specifies the size (in megabytes) of the hash filter area. Functionally, this property is the same as the adb_sql_exe_hashflt_area_size operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_exe_hashflt_area_size operand in the client definition.	
19	adb_sql_prep_delrsvd_use_srvdef	Specifies whether reserved words are to be unregistered if specified as such in the adb_sql_prep_delrsvd_words operand in the server definition. Functionally, this property is the same as the adb_sql_prep_delrsvd_use_srvdef operand in the client definition. For details about this property and its permitted values, see the description of the adb_sql_prep_delrsvd_use_srvdef operand in the client definition.	
20	adb_clt_trn_iso_lv	Specifies the transaction isolation level. Functionally, this property is the same as the adb_clt_trn_iso_lv operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_iso_lv operand in the client definition.	
21	adb_clt_trn_access_mode	Specifies the transaction access mode. Functionally, this property is the same as the adb_clt_trn_access_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_trn_access_mode operand in the client definition.	
22	adb_clt_sql_text_out	Specifies whether SQL statements issued by the HADB client are to be output to the client message log files and the server message log files. Functionally, this property is the same as the adb_clt_sql_text_out operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_text_out operand in the client definition.	
23	adb_clt_sql_order_mode	Specifies the sort order for character string data in a SELECT statement in which the ORDER BY clause is specified. Functionally, this property is the same as the adb_clt_sql_order_mode operand in the client definition. For details about this property and its permitted values, see the description of the adb_clt_sql_order_mode operand in the client definition.	
24	adb_sql_prep_dec_div_rs_prior	Specifies the minimum scaling value of the result of a division operation (arithmetic operation) specified in an SQL statement when the data type of the result is DECIMAL. Functionally, this property is the same as the adb_sql_prep_dec_div_rs_prior operand. For details about this property and its permitted values, see the description of the adb_sql_prep_dec_div_rs_prior operand in the client definition.	
25	adb_jdbc_exc_trc_out_path	Specifies the absolute path of the directory to which exception trace logs are to be output.	For details about these properties and their permitted values, see (b) Setup for acquisition of the exception trace log (setting properties) in (1) Methods to be acquired and setup for log acquisition in 7.7.2 Exception trace log.
26	adb_jdbc_info_max	Specifies the maximum number of information items to be output to one file.	
27	adb_jdbc_cache_info_max	Specifies the maximum number of information items to be stored in memory.	

No.	Property name	Description	
28	adb_jdbc_trc_out_lv	Specifies the trace acquisition level.	

Note

- You can use other methods of specifying these properties. For details about other specification methods and priorities, see 7.3.3 Connection information priorities.
- If null is specified for any property, the JDBC driver assumes that specification of that property was omitted.

Note

The property names of user properties were changed in HADB 03-00, as shown below. The previous property names are still supported, but if you have upgraded your HADB to version 03-00 or later, we recommend that you change the property names.

No.	Property name before change (property name used in HADB versions earlier than 03-00)	Property name after change (property name used in HADB version 03-00 or later)
1	apname	adb_clt_ap_name
2	host	adb_clt_rpc_srv_host
3	port	adb_clt_rpc_srv_port

7.3.2 Using the getConnection method of the DataSource class to connect to the HADB server

Database connection (connection to the HADB server) using DataSource and JNDI can now be used by the JDBC 2.0 Optional Package.

Although it is not essential to use JNDI, using it offers the benefit that you need to specify the connection information only once. Because the DataSource class interface definition and JNDI are not included in JDK as standard features, you must obtain them from the JavaSoft web site when you develop an application program.

The following explains the procedure for using DataSource and JNDI to connect to the HADB server.

To connect to the HADB server:

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 HADB server.

If you are not using JNDI, steps 3 and 4 are not necessary.

If you are using JNDI, steps 1 through 3 need to be executed only once. Thereafter, you can connect to the HADB server 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 `AdbDataSource`.

Shown below is an example of generating the `DataSource` class object:

```
com.hitachi.hadb.jdbc.AdbDataSource  ds = null ;
ds = new com.hitachi.hadb.jdbc.AdbDataSource() ;
```

(2) Setting up the connection information

Call the method for setting up connection information for the `DataSource` object, and set up the connection information. There is also a method for acquiring the connection information, which you can use to check the current connection information. For details about the connection information setup and acquisition methods, see 10.5 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
com.hitachi.hadb.jdbc.AdbDataSource  ds;
ds = new com.hitachi.hadb.jdbc.AdbDataSource();

// 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 HADB server
// 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 JDBC 2.0 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 an acquisition example for the `DataSource` object (this an example is for Windows). This acquisition 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 HADB server

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 specified for the `DataSource` object. The JDBC driver throws an `SQLException` in the following cases:

- Required connection information is not specified in the `DataSource` object.
- Specified connection information is invalid.
- Connection with the HADB server fails.

After you have obtained the DataSource object from JNDI, you can 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
com.hitachi.hadb.jdbc.AdbDataSource  adb_ds;

// Get DataSource object from JNDI
                    :

// Cast DataSource object to DataSource class type provided
// by JDBC driver
adb_ds = (com.hitachi.hadb.jdbc.AdbDataSource)ds;

// Set up connection information again
         :
```

7.3.3 Connection information priorities

(1) Connection information needed when a connection to the HADB server is established

When a connection to the HADB server is established, the following connection information is required:

- HADB server's host name
- HADB server's port number
- Authorization identifier and password used to connect to the HADB server
- Application identifier
- Other items that can be specified in properties

The JDBC driver enables you to use any of several methods of specifying this connection information. For example, the HADB server's host name can be specified with the adb_clt_rpc_srv_host system property as well as in host for the connection URL.

The following table shows the priorities when the connection information is specified by multiple methods.

Table 7-4: Priorities for connection information

Connection information	Specification method	Priority	
		DM	DS
HADB server's host name	Value of the adb_clt_rpc_srv_host system property	1	1
	adb_clt_rpc_srv_host property value specified in the info argument of the getConnection method of the DriverManager class	2	--
	Value of host specified in the url argument of the getConnection method of the DriverManager class	3	--
	Host name specified by the setHostName method of an interface for setting/getting connection information	--	2
HADB server's port number	Value of the adb_clt_rpc_srv_port system property	1	1

Connection information	Specification method	Priority	
		DM	DS
	adb_clt_rpc_srv_port property value specified in the info argument of the getConnection method of the DriverManager class	2	--
	Value of port specified in the url argument of the getConnection method of the DriverManager class	3	--
	Port number specified by the setPort method of an interface for setting/getting connection information	--	2
Authorization identifier and password used for establishing a connection	One of the following sets of values: • Values of the user and password arguments of the getConnection method of the DriverManager class • user and password property values specified in the info argument of the getConnection method of the DriverManager class	1	--
	Values of user and password specified in the url argument of the getConnection method of the DriverManager class	2	--
	One of the following sets of values: • Values of the username and password arguments of the getConnection method of the DataSource interface • Values of the user and password arguments of the getPooledConnection method of the ConnectionPoolDataSource interface	--	1
	• Authorization identifier specified by the setUser method of an interface for setting/getting connection information • Password specified by the setPassword method of an interface for setting/getting connection information	--	2
Application identifier	Value of the adb_clt_ap_name system property	1	1
	adb_clt_ap_name property value specified in the info argument of the getConnection method of the DriverManager class	2	--
	Value of adb_clt_ap_name specified in the url argument of the getConnection method of the DriverManager class	3	--
	Application identifier specified by the setApName method of the connection information setup and acquisition interface	--	2
Timeout time for HADB server connection processing	Value of adb_clt_rpc_con_wait_time system property	1	1
	Value of adb_clt_rpc_con_wait_time specified in the info argument of the getConnection method of the DriverManager class	2	--
	Value of adb_clt_rpc_con_wait_time specified in the url argument of the getConnection method of the DriverManager class	3	--
	Value specified in the setLoginTimeout method of the DriverManager class	4	--
	One of the following values: • Value specified in the setLoginTimeout method of the DataSource interface • Value specified in the setLoginTimeout method of the ConnectionPoolDataSource interface	--	2
Other items that can be specified in properties:	Property values specified in system properties	1	1
	Property values specified in the info argument of the getConnection method of the DriverManager class	2	--

Connection information	Specification method	Priority	
		DM	DS
• adb_clt_rpc_sql_wait_time • adb_clt_group_name • adb_clt_fetch_size • adb_clt_sql_text_out • adb_clt_trn_iso_lv • adb_clt_sql_order_mode • adb_sql_prep_dec_div_rs_prior • adb_clt_trn_access_mode • adb_dbbuff_wrktbl_clt_blk_num • adb_sql_prep_delrsvd_use_srvdef • adb_sql_exe_max_rthd_num • adb_sql_exe_hashgrp_area_size • adb_sql_exe_hashtbl_area_size • adb_sql_exe_hashflt_area_size • adb_jdbc_exc_trc_out_path • adb_jdbc_info_max • adb_jdbc_cache_info_max • adb_jdbc_trc_out_lv	Property values specified in the `url` argument of the `getConnection` method of the `DriverManager` class	3	--

Legend:

DM: For a connection that uses the `DriverManager` class

DS: For a connection that uses the `DataSource` class

--: Connection information cannot be specified

Note

The smaller the priority number, the higher the priority. Priority number 1 is higher in priority than priority number 2.

(2) List of connection information items that can be specified in individual properties

You can use system properties, user properties, or URL connection properties to specify the connection information needed to establish a connection to the HADB server. The following table lists the connection information that can be specified in individual properties.

Table 7-5: List of connection information items that can be specified in individual properties

No.	Classification	Property name	Whether specifiable in		
			System properties	User properties	URL connection properties
1	Properties with the same name and function as in the client definition[#1]	adb_clt_rpc_srv_host	Y	Y	A[#2]
2		adb_clt_rpc_srv_port	Y	Y	A[#3]
3		adb_clt_rpc_con_wait_time	Y	Y	Y
4		adb_clt_rpc_sql_wait_time	Y	Y	Y
5		adb_clt_ap_name	Y	Y	Y
6		adb_clt_group_name	Y	Y	Y
7		adb_clt_fetch_size	Y	Y	Y
8		adb_dbbuff_wrktbl_clt_blk_num	Y	Y	Y
9		adb_sql_exe_max_rthd_num	Y	Y	Y
10		adb_sql_exe_hashgrp_area_size	Y	Y	Y
11		adb_sql_exe_hashtbl_area_size	Y	Y	Y
12		adb_sql_exe_hashflt_area_size	Y	Y	Y
13		adb_sql_prep_delrsvd_use_srvdef	Y	Y	Y
14		adb_clt_trn_iso_lv	Y	Y	Y
15		adb_clt_trn_access_mode	Y	Y	Y
16		adb_clt_sql_text_out	Y	Y	Y
17		adb_clt_sql_order_mode	Y	Y	Y
18		adb_sql_prep_dec_div_rs_prior	Y	Y	Y
19	Properties related to exception trace logs	adb_jdbc_exc_trc_out_path	Y	Y	Y
20		adb_jdbc_info_max	Y	Y	Y
21		adb_jdbc_cache_info_max	Y	Y	Y
22		adb_jdbc_trc_out_lv	Y	Y	Y
23	Other properties	user	N	Y	Y
24		password	N	Y	Y
25		encodelang	N	Y	Y
26		methodtrace	N	Y	Y
27		tracenum	N	Y	Y
28		sqlwarningkeep	N	Y	Y

Legend

Y: Property can be specified.

A: Property cannot be specified, but an alternate specification is available.

N: Property cannot be specified.

#1

Each of these properties has the same name and function as an operand in the client definition.

#2

The `adb_clt_rpc_srv_host` URL connection property cannot be used to specify the host name of the HADB server at the connection destination. You must use the `host` URL connection property to specify this host name.

#3

The `adb_clt_rpc_srv_port` URL connection property cannot be used to specify the port number of the HADB server. You must use the `port` URL connection property to specify this port number.

Important

The client definition operands are not applicable when the JDBC driver is used. The specified system properties, user properties, or URL connection properties are applied instead.

For details about system properties, see 3.1.6 Setting system properties.

For details about the user properties, see (d) Values to be specified in the info argument (specifying the user properties) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

For details about the URL connection properties, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

7.4 Retrieving data (executing the SELECT statement)

This section explains how to use the JDBC driver to retrieve data.

7.4.1 How to retrieve data

The procedure for using the SELECT statement to retrieve data is as follows:

- Generate a Statement object
- Execute the SELECT statement
- Get the retrieval results

(1) Generating a Statement object

Generate a Statement object, and then send the SELECT statement to the HADB server.

If a connection to the HADB server has already been established, you can use the createStatement method of the Connection object to generate a Statement object.

The following example generates a Statement object:

```
// Connect to the HADB server
Connection con = DriverManager.getConnection(url,info);

// Generate a Statement object
Statement stmt = con.createStatement();
```

(2) Executing the SELECT statement

Execute the SELECT statement by specifying it in the argument of the executeQuery method. The following example executes a SELECT statement:

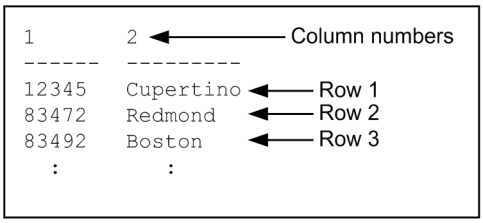
```
Statement stmt = con.createStatement();

// Execute the SELECT statement and obtain the ResultSet object
ResultSet rs = stmt.executeQuery("SELECT \"CODE\", \"STATE\" FROM \"SAMPLE\"");
```

When a SELECT statement is executed, the retrieval results are stored in a ResultSet object.

(3) Getting the retrieval results

The retrieval results are stored in a ResultSet object in a tabular format that consists of column numbers and values corresponding to the retrieval results. The following figure shows an example of the format of a ResultSet object.

Figure 7-2: Example of the format of a ResultSet object



The `ResultSet` object contains a cursor that points to the current row. You obtain retrieval results from the `ResultSet` object by using the `next` method to move the cursor and then a `get`*XXX* method to obtain the data on the current row.

When a `ResultSet` object is generated, the cursor is positioned immediately before the first row. When the first `next` method is called, the cursor moves to the first row. Each time the `next` method is called thereafter, the cursor moves down by one row.

The following example obtains retrieval results data:

```
ResultSet rs = stmt.executeQuery("SELECT \"CODE\", \"STATE\" FROM \"SAMPLE\"");

// Repeat until there is no more result row
while(rs.next())
{
  // Get data from column 1
  int i = rs.getInt(1);
  // Get data from column 2
  String s = rs.getString(2);
  // Output the result data
  System.out.println("Retrieval results: " + i + ", " + s);
}
```

(4) Note about executing multiple SELECT statements concurrently in the same connection

When you execute multiple `SELECT` statements concurrently in the same connection, HADB might be unable to supply enough processing real threads to execute the `SELECT` statements. In this situation, processing to allocate processing real threads is repeated until the required number is allocated. You need to use one of the following approaches to make sure that allocation processing will not continue indefinitely. We recommend that you use the first approach if possible.

Approaches to resolving the issue

1. Amend the application.

   If you are able to amend the application, amend it so that unnecessary `ResultSet` objects are closed. If you are unable to amend the application, look into whether you can change the server definition as follows.

2. Change the server definition.

   If you are able to change the server definition, change it so that it meets the following formula:

   $$A \geq B \times C \times D$$

   A: Number of processing threads (`adb_sys_rthd_num` operand's value)

   B: Maximum number of SQL processing real threads (`adb_sql_exe_max_rthd_num` operand's value)

   C: Number of `SELECT` statements that can be executed concurrently in one connection (that are placed in cursor open status)

D: Number of connections that can execute SQL statements concurrently

3. Set a wait time.

   If the preceding two approaches are unavailable to you, specify a wait time in the following method or property that governs how long HADB waits to allocate the required processing real threads:

   - `setQueryTimeout` method of the `Statement` interface
   - `adb_clt_rpc_sql_wait_time` in the system properties, user properties, or connection URL properties

7.4.2 How to use dynamic parameters

The procedure for using dynamic parameters to retrieve data is as follows:

- Generate a `PreparedStatement` object
- Execute the `SELECT` statement
- Get the retrieval results

(1) Generating a PreparedStatement object

Generate a `PreparedStatement` object, and then send the `SELECT` statement with the dynamic parameters specified to the HADB server.

If a connection to the HADB server has already been established, you can use the `prepareStatement` method of the `Connection` object to generate a `PreparedStatement` object.

The following example generates a `PreparedStatement` object:

```
 // Connect to the HADB server
 Connection con = DriverManager.getConnection(url, info);

 // Generate a PreparedStatement object
 PreparedStatement pstmt =
 con.prepareStatement("SELECT * FROM \"SAMPLE\" WHERE \"CODE\" = ? AND \"STATE\" = ?"
);
```

If the `SELECT` statement is specified in the argument of the `prepareStatement` method, the `SELECT` statement is preprocessed and a `PreparedStatement` object is generated.

To specify dynamic parameters in an SQL statement, values must have been set for all the specified dynamic parameters before the SQL statement is executed. You use a `set`*XXX* method to set a value in a dynamic parameter.

(2) Executing the SELECT statement

Use the `executeQuery` method with no argument specified to execute the `SELECT` statement. The following example executes a `SELECT` statement that uses dynamic parameters:

```
 PreparedStatement pstmt =
 con.prepareStatement("SELECT * FROM \"SAMPLE\" WHERE \"CODE\" = ? AND \"STATE\" = ?"
);

 // Set a value in the first dynamic parameter
 pstmt.setInt(1, 12345);
 // Set a value in the second dynamic
```

```
pstmt.setString(2, "Boston");

// Execute a preprocessed SQL statement to obtain the ResultSet object
ResultSet rs = pstmt.executeQuery();
```

When the `SELECT` statement is executed, the retrieval results are stored in a `ResultSet` object.

(3) Getting the retrieval results

For details about how to get the retrieval results, see (3) Getting the retrieval results in 7.4.1 How to retrieve data.

7.5 Adding, updating, or deleting data (executing the INSERT, UPDATE, or DELETE statement)

You use the `executeUpdate` method or the `executeLargeUpdate` method of the `Statement` object (the `PreparedStatement` object if a dynamic parameter is used) to add, update, or delete data by executing a data manipulation SQL statement, such as the `INSERT`, `UPDATE`, or `DELETE` statement.

The following example updates and deletes data:

```
 Connection con = DriverManager.getConnection(url, info);
 Statement stmt = con.createStatement();

 // Update the data that satisfies the condition
 stmt.executeUpdate("UPDATE \"SAMPLE\" SET \"CODE\"=98765 WHERE \"STATE\" = 'Redmond'
");

 // Delete all rows
 stmt.executeUpdate("DELETE FROM \"SAMPLE\" ");
```

■ Effects of update operations on a retrieval using a cursor

If an update operation using a cursor is performed during a retrieval, the results of the update operation might be applied to the retrieval results, depending on the timing. Do the following to prevent the results of such update operations from being applied to retrieval results:

- Close the cursor before adding or updating rows.
- Specify data and search conditions in such a manner that rows to be added or updated will not be included in the retrieval results.

7.6 Data processing

This section explains how data is processed between the JDBC driver and the HADB server.

7.6.1 Mapping data types

This subsection explains mapping between HADB's data types and JDBC's SQL data types.

(1) Correspondence between HADB's data types and JDBC's SQL data types

There is not an exact one-to-one match between the HADB data types and the JDBC SQL data types. For this reason, the JDBC driver performs mapping (conversion) between HADB's data types and JDBC's SQL data types. If an unmappable data type is used for data access, the JDBC driver throws an `SQLException`.

The data types are mapped with `get`*XXX* and `set`*XXX* methods in the `ResultSet` or `PreparedStatement` class. For details about the mapping rules employed by the `get`*XXX* and `set`*XXX* methods, see the documentation for the JDBC 1.0 standard or the JDBC 2.0 basic standard.

The following table shows the correspondences between HADB's data types and JDBC's SQL data types.

Table 7-6: Correspondences between HADB's data types and JDBC's SQL data types

HADB's data type	JDBC's SQL data type
INTEGER	BIGINT
SMALLINT	INTEGER, SMALLINT[#1]
DECIMAL	DECIMAL (, NUMERIC)[#2]
DOUBLE PRECISION	DOUBLE
CHAR	CHAR
VARCHAR	VARCHAR (, LONGVARCHAR)[#2]
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
BINARY	BINARY (, VARBINARY, LONGVARBINARY)[#2]
VARBINARY	VARBINARY (, BINARY, LONGVARBINARY)[#2]
ROW	BINARY (, VARBINARY, LONGVARBINARY)[#2]
BOOLEAN[#3]	BOOLEAN

#1

This column is found only in a `ResultSet` created from `DatabaseMetaData`. If the column's data type is defined as `short`, the JDBC driver returns the value corresponding to `SMALLINT` for the metadata that can be obtained by `ResultSetMetaData`. For data other than metadata, the JDBC driver returns the value corresponding to `INTEGER`.

#2

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. Otherwise, they are not supported for mapping from HADB's data types to JDBC's SQL data types.

#3

This refers to a `BOOLEAN` column in a `ResultSet` object that is generated by the `getTypeInfo` method of `DatabaseMetaData`.

(2) Mapping during retrieval data acquisition

The table below shows the mapping between the `get`*XXX* methods in 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 7-7: Mapping between getXXX methods and JDBC's SQL data types

Method name	JDBC's SQL data type									
	BIG	INT	DCML	DBL	CHAR	VCHR	DATE	TIME	TSTMP	BIN, VARBIN
getByte	Y	Y	Y	Y	Y#	Y#	N	N	N	N
getShort	Y	Y	Y	Y	Y#	Y#	N	N	N	N
getInt	Y	Rec	Y	Y	Y#	Y#	N	N	N	N
getLong	Rec	Y	Y	Y	Y#	Y#	N	N	N	N
getFloat	Y	Y	Y	Y	Y#	Y#	N	N	N	N
getDouble	Y	Y	Y	Rec	Y#	Y#	N	N	N	N
getBigDecimal	Y	Y	Rec	Y	Y#	Y#	N	N	N	N
getBoolean	Y	Y	Y	Y	Y	Y	N	N	N	N
getString	Y	Y	Y	Y	Rec	Rec	Y	Y	Y	Y
getBytes	N	N	N	N	N	N	N	N	N	Rec
getDate	N	N	N	N	Y#	Y#	Rec	Y	Y	N
getTime	N	N	N	N	Y#	Y#	N	Rec	Y	N
getTimestamp	N	N	N	N	Y#	Y#	Y	Y	Rec	N
getAsciiStream	N	N	N	N	Y	Y	N	N	N	Y
getObject	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
getCharacterStream	N	N	N	N	Y	Y	N	N	N	Y
getBinaryStream	N	N	N	N	N	N	N	N	N	Y

Legend:

BIG: `BIGINT`

INT: `INTEGER`

DBL: `DOUBLE`

DCML: `DECIMAL`

VCHR: `VARCHAR`

TSTMP: `TIMESTAMP`

BIN: `BINARY`

VARBIN: `VARBINARY`

Rec: Mapping is recommended.

Y: Can be mapped. Note, however, that data loss or a conversion error might occur, depending on the format of the conversion-source data.

N: Cannot be mapped.

#

During conversion for this method, the JDBC driver removes any spaces preceding or following the character string data retrieved from the database. After removing the spaces, the JDBC driver converts the data to the Java data type returned by the `get`*XXX* method.

The following notes apply to converting data to a Java data type:

- If 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.
- If overflow occurs after character string data is converted to a Java data type, the JDBC driver throws an `SQLException`.

(3) Mapping when a dynamic parameter is specified

The following table lists the `set`*XXX* methods of the `PreparedStatement` class and JDBC's SQL data types that can be mapped to the methods. For a JDBC's SQL data type that cannot be used, an `SQLException` is thrown.

If a `set`*XXX* method is called for one of JDBC's unmappable SQL data types, the JDBC driver throws an `SQLException`.

Table 7-8: Mapping between setXXX methods and JDBC's SQL data types

Method name	JDBC's SQL data type									
	BIGINT	INTEGER	DECIMAL[#1]	DOUBLE	CHAR	VARCHAR	DATE	TIME	TIMESTAMP	BINARY, VARBINARY
`setByte`	Y	Y	Y	Y	Y	Y	N	N	N	N
`setShort`	Y	Y	Y	Y	Y	Y	N	N	N	N
`setInt`	Y	Rec	Y	Y	Y	Y	N	N	N	N
`setLong`	Rec	Y	Y	Y	Y	Y	N	N	N	N
`setFloat`	Y	Y	Y	Y	Y	Y	N	N	N	N
`setDouble`	Y	Y	Y	Rec	Y	Y	N	N	N	N
`setBigDecimal`	Y	Y	Rec	Y	Y	Y	N	N	N	N
`setBoolean`	Y	Y	Y	Y	Y	Y	N	N	N	N

Method name	JDBC's SQL data type									
	BIGINT	INTEGER	DECIMAL[#1]	DOUBLE	CHAR	VARCHAR	DATE	TIME	TIMESTAMP	BINARY, VARBINARY
setString	Y	Y	Y	Y	Rec	Rec	Y	Y	Y	Y
setBytes	N	N	N	N	N	N	N	N	N	Rec
setDate	N	N	N	N	Y	Y	Rec	Y	Y	N
setTime	N	N	N	N	Y	Y	N	Rec	Y	N
setTimestamp[#2]	N	N	N	N	Y	Y	Y	Y	Rec	N
setAsciiStream	N	N	N	N	Y	Y	N	N	N	Y
setObject[#3]	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
setCharacterStream	N	N	N	N	Y[#4]	Y[#4]	N	N	N	Y
setBinaryStream	N	N	N	N	N	N	N	N	N	Y

Legend:

Rec: Mapping is recommended.

Y: Can be mapped. Note, however, that data loss or a conversion error might occur, depending on the format of the conversion-source data.

N: Cannot be mapped.

#1

If a `set`*XXX* method specifies a value for a dynamic parameter of HADB's `DECIMAL` data type, and the dynamic parameter and the value have different precisions or scalings, the JDBC driver performs one of the following operations, as applicable:

- When the value has a larger precision than the dynamic parameter: Throws an `SQLException`.
- When the value has a smaller precision than the dynamic parameter: Increases the precision.
- When the value has a larger scaling than the dynamic parameter: Truncates the value according to the actual scaling.
- When the value has a smaller scaling than the dynamic parameter: Adds zeros to increase the scaling.

#2

If a `set`*XXX* method specifies a value for a dynamic parameter of HADB's `TIME` or `TIMESTAMP` data type, and the dynamic 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 dynamic parameter: Truncates the fractional seconds part of the value.
- If the value has a smaller fractional seconds precision than the dynamic parameter: Expands the fractional seconds part of the value.

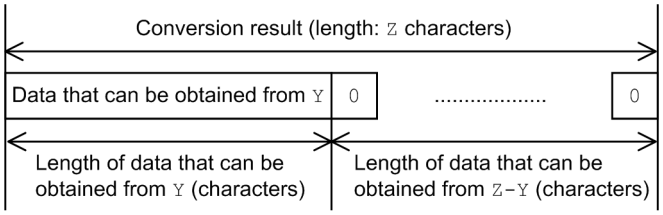
#3

Objects of the `InputStream` class and the `Reader` class (including subclasses) cannot be specified in the `setObject` method.

#4

If the length of the data that can be retrieved from a `java.io.Reader` object is shorter than the length specified in the arguments, the JDBC driver adds zeros as shown below until the length specified in the arguments is reached:

```
int X,Z;
java.io.Reader Y;
setCharacterStream(X,Y,Z)
```



7.6.2 Data conversion process

This subsection explains the data conversion process when `set`*XXX* and `get`*XXX* methods are executed.

(1) Data conversion process during execution of a setXXX method (for DATE, TIME or TIMESTAMP type)

This subsection explains the conversion process when data of HADB's `DATE`, `TIME` 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 HADB's `DATE`, `TIME` or `TIMESTAMP` data type, data conversion takes place according to the rules described in the following table.

Table 7-9: Conversion process during execution of the setXXX methods

Method executed	Conversion process for DATE type	Conversion process for TIME type	Conversion process for TIMESTAMP type
`setDate(Date Obj)` #1	Stores the application program settings in the database without performing conversion.	`00:00:00` is stored in the database.	Stores in the database the data with `00:00:00` added after the application program setting for *YYYY-MM-DD*.
`setTime(Time Obj)` #2	Throws an `SQLException`.	The data for the application program setting for *hh*:*mm*:*ss*.*fff* is stored in the database.	Stores in the database the data with `1970-01-01` added before the application program setting for *hh*:*mm*:*ss*.*fff*.
`setTimestamp(Timestamp Obj)` #3	Stores in the database the data formed when *YYYY-MM-DD* is removed from the application program setting.	The data obtained by extracting *hh*:*mm*:*ss*.*fffffffff* from the application program setting is stored in the database.	The data for the application program setting for *YYYY-MM-DD*Δ*hh*:*mm*:*ss*.*fffffffff* is stored in the database.#4

Method executed	Conversion process for DATE type	Conversion process for TIME type	Conversion process for TIMESTAMP type
`setString` (character string in *YYYY-MM-DD* format)	Converts the specified date with `java.sql.Date.valueOf()` and stores the result in the database.[#5]	`00:00:00` is stored in the database.	Throws an `SQLException`.
`setString` (character string in *hh*:*mm*:*ss*`[`.*f*...`]` format)	Throws an `SQLException`.	The data for the application program setting for *hh*:*mm*:*ss*`[`.*f*...`]` is stored in the database.	Throws an `SQLException`.
`setString` (character string in *YYYY-MM-DD*Δ*hh*:*mm*:*ss*`[`.*f*...`]` format) [#4]	Throws an `SQLException`.	The data for the application program setting for *hh*:*mm*:*ss*`[`.*f*...`]` is stored in the database.	The data for the application program setting for *YYYY-MM-DD*Δ*hh*:*mm*:*ss*`[`.*f*...`]` is stored in the database.[#5]

Note

If a non-existent date or time is specified, the Java Virtual Machine (JVM) returns the specified value.

#1

`Date Obj` is an object that has the value of the `java.sql.Date` object with the format *year-month-date*.

#2

`Time Obj` is an object that has the value of the `java.sql.Time` object with the format *hour*:*minute*:*second*:*millisecond*.

#3

`Timestamp Obj` is an object that has the value of the `java.sql.Timestamp` object with the format *year-month-date hour*:*minute*:*second*:*nanosecond*.

#4

Δ represents a space.

#5

The result when a non-existent date or time is specified depends on `java.sql.Date.valueOf()`, `java.sql.Time.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` (the calendar switches from the Julian to the Gregorian).

(2) Data conversion process during execution of a getXXX method (for DATE, TIME, TIMESTAMP, or character string type)

This subsection explains the conversion process when data of HADB's `DATE`, `TIME`, `TIMESTAMP`, or character string (`CHAR` or `VARCHAR`) 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 HADB's `DATE`, `TIME`, `TIMESTAMP`, or character string data type, data conversion takes place according to the rules described in the following table.

Table 7-10: Conversion process during execution of the getXXX methods

Method executed	Conversion process for DATE type	Conversion process for TIME type	Conversion process for TIMESTAMP type	Conversion process for character string type
`getDate()`#1	Gets the value stored in the database and sets it as the `java.sql.Date` object without performing conversion.#2	Gets the value as the `java.sql.Date` object that has the data `1970-01-01`.#2	Removes the *year*−*month*−*date* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Date` object.#2	Gets only a *YYYY*−*MM*−*DD* character string representation as the `java.sql.Date` object. For other representations, the method throws an `SQLException`.
`getTime()`#1	Throws an `SQLException`.	Removes the *hour*:*minute*:*second*.*millisecond* data from the `TIME` data retrieved from the database and sets the result as the `java.sql.Time` object.#2	Removes the *hour*:*minute*:*second*:*millisecond* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Time` object.#2	Gets only an *hh*:*mm*:*ss*`[`.*f*...`]` character string representation as the `java.sql.Time` object. For other representations, the method throws an `SQLException`.
`getTimestamp()`#1	Appends `00:00:00.000000000` to the `DATE` data retrieved from the database and sets the result as the `java.sql.Timestamp` object.	Adds `1970-01-01` before the `TIME` data retrieved from the database and sets the result as the `java.sql.Timestamp` object.	Removes the *year*−*month*−*date*Δ*hour:minute:second*.*nanosecond* data from the `TIMESTAMP` data retrieved from the database and sets the result as the `java.sql.Timestamp` object.	Gets only a *YYYY*−*MM*−*DD*Δ*hh:mm:ss*`[`.*f*...`]` character string representation of the `TIMESTAMP` type as the `java.sql.Timestamp` object (Δ is a space). For other expressions, the method throws an `SQLException`.

#1

The date and time stored in the database might 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).

#2

The setting for an unspecified date item (*year*−*month*−*day*) is `1970-01-01`, and the setting for an unspecified time item (*hour*:*minute*:*second*) is `00:00:00`.

7.6.3 Overflow handling

This subsection discusses the possibility of overflow when a `set`*XXX* or `get`*XXX* method is executed.

(1) Possibility of overflow when a setXXX method (other than the setObject method) is executed

The following table shows whether overflow might occur when a `set`*XXX* method (other than the `setObject` method) is executed.

Table 7-11: Possibility of overflow when a setXXX method (other than the setObject method) is executed

Method executed	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE#	TIME#	TSTMP#	ROW	BIN, VARBIN
setByte	N	N	N	Y	N	--	--	--	--	--
setShort	N	N	N	Y	N	--	--	--	--	--
setInt	N	N	N	Y	N	--	--	--	--	--
setLong	Y	N	N	Y	N	--	--	--	--	--
setFloat	Y	Y	N	Y	N	--	--	--	--	--
setDouble	Y	Y	N	Y	N	--	--	--	--	--
setBigDecimal	Y	Y	N	Y	N	--	--	--	--	--
setBoolean	N	N	N	N	N	--	--	--	--	--
setString	Y	Y	N	Y	N	Y	N	Y	--	N
setBytes	--	--	--	--	--	--	--	--	N	N
setDate	--	--	--	--	N	Y	N	Y	--	--
setTime	--	--	--	--	N	--	N	Y	--	--
setTimestamp	--	--	--	--	N	Y	N	Y	--	--
setAsciiStream	--	--	--	--	N	--	--	--	--	N
setBinaryStream	--	--	--	--	N	--	--	--	--	N
setCharacterStream	--	--	--	--	N	--	--	--	--	N

Legend:

SML: `SMALLINT`

INT: `INTEGER`

DBL PREC: `DOUBLE PRECISION`

DCML: `DECIMAL`

VCHR: `VARCHAR`

TSTMP: `TIMESTAMP`

BIN: `BINARY`

VARBIN: `VARBINARY`

N: Overflow does not occur regardless of the value.

Y: Overflow might occur depending on the value.

--: Combination that 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 HADB'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) Possibility of overflow when the setObject method is executed

The following table shows whether overflow might occur when the `setObject` method is executed.

Table 7-12: Possibility of overflow when the setObject method is executed

Data type of object specified by the setObject method	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE#	TIME#	TSTMP#	ROW	BIN, VARBIN
Byte	N	N	N	Y	N	--	--	--	--	--
Short	N	N	N	Y	N	--	--	--	--	--
Integer	N	N	N	Y	N	--	--	--	--	--
Long	Y	N	N	Y	N	--	--	--	--	--
Decimal	Y	Y	N	Y	N	--	--	--	--	--
Float	Y	Y	N	Y	N	--	--	--	--	--
Double	Y	Y	N	Y	N	--	--	--	--	--
Boolean	N	N	N	N	N	--	--	--	--	--
String	Y	Y	N	Y	N	Y	N	Y	--	--
Date	--	--	--	--	N	Y	N	Y	--	--
Time	--	--	--	--	N	--	N	--	--	--
Timestamp	--	--	--	--	N	Y	N	Y	--	--
byte[]	--	--	--	--	N	--	--	--	N	N

Legend:

SML: `SMALLINT`

INT: `INTEGER`

DBL PREC: `DOUBLE PRECISION`

DCML: `DECIMAL`

VCHR: `VARCHAR`

TSTMP: `TIMESTAMP`

BIN: BINARY

VARBIN: VARBINARY

N: Overflow does not occur regardless of the value.

Y: Overflow might occur depending on the value.

--: Combination that 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 HADB'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) Possibility of overflow when a getXXX method (other than the getObject method) is executed

The following table shows whether overflow might occur when a get*XXX* method (other than the getObject method) is executed.

Table 7-13: Possibility of overflow when a getXXX method (other than the getObject method) is executed

Method executed	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE	TIME	TSTMP	ROW	BIN, VARBIN
getByte	Y	Y	Y	Y	Y	--	--	--	--	--
getShort	Y	Y	Y	Y	Y	--	--	--	--	--
getInt	N	Y	Y	Y	Y	--	--	--	--	--
getLong	N	N	Y	Y	Y	--	--	--	--	--
getFloat	N	N	N	N	N	--	--	--	--	--
getDouble	N	N	N	N	N	--	--	--	--	--
getBigDecimal	N	N	N	N	N	--	--	--	--	--
getBoolean	N	N	N	N	N	--	--	--	--	--
getString	N	N	N	N	N	N	N	N	--	N
getBytes	--	--	--	--	--	--	--	--	N	N
getDate	--	--	--	--	N	N	N	N	--	--
getTime	--	--	--	--	N	--	N	N	--	--

Method executed	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE	TIME	TSTMP	ROW	BIN, VARBIN
getTimestamp	--	--	--	--	N	N	N	N	--	--
getAsciiStream	--	--	--	--	N	--	--	--	--	N
getBinaryStream	--	--	--	--	N	--	--	--	--	N
getCharacterStream	--	--	--	--	N	--	--	--	--	N

Legend:

SML: SMALLINT

INT: INTEGER

DBL PREC: DOUBLE PRECISION

DCML: DECIMAL

VCHR: VARCHAR

TSTMP: TIMESTAMP

BIN: BINARY

VARBIN: VARBINARY

N: Overflow does not occur regardless of the value.

Y: Overflow might occur depending on the value.

--: Combination that is not allowed.

(4) Possibility of overflow when the getObject method is executed

The following table shows whether overflow might occur when the getObject method is executed.

Table 7-14: Possibility of overflow when the getObject method is executed

Data type of object obtained by the getObject method	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE	TIME	TSTMP	ROW	BIN, VARBIN
Byte	Y	Y	Y	Y	Y	--	--	--	--	--
Short	Y	Y	Y	Y	Y	--	--	--	--	--
Int	N	Y	Y	Y	Y	--	--	--	--	--
Long	N	N	Y	Y	Y	--	--	--	--	--
Float	N	N	Y	Y	Y	--	--	--	--	--
Double	N	N	N	Y	Y	--	--	--	--	--
BigDecimal	N	N	N	Y	Y	--	--	--	--	--
Boolean	N	N	N	N	N	--	--	--	--	--
String	N	N	N	N	N	N	N	N	--	N
Bytes	--	--	--	--	--	--	--	--	N	N
Date	--	--	--	--	N	N	N	N	--	--

Data type of object obtained by the getObject method	HADB data type									
	SML	INT	DBL PREC	DCML	CHAR, VCHR	DATE	TIME	TSTMP	ROW	BIN, VARBIN
Time	--	--	--	--	N	--	N	N	--	--
Timestamp	--	--	--	--	N	N	N	N	--	--
AsciiStream	--	--	--	--	N	--	--	--	--	N
BinaryStream	--	--	--	--	N	--	--	--	--	N
Object	N	N	N	N	N	N	N	N	N	N
CharacterStream	--	--	--	--	N	--	--	--	--	N

Legend:

SML: SMALLINT

INT: INTEGER

DBL PREC: DOUBLE PRECISION

DCML: DECIMAL

VCHR: VARCHAR

TSTMP: TIMESTAMP

BIN: BINARY

VARBIN: VARBINARY

N: Overflow does not occur regardless of the value.

Y: Overflow might occur depending on the value.

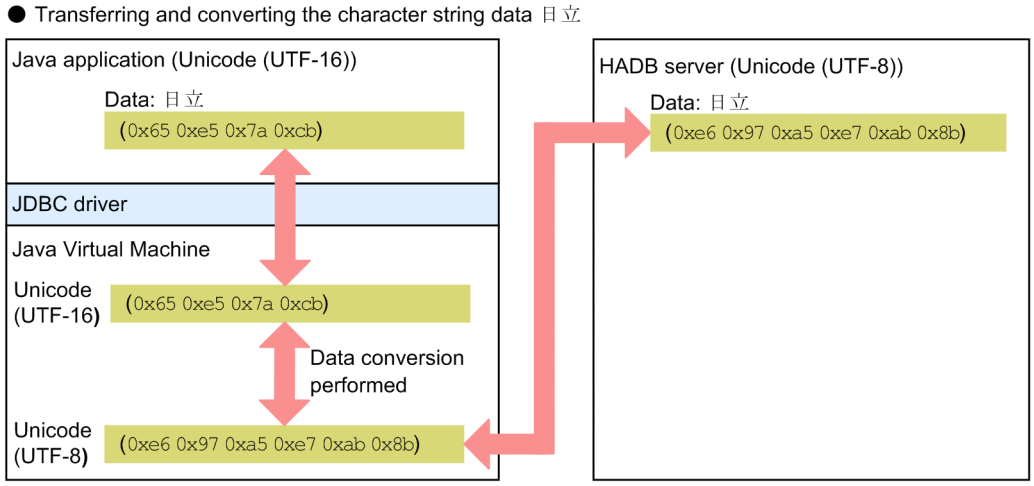
--: Combination that is not allowed.

7.6.4 Conversion of character encoding

Because Java programs consider the character encoding that is used to be Unicode (UTF-16), the JDBC driver performs bi-directional conversion processing of the character encoding between HADB character string data and Unicode (UTF-16). The JDBC driver uses the encoder provided by the Java Virtual Machine (JVM) for this character encoding conversion processing.

The following figure shows an overview of this bi-directional conversion of character encoding between HADB character string data and Unicode (UTF-16).

Figure 7-3: Overview of bi-directional conversion of the character encoding between HADB character string data and Unicode (UTF-16)



When the JDBC driver exchanges character string data with HADB, it specifies the character set name to the Java Virtual Machine's (JVM) encoder. This is how the JDBC driver obtains the HADB server's character encoding (Unicode (UTF-8)) and specifies the character set name that corresponds to that encoding.

The following table shows the character set names that correspond to the HADB server's character encoding for specification in the Java Virtual Machine's (JVM) encoder.

Table 7-15: Names of the character sets supported for the HADB server's character encoding

No.	HADB server's character encoding (character encoding specified in the ADBLANG environment variable)	Name of the character encoding to be specified in the Java Virtual Machine's (JVM) encoder#
1	`Unicode(UTF-8)(UTF8)`	`UTF-8`
2	`Shift-JIS(SJIS)`	`Windows-31j (MS932)`

#

The appropriate character set name shown in the table is specified in the Java Virtual Machine's (JVM) encoder after a connection has been established with the HADB server.

Before a connection is established, the Java Virtual Machine's (JVM) default character set is used for converting the character encoding that is used.

7.7 Troubleshooting

This section explains the JDBC interface method traces and exception trace logs that are provided as troubleshooting functions.

7.7.1 JDBC interface method traces

You can acquire a JDBC interface method trace as troubleshooting information when you call a method in the JDBC interface.

(1) Environment setup

(a) Connection with the DriverManager class

The following explains the environment setup procedure.

To set up the environment:

1. Specify a valid log writer by executing the `setLogWriter` method of the `DriverManager` class.
2. Connect to the HADB server by executing the `getConnection` method of the `DriverManager` class. In the `url` or `info` argument of the `getConnection` method, specify that a JDBC interface method trace is to be acquired (specify `methodtrace` and `tracenum`).

For details about the specification of the `url` argument of the `getConnection` method, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

For details about the specification of the `info` argument of the `getConnection` method, see (d) Values to be specified in the info argument (specifying the user properties) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

(b) Connection with the DataSource class

The following explains the environment setup procedure.

To set up the environment:

1. Specify a valid log writer by executing the `setLogWriter` method of the `DataSource` or `ConnectionPoolDataSource` interface.
2. Specify that a JDBC interface method trace is to be acquired by executing the `setInterfaceMethodTrace` and `setTraceNumber` methods in the connection information setup and acquisition interface.

For details about the `setLogWriter` method of the `DataSource` interface, see 10.2.7 setLogWriter(PrintWriter out). For details about the `setLogWriter` method of the `ConnectionPoolDataSource` interface, see 10.3.7 setLogWriter(PrintWriter out).

For details about the `setInterfaceMethodTrace` method, see 10.5.14 setInterfaceMethodTrace(boolean flag). For details about the `setTraceNumber` method, see 10.5.18 setTraceNumber(int num).

(2) Rules for acquiring a JDBC interface method trace

This subsection describes the rules for acquiring a JDBC interface method trace.

- Trace information is acquired when a method in the JDBC interface is called and when processing is returned from the method. However, trace information is not acquired for methods executed before a connection to the database is established. Trace information is also not acquired for the following methods:

  Driver interface

  - `acceptsURL(String url)`
  - `getMajorVersion()`
  - `getMinorVersion()`
  - `getPropertyInfo(String url, Properties info)`
  - `jdbcCompliant()`

  DataSource and ConnectionPoolDataSource interfaces

  - `getLoginTimeout()`
  - `getLogWriter()`
  - `setLoginTimeout(int seconds)`
  - `setLogWriter(PrintWriter out)`

  Wrapper interfaces

  - `isWrapperFor(Class<?> iface)`
  - `unwrap(Class<T> iface)`
- Trace information is stored for the specified number of entries and is output to the specified log writer at the following times:
  - When the `Connection.close` method is called (normal termination)
  - When an `SQLException` is thrown (error occurrence)
  - When a `BatchUpdateException` is thrown (error occurrence)
  - When an `SQLClientInfoException` is thrown (error occurrence)
  - When an `UnsupportedOperationException` is thrown (error occurrence)
- If the number of trace information items exceeds the number of entries, the oldest stored trace information is discarded in chronological order and the newest trace information is retained.
- A JDBC interface method trace uses a single-entry trace area for each `Entry` and each `Return`.

(3) Output example

This subsection shows an output example of a JDBC interface method trace. The item numbers in the explanation below correspond to the bracketed numbers in the figure.

Output example

```
[1]                                     [2]                    [3]
[Hitachi Advanced Data Binder JDBC Driver][JDBC Interface Entry ][AdbStatement.executeQuery]
                                                               [4]
[Hitachi Advanced Data Binder JDBC Driver]                     sql=select * from pp
[Hitachi Advanced Data Binder JDBC Driver][JDBC Interface Return][AdbStatement.executeQuery]
                                                               [5]
[Hitachi Advanced Data Binder JDBC Driver]                     Return=com.hitachi.hadb.jdbc.Adb...
[Hitachi Advanced Data Binder JDBC Driver][JDBC Interface Entry ][AdbResultSet.getMetaData]
[Hitachi Advanced Data Binder JDBC Driver][JDBC Interface Return][AdbResultSet.getMetaData]
[Hitachi Advanced Data Binder JDBC Driver]                     Return=com.hitachi.hadb.jdbc.Adb...
```

Explanation

1. `[Hitachi Advanced Data Binder JDBC Driver]`
   Name of the JDBC driver
2. `[JDBC Interface Entry ]` and `[JDBC Interface Return]`
   `[JDBC Interface Entry ]`: Call to a JDBC method
   `[JDBC Interface Return]`: Return from a JDBC method
3. [*XXXXX*.*YYYYY*]
   YYYYY method of the *XXXXX* class
4. `select * from pp`
   Argument in the JDBC method (for the password argument, an asterisk (`*`) is output, as in `password=*`)
5. `com.hitachi.hadb.jdbc.Adb...`
   Value returned from the JDBC method

7.7.2 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 cause of the failure is output to the exception trace log.

The following constitute the output contents:

- Information (such as error messages) generated when the exception occurred
- Execution record of JDBC's API methods up to the point where the exception occurred

When this function is used, information about JDBC's API methods that are called from an application program is stored in the JDBC driver memory. If an `SQLException`, `BatchUpdateException`, `SQLClientInfoException`, or `UnSupportedOperationException` occurs, the information stored in the memory is output to a file before the exception is thrown.

(1) Methods to be acquired and setup for log acquisition

(a) 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 coded in the `java.sql` and `javax.sql` packages found in the API specifications of Java Platform Standard Edition 6.

Methods that satisfy the following condition are acquired:

- Methods listed in Table 7-16: Methods that are acquisition targets of the exception trace log and their applicable trace acquisition levels and for which a trace acquisition level that is needed to acquire the trace is specified

Methods that only look up and return information found in objects or methods that only store information into objects, such as a `get`*XXX* method of the `ResultSet` object, a `set`*XXX* method of the `PreparedStatement` method, or the `isClosed` method of the `Connection` object, are not acquisition targets.

The following table lists the methods that are acquisition targets of the exception trace log. The table also shows the trace acquisition levels applicable to each method.

Table 7-16: Methods that are acquisition targets of the exception trace log and their applicable trace acquisition levels

Class	Method	Trace acquisition levels				
		1	2	3	4	5[#1]
`Connection`	`void close()`	Y	Y	Y	Y	Y
	`void commit()`	N	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()`	N	Y	Y	Y	Y
	`boolean isValid(int timeout)`	N	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]	N	Y	Y	Y	Y
	`void setAutoCommit(boolean autoCommit)`	N	Y	Y	Y	Y
`DatabaseMetaData`	`boolean autoCommitFailureClosesAllResultSets()`	N	Y	Y	Y	Y
	`ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)`	N	Y	Y	Y	Y
	`ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)`	N	Y	Y	Y	Y
	`ResultSet getCatalogs()`	N	Y	Y	Y	Y
	`ResultSet getClientInfoProperties()`	N	Y	Y	Y	Y
	`ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)`	N	Y	Y	Y	Y

Class	Method	Trace acquisition levels				
		1	2	3	4	5[#1]
	ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	N	Y	Y	Y	Y
	Connection getConnection()	N	Y	Y	Y	Y
	ResultSet getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	N	Y	Y	Y	Y
	ResultSet getExportedKeys(String catalog, String schema, String table)	N	Y	Y	Y	Y
	ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)	N	Y	Y	Y	Y
	ResultSet getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	N	Y	Y	Y	Y
	ResultSet getImportedKeys(String catalog, String schema, String table)	N	Y	Y	Y	Y
	ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	N	Y	Y	Y	Y
	ResultSet getPrimaryKeys(String catalog, String schema, String table)	N	Y	Y	Y	Y
	ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	N	Y	Y	Y	Y
	ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	N	Y	Y	Y	Y
	ResultSet getPseudoColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)	N	Y	Y	Y	Y
	RowIdLifetime getRowIdLifetime()	N	Y	Y	Y	Y
	ResultSet getSchemas()	N	Y	Y	Y	Y
	ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	N	Y	Y	Y	Y
	ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	N	Y	Y	Y	Y

Class	Method	Trace acquisition levels				
		1	2	3	4	5[#1]
	ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	N	Y	Y	Y	Y
	ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	N	Y	Y	Y	Y
	ResultSet getTableTypes()	N	Y	Y	Y	Y
	ResultSet getTypeInfo()	N	Y	Y	Y	Y
	ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	N	Y	Y	Y	Y
	ResultSet getVersionColumns(String catalog, String schema, String table)	N	Y	Y	Y	Y
Driver	Connection connect(String url, Properties info)	Y	Y	Y	Y	Y
PreparedStatement	boolean execute()[#2]	N	Y	Y	Y	Y
	ResultSet executeQuery()[#2]	N	Y	Y	Y	Y
	int executeUpdate()[#2]	N	Y	Y	Y	Y
	long executeLargeUpdate()[#2]	N	Y	Y	Y	Y
	ResultSetMetaData getMetaData()	N	Y	Y	Y	Y
	boolean execute(String sql)[#3, #4]	N	Y	Y	Y	Y
	int[] executeBatch()[#4]	N	Y	Y	Y	Y
	long[] executeLargeBatch()[#4]	N	Y	Y	Y	Y
	ResultSet executeQuery(String sql)[#3, #4]	Y	Y	Y	Y	Y
	int executeUpdate(String sql)[#3, #4]	Y	Y	Y	Y	Y
	long executeLargeUpdate(String sql)[#3, #4]	Y	Y	Y	Y	Y
	ParameterMetaData getParameterMetaData()	N	Y	Y	Y	Y
ResultSet	boolean absolute(int row)	N	Y	Y	Y	Y
	void afterLast()	N	Y	Y	Y	Y
	void beforeFirst()	N	Y	Y	Y	Y
	void close()	N	Y	Y	Y	Y
	boolean first()	N	Y	Y	Y	Y
	ResultSetMetaData getMetaData()	N	Y	Y	Y	Y
	int getHoldability()	N	Y	Y	Y	Y
	Statement getStatement()	N	Y	Y	Y	Y

Class	Method	Trace acquisition levels				
		1	2	3	4	5[#1]
	boolean isClosed()	N	Y	Y	Y	Y
	boolean last()	N	Y	Y	Y	Y
	boolean next()	N	Y	Y	Y	Y
	boolean relative(int rows)	N	Y	Y	Y	Y
	boolean isAfterLast()	N	Y	Y	Y	Y
	boolean isBeforeFirst()	N	Y	Y	Y	Y
	boolean isLast()	N	Y	Y	Y	Y
Statement	void cancel()	N	Y	Y	Y	Y
	void close()	Y	Y	Y	Y	Y
	boolean execute(String sql)	Y	Y	Y	Y	Y
	int[] executeBatch()	N	Y	Y	Y	Y
	long[] executeLargeBatch()	N	Y	Y	Y	Y
	ResultSet executeQuery(String sql)	Y	Y	Y	Y	Y
	int executeUpdate(String sql)	Y	Y	Y	Y	Y
	long executeLargeUpdate(String sql)	Y	Y	Y	Y	Y
	ResultSet getResultSet()	N	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

Legend:

Y: Exception trace log is acquired.

N: Exception trace log is not acquired.

#1

When the trace acquisition level is 5, an exception trace log that includes internal calls is acquired.

#2

method-name(1) is output as the method name.

#3

method-name(2) is output as the method name.

#4

This method overrides the method in the Statement class.

(b) Setup for acquisition of the exception trace log (setting properties)

You use system properties, user properties, or URL connection properties to set the file output destination for 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 following table lists and describes the items that are specified in properties.

Table 7-17: Exception trace log items specified in properties

Item	Property	Description	Default value#
File output destination	`adb_jdbc_exc_trc_out_path`	Specify the absolute path of the directory to which the exception trace log is to be output. The exception trace log is output directly under the specified directory	Current directory
Number of outputs to the file	`adb_jdbc_info_max`	Specify the maximum number of information items to be output to one file. The value must be in the range from 1 to 50. The actual maximum number of information items to be output to one file is *number of outputs to the file* × *number of information items to be acquired in memory*. For the number of outputs to the file, each of the formats from Format 2 to Format 4 shown in (2) Exception trace log output formats counts 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 the file, the items are wrapped into a second file. The file names are as follows: • `adbjdbcexception01.trc` • `adbjdbcexception02.trc` Note that the output destination file does not change between Format 1 and Format 2 shown in (2) Exception trace log output formats.	5
Number of information items to be acquired in memory	`adb_jdbc_cache_info_max`	Specify the maximum number of information items to be stored in memory. The value must be in the range from 500 to 10,000. For the information acquired in memory, each method shown in Table 7-16: Methods that are acquisition targets of the exception trace log and their applicable trace acquisition levels 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 order.	1,000
Trace acquisition level	`adb_jdbc_trc_out_lv`	Specify a trace acquisition level. The value must be 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

#

In an exception trace log that is acquired in the following cases, the system assumes that no value is specified for the property. In this case, the default value applies.

- When an invalid value is specified in the properties and an `SQLException` is thrown at the time that a connection to the database is established.
- When the Java Virtual Machine (JVM) denies the JDBC driver permission to exchange properties because of security manager reasons.
- Before the initial connection of the Java Virtual Machine (JVM) is established.

(2) Exception trace log output formats

An exception trace log consists of the four formats show below.

Format 1: Header section

```
[AA....AA] Hitachi Advanced Data Binder JDBC Driver BB-CC
```

Format 2: Method execution history (start of a method's execution)

```
AAAAAAAAAAAAAAAAAAAAAAA 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 of output that occurred

```
AAAAAAAAAAAAAAAAAAAAAAA BB....BB:Exception:
II....II
```

Formats 2 and 3 are output in time sequence as many times as the number of methods that are executed.

(a) Explanation of variables in Format 1

AA....AA

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

JDBC driver's version number

CC

JDBC driver's revision number

(b) Explanation of variables in Formats 2, 3, and 4

AAAAAAAAAAAAAAAAAAAAAAA

Acquisition date and time of the exception trace log, in the following format:

```
YYYY/MM/DD hh:mm:ss.sss
```

YYYY: Year (Western calendar)

MM: Month

DD: Date

hh: Hour (24-hour clock)

mm: Minute

ss.sss: Second (includes 3 digits after the decimal point)

BB....BB:

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, and thread group name. The Java Virtual Machine (JVM) determines the format.

bb....bb:

Hash code of the object. The Java Virtual Machine (JVM) determines the format.

C

Call identification information for the method:

`E`:

The information is history information for when the method was started.

`R`:

The information is history information for when the method terminated normally.

DD....DD:

Object identifier and method name, in the following format:

```
aa....aa.bb....bb
```

aa....aa:

Object identifier (maximum of 32 characters). The Java Virtual Machine (JVM) determines the format.

bb....bb:

Method name

EE....EE:

Connection ID (maximum of four characters)

FF....FF:

Section ID (maximum of four characters)

GG....GG:

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 (maximum of 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` class

For the `info` argument in `connect(String url,Properties info)` of the `Driver` class, the value of the following property is replaced by one asterisk (`*`) and then output:

- `password`

HH....HH

Return value of the method, in the format shown below. 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 (JVM) determines the format.

```
Return=aa....aa
```

aa....aa:

Method's return value

II....II:

Troubleshooting information, in the following format:

```
ExceptionClass: aa....aa
ConnectionInformation: bb....bb
Message: cc....cc
ErrorCode: dd....dd
UpdateCounts: ee....ee, ..<omitted>.. ,ee....ee
ff....ff
```

aa....aa:

Execution class name of the exception object that was thrown

bb....bb:

Connection information for the exception object, in the format shown below. 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: Type of connection information item:

- `host` (HADB server's host name)
- `port` (HADB server's port number)
- `user` (authorization identifier)
- `sqlwaittime` (HADB client's maximum response wait time (seconds))

zz.....zz: Contents of the connection information item. Note that the password part of `user` is not displayed.

cc....cc:

Message of the exception object

If there are multiple messages, each message is displayed, separated by an end-of-line code, after the message corresponding to `SQLCODE`. In this case, the message that is returned by the `getMessage` method of an exception object is also displayed as a character string separated by an end-of-line code.

dd....dd:

`SQLCODE` error code (maximum of 11 characters)

This item is output when the execution class of the thrown exception object is the following class or subclass:

- `SQLException`

ee.....ee:

Number of update rows for each update statement in a batch update that was executed normally before this exception occurred (maximum of 11 characters).

This item is output when the execution class of the exception object is `BatchUpdateException`.

If the number of update rows cannot be obtained, an asterisk (*) is output.

ff.....ff:

Stack trace in which the exception-throwing method is set as the base point. The Java Virtual Machine (JVM) determines the format.

(3) Output example and analysis methodology

(a) Output example

```
[1] Hitachi Advanced Data Binder JDBC Driver VV-RR#
2011/07/06 23:07:09.129 Thread[main,5,main]@1259414:[E][AdbConnection@82c01f.createSt
atement(1)]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:09.160 Thread[main,5,main]@1259414:[R][AdbConnection@82c01f.createSt
atement(1)]
                        ConnectionID(1) : SID(0)
                        Return=com.hitachi.hadb.jdbc.AdbStatement@1e4cbc4
2011/07/06 23:07:09.160 Thread[main,5,main]@1259414:[E][AdbStatement@1e4cbc4.execute]
                        ConnectionID(1) : SID(0)
                        sql=DELETE FROM SEINO_TABLE
2011/07/06 23:07:14.285 Thread[main,5,main]@1259414:[E][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:14.301 Thread[main,5,main]@1259414:[R][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:14.301 Thread[main,5,main]@1259414:[R][AdbStatement@1e4cbc4.execute]
                        ConnectionID(1) : SID(1)
                        Return=false
2011/07/06 23:07:14.301 Thread[main,5,main]@1259414:[E][AdbConnection@82c01f.prepareS
tatement(1)]
                        ConnectionID(1) : SID(0)
                        sql=INSERT INTO SEINO_TABLE VALUES(?, ?)
2011/07/06 23:07:14.348 Thread[main,5,main]@1259414:[R][AdbConnection@82c01f.prepareS
tatement(1)]
                        ConnectionID(1) : SID(0)
                        Return=com.hitachi.hadb.jdbc.AdbPreparedStatement@15d56d5
2011/07/06 23:07:26.567 Thread[main,5,main]@1259414:[E][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:26.567 Thread[main,5,main]@1259414:[R][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:26.567 Thread[main,5,main]@1259414:[E][AdbStatement@1e4cbc4.executeQ
uery]
                        ConnectionID(1) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2011/07/06 23:07:26.676 Thread[main,5,main]@1259414:[R][AdbStatement@1e4cbc4.executeQ
uery]
                        ConnectionID(1) : SID(1)
                        Return=com.hitachi.hadb.jdbc.AdbResultSet@3eca90
2011/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][AdbResultSet@3eca90.close]
                        ConnectionID(1) : SID(1)
2011/07/06 23:07:28.332 Thread[main,5,main]@1259414:[E][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][AdbConnection@82c01f.commit]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:28.332 Thread[main,5,main]@1259414:[R][AdbResultSet@3eca90.close]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:28.332 Thread[Thread-0,5,main]@30090737:[E][AdbConnection@82c01f.pre
pareStatement(1)]
                        ConnectionID(1) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2011/07/06 23:07:28.332 Thread[Thread-0,5,main]@30090737:[R][AdbConnection@82c01f.pre
pareStatement(1)]
                        ConnectionID(1) : SID(0)
                        Return=com.hitachi.hadb.jdbc.AdbPreparedStatement@2808b3
```

```
2011/07/06 23:07:28.348 Thread[Thread-1,5,main]@5462872:[E][AdbConnection@82c01f.prep
areStatement(1)]
                        ConnectionID(1) : SID(0)
                        sql=DELETE FROM SEINO_TABLE WHERE I1=?
2011/07/06 23:07:28.358 Thread[Thread-1,5,main]@5462872:[E][AdbConnection@82c01f.comm
it]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:29.672 Thread[Thread-1,5,main]@5462872:[R][AdbConnection@82c01f.comm
it]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:30.098 Thread[Thread-1,5,main]@5462872:[R][AdbConnection@82c01f.prep
areStatement(1)]
                        ConnectionID(1) : SID(0)
                        Return=com.hitachi.hadb.jdbc.AdbPreparedStatement@922804
2011/07/06 23:07:30.332 Thread[Thread-2,5,main]@25253977:[E][AdbConnection@82c01f.rol
lback(1)]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[R][AdbConnection@82c01f.rol
lback(1)]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[E][AdbConnection@82c01f.clo
se]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:42.098 Thread[Thread-2,5,main]@25253977:[R][AdbConnection@82c01f.clo
se]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:42.535 Thread[Thread-1,5,main]@5462872:Exception:
ExceptionClass: SQLException
ConnectionInformation: *
Message: KFAA71206-E Processing cannot continue because the connection is already clo
sed. [AdbPreparedStatement.setInt]
ErrorCode: -1020006
java.sql.SQLException: KFAA71206-E Processing cannot continue because the connection
is already closed. [AdbPreparedStatement.setInt]
at com.hitachi.hadb.jdbc.JdbMakeException.generateSQLException(JdbMakeException.java:
31)
at com.hitachi.hadb.jdbc.AdbStatement.generateClosedSQLException(AdbStatement.java:30
05)
at com.hitachi.hadb.jdbc.AdbPreparedStatement.setInt(AdbPreparedStatement.java:1170)
at Exception1.run(ExceptionTraceSample.java:57)
[2] Hitachi Advanced Data Binder JDBC Driver VV-RR#
2011/07/06 23:07:25.723 Thread[Thread-3,5,main]@13249998:[E][AdbConnection@119cca4.pr
epareStatement(1)]
                        ConnectionID(1) : SID(0)
                        sql=SELECT * FROM SEINO_TABLE
2011/07/06 23:07:25.770 Thread[Thread-4,5,main]@25839584:[E][AdbConnection@119cca4.ro
llback(1)]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:25.770 Thread[Thread-4,5,main]@25839584:[R][AdbConnection@119cca4.ro
llback(1)]
                        ConnectionID(1) : SID(0)
2011/07/06 23:07:25.770 Thread[Thread-5,5,main]@24431647:[E][AdbConnection@119cca4.pr
epareStatement(1)]
                        ConnectionID(1) : SID(0)
                        sql=SELECT ** FROM SEINO_TABLE
2011/07/06 23:07:25.863 Thread[Thread-5,5,main]@24431647:Exception:
ExceptionClass: SQLException
ConnectionInformation: user(ADBUSER01), sqlwaittime(0), host(dragon2), port(20249)
Message: KFAA30105-E Token "*"(non-reserved word), which is after token "*", is inval
id.[AdbStatement.prepare]
ErrorCode: -105
java.sql.SQLException: KFAA30105-E  Token "*"(non-reserved word), which is after toke
n "*", is invalid.[AdbStatement.prepare]
at com.hitachi.hadb.jdbc.JdbSection.prepare(JdbSection.java:1497)
```

```
at com.hitachi.hadb.jdbc.AdbStatement.prepare(AdbStatement.java:2834)
at com.hitachi.hadb.jdbc.AdbPreparedStatement.<init>(AdbPreparedStatement.java:109)
at com.hitachi.hadb.jdbc.AdbConnection.prepareStatement(AdbConnection.java:1041)
at Exception1.run(ExceptionTraceSample.java:64)
```

#

For *VV-RR*, the version of the JDBC driver is output.

(b) Analysis methodology

This subsection explains the analysis methodology for an exception trace log. You can use a text editor to view an exception trace log.

This analysis example analyzes the exception trace log shown in (a) Output example.

To analyze the exception trace log:

1. From the sequentially numbered information, extract 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 times.
   The following table shows what the results look like.

Table 7-18: 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
2011/07/06 23:07:09.129	AdbConnection@82c01f.createStatement(1)			
2011/07/06 23:07:09.160	AdbStatement@1e4cbc4.execute			
2011/07/06 23:07:14.285	AdbConnection@82c01f.commit			
2011/07/06 23:07:14.301	AdbConnection@82c01f.prepareStatement(1)			
2011/07/06 23:07:26.567	AdbConnection@82c01f.commit			
2011/07/06 23:07:26.567	AdbConnection@82c01f.commit			
2011/07/06 23:07:26.567	AdbStatement@1e4cbc4.executeQuery			
2011/07/06 23:07:28.332	AdbResultSet@3eca90.close	AdbConnection@82c01f.prepareStatement(1)		
2011/07/06 23:07:28.332	AdbConnection@82c01f.commit			

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
2011/07/06 23:07:28.348			AdbConnection@82c01f.prepareStatement(1)	
2011/07/06 23:07:28.358			AdbConnection@82c01f.commit	
2011/07/06 23:07:30.332				AdbConnection@82c01f.rollback(1)
2011/07/06 23:07:42.098				AdbConnection@82c01f.close
2011/07/06 23:07:42.535			SQLException occurred "KFAA71206-E Processing cannot continue because the connection is already closed."	

4. Check the nature of the exception error.

   The information indicates that an `SQLException` occurred in Thread 3 on July 6, 2011 at 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 four threads were being processed in the same connection.

   - Thread 1 at `2011/07/06 23:07:09.129`
   - Thread 2 at `2011/07/06 23:07:28.332`
   - Thread 3 at `2011/07/06 23:07:28.348`
   - Thread 4 at `2011/07/06 23:07:30.332`

6. Search for the location of the cause of the error.

   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 we learn that Thread 4 executed the `Connection.close` method on July 6, 2011 at 23:07:42.098. From this, we know that the reason for the `SQLException` that occurred in Thread 3 on July 6, 2011 at 23:07:42.535 was that Thread 4 executed the `Connection.close` method on July 6, 2011 at 23:07:42.098.

(4) Required memory size and file size

(a) Required memory size

The memory size required for acquiring an exception trace log is determined from the following formula:

Formula

↑360 × *n* ÷ 1,024↑ (kilobytes)

Explanation of variable

n: Number of information items to be acquired in memory (value of `adb_jdbc_cache_info_max` in the system properties, user properties, or URL connection properties)

(b) Required file size

The approximate file size required for acquiring an exception trace log is determined from the following formula:

Formula

↑180 × *n* × *m* ÷ 1,024↑ + 1 (kilobytes)

Explanation of variables

n: Number of information items to be acquired in memory (value of `adb_jdbc_cache_info_max` in the system properties, user properties, or URL connection properties)

m: Number of file output information items (value of `adb_jdbc_info_max` in the system properties, user properties, or URL connection properties)

(5) Notes

(a) First output after startup of the Java Virtual Machine (JVM)

The first exception trace log output to a file after the Java Virtual Machine (JVM) has started is the one output to the file with the oldest update date and time. If the date and time are the same for both files, the log is output to `adbjdbcexception01.trc`.

(b) 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 will be output to the same file. To acquire traces separately for each process, specify a different file output destination for each process.

The JDBC driver uses the facilities of the Java Virtual Machine (JVM) to create log files in the file system provided by the OS. Therefore, the following items depend on the Java Virtual Machine (JVM) and file system being used:

- Prefix for the absolute pathname
- Path delimiter character
- Maximum number of characters for the output destination file (absolute path)
- Size per file

(c) Error handling procedure

No information is output to the exception trace log when file creation or output fails. An error message is not returned to the application program and file output is not retried.

(d) Character encoding

The exception trace log is output using the default conversion character set of the Java Virtual Machine (JVM) that is being used.

7.8 Scalar functions that can be specified in the escape clause

The following table lists the scalar functions that can be specified in the escape clause.

Table 7-19: Scalar functions that can be specified in the escape clause

Scalar function	Standard format of scalar function
Mathematical functions	ABS(number)
	ACOS(float)
	ASIN(float)
	ATAN(float)
	ATAN2(float1, float2)
	CEILING(number)
	COS(float)
	DEGREES(number)
	EXP(float)
	FLOOR(number)
	LOG(float)
	LOG10(float)
	MOD(integer1, integer2)
	PI()
	POWER(number, power)
	RADIANS(number)
	RAND([number, number])
	ROUND(number, places)
	SIGN(number)
	SIN(float)
	SQRT(float)
	TAN(float)
	TRUNCATE(number[, places])
String functions	ASCII(string)
	CHAR(code)
	CONCAT(string1,string2)
	LCASE(string)
	LEFT(string,count)
	LENGTH(string)
	LTRIM(string)
	OCTET_LENGTH(string)

Scalar function	Standard format of scalar function
	REPLACE(string1,string2[,string3])
	RIGHT(string,count)
	RTRIM(string)
	SUBSTRING(string, start[, length])
	UCASE(string)
Time and date functions	CURDATE()
	CURRENT_DATE()
	CURRENT_TIME()
	CURRENT_TIMESTAMP()
	CURTIME()
	DAYOFWEEK(date)
	DAYOFYEAR(date)
	EXTRACT(extract-field FROM extract-source)
	NOW()
System function	USER()
Data type conversion function	CONVERT(value, SQLtype)

8 The JDBC 1.2 API

This chapter explains the interfaces and methods in the JDBC 1.2 API.

8.1 Driver interface

This section explains the methods provided by the `Driver` interface.

8.1.1 List of the methods in the Driver interface

(1) Main functions of the Driver interface

The `Driver` interface provides the following main functions:

- Connection to a database
- Validity check of the URL for connection
- Acquisition of connection properties specified with the `DriverManager.getConnection` method
- Return of the JDBC driver version

(2) Methods in the Driver interface that are supported by HADB

The following table lists and describes the methods in the `Driver` interface that are supported by HADB.

Table 8-1: Methods in the Driver interface

No.	Method in the Driver interface	Function
1	`acceptsURL(String url)`	Checks whether a connection can be established with the HADB server by means of the connection information specified by the URL for connection.
2	`connect(String url, Properties info)`	Connects to the HADB server according to the connection information.
3	`getMajorVersion()`	Acquires the JDBC driver's major version.
4	`getMinorVersion()`	Acquires the JDBC driver's minor version.
5	`getPropertyInfo(String url, Properties info)`	Acquires information about the JDBC driver's valid properties.
6	`jdbcCompliant()`	Reports whether the JDBC driver is JDBC Compliant™.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` is thrown.

(3) Required package name and class name

The package and class names required in order to use the `Driver` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `HADBDriver`

8.1.2 acceptsURL(String url)

(1) Function

This method checks whether the JDBC driver can connect to the database specified by the URL.

(2) Format

```
public boolean acceptsURL(String url) throws SQLException
```

(3) Arguments

`String url`

Specifies the URL to be used for a connection.

For details about the specification format of the URL to be used for connection, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

(4) Return value

The method returns `true` if the JDBC driver can connect to the database specified by the URL; if not, the method returns `false`.

(5) Exceptions

None.

8.1.3 connect(String url, Properties info)

(1) Function

This method connects to an HADB server according to the connection information.

Note that you must have the `CONNECT` privilege to execute the `connect` method.

(2) Format

```
public Connection connect(String url, Properties info) throws SQLException
```

(3) Arguments

`String url`

Specifies the URL to be used for connection.

For details about the specification format of the URL to be used for connection, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the

getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

`Properties info`

Specifies a list of property names and their values as the connection arguments. For details about the specification format, see (d) Values to be specified in the info argument (specifying the user properties) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

(4) Return value

The method returns a `Connection` object.

If the specified URL is not valid (the JDBC driver cannot connect to the database specified by the URL), the method returns `null`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurs.
- The specified connection information is not valid.

(6) Notes

You can set connection information in a number of locations, such as in various properties and methods. For details about the priority of connection information to be applied, see (1) Connection information needed when a connection to the HADB server is established in 7.3.3 Connection information priorities.

8.1.4 getMajorVersion()

(1) Function

This method acquires the JDBC driver's major version.

(2) Format

```
public synchronized int getMajorVersion()
```

(3) Arguments

None.

(4) Return value

This method returns the JDBC driver's major version number.

(5) Exceptions

None.

8.1.5 getMinorVersion()

(1) Function

This method acquires the JDBC driver's minor version.

(2) Format

```
public synchronized int getMinorVersion()
```

(3) Arguments

None.

(4) Return value

This method returns the JDBC driver's minor version number.

(5) Exceptions

None.

8.1.6 getPropertyInfo(String url, Properties info)

(1) Function

This method acquires information about the JDBC driver's valid properties.

(2) Format

```
public synchronized DriverPropertyInfo[] getPropertyInfo(String url, Properties info)
 throws SQLException
```

(3) Arguments

`String url`

Specifies the URL to be used for connection.

For details about the specification format of the URL to be used for connection, see (a) Values to be specified in the url argument (specifying the URL for the connection) in (2) Connecting to the HADB server with the getConnection method in 7.3.1 Using the getConnection method of the DriverManager class to connect to the HADB server.

`Properties info`

Specifies a list of property names and their values as the connection arguments.

(4) Return value

This method returns the array of the `DriverPropertyInfo` object for specifying valid properties. If no properties are needed, this array might be empty.

The following table lists the settings for the fields of `DriverPropertyInfo`.

Table 8-2: Settings for fields of DriverPropertyInfo

Property name	DriverPropertyInfo field				
	name	value	description	required	choices
adb_clt_rpc_srv_host	Same as the property name	null	"Host Name"	true	null
adb_clt_rpc_srv_port		null	"Port Number"	true	null
adb_clt_rpc_con_wait_time		"300"	"Connect Wait Time"	false	null
adb_clt_rpc_sql_wait_time		"0"	"Sql Wait Time"	false	null
adb_clt_ap_name		"********"	"Application Name"	false	null
adb_clt_group_name		null	"Client Group Name"	false	null
adb_clt_fetch_size		"1024"	"Fetch Size"	false	null
adb_clt_sql_text_out		"N"	"Text Out"	false	{Y","N"}
adb_clt_trn_iso_lv		"READ_COMMITTED"	"Isolation Level"	false	{"READ_COMMITTED","REPEATABLE_READ"}
adb_clt_sql_order_mode		"BYTE"	"Order Mode"	false	{"BYTE","ISO"}
adb_clt_trn_access_mode		"READ_WRITE"	"Access Mode"	false	{"READ_WRITE","READ_ONLY"}
adb_dbbuff_wrktbl_clt_blk_num		"256"	"Work Table Block Number"	false	null
adb_sql_prep_delrsvd_use_srvdef		"Y"	"Delete Reserved Word Using Server Definition"	false	{Y","N"}
adb_sql_prep_dec_div_rs_prior		"INTEGRAL_PART"	"Decimal Division Result Prior"	false	{"INTEGRAL_PART","FRACTIONAL_PART"}
adb_sql_exe_max_rthd_num		"4"	"Sql Execute Max Real Thread Number"	false	null

Property name	DriverPropertyInfo field				
	name	value	description	required	choices
adb_sql_exe_hashgrp_area_size		"4800"	"Hash Group Area Size"	false	null
adb_sql_exe_hashtbl_area_size		"2000"	"Hash Table Area Size"	false	null
adb_sql_exe_hashflt_area_size		"200"	"Hash Filter Area Size"	false	null
adb_jdbc_exc_trc_out_path		null	"Exception Trace Out Path"	false	null
adb_jdbc_info_max		"5"	"Exception Trace Information Max Number"	false	null
adb_jdbc_cache_info_max		"1000"	"Exception Trace Cache Information Max Number"	false	null
adb_jdbc_trc_out_lv		"1"	"Exception Trace Out Level"	false	null
encodelang		null	"Encode Lang"	false	null
methodtrace		"OFF"	"JDBC Interface Trace"	false	{"ON","OFF"}
tracenum		"500"	"Trace Entry Number"	false	null
sqlwarningkeep		"TRUE"	"Keeping up the Warning Objects"	false	{"TRUE","FALSE"}
user		null	"UserID"	true	null
password		null	"Password"	true	null

This method analyzes the information specified in `url` and `info` and returns the information needed for connecting to the HADB server.

If the `acceptsURL` method returns `false`, this method returns `null`.

(5) Exceptions

None.

8.1.7 jdbcCompliant()

(1) Function

This method reports whether this JDBC driver is JDBC Compliant™.

(2) Format

```
public synchronized boolean jdbcCompliant()
```

(3) Arguments

None.

(4) Return value

If the JDBC driver is JDBC-compliant, this method returns `true`; if not, the method returns `false`.

(5) Exceptions

None.

8.1.8 Escape clause

A part enclosed in curly brackets (`{ }`) in an SQL statement is called an escape clause. An escape clause consists of a keyword and parameters. The keyword is not case sensitive. The following table lists the escape clauses.

Table 8-3: List of escape clauses

Type of escape clause	Keyword
Date	`d`
Time	`t`
Time stamp	`ts`
`LIKE` escape-character	`escape`
Outer join	`oj`
Scalar function	`fn`

For details about the scalar functions that can be specified in an escape clause, see 7.8 Scalar functions that can be specified in the escape clause.

Analysis of escape syntax

You can use the `setEscapeProcessing` method of the `Statement` class to specify whether analysis of the escape syntax is to be enabled. When this specification is omitted, analysis of escape syntax is enabled. Analysis of escape syntax means that the JDBC driver checks each SQL statement for escape clauses. If an SQL statement contains an escape clause, the JDBC driver converts the SQL statement so that the statement can be executed by HADB.

You can reduce the overhead required for syntax analysis by using the `setEscapeProcessing` method of the `Statement` object to disable analysis of escape syntax.

8.2 Connection interface

This section explains the methods provided by the Connection interface.

8.2.1 List of the methods in the Connection interface

(1) Main functions of the Connection interface

The Connection interface provides the following main functions:

- Creation of the Statement and PreparedStatement class objects
- Transaction settlement (commit or rollback)
- Specification of the automatic commit mode

(2) Methods in the Connection interface that are supported by HADB

The following table lists and describes the methods in the Connection interface that are supported by HADB.

Table 8-4: Methods in the Connection interface

No.	Method in the Connection interface	Function
1	clearWarnings()	Clears all warnings reported to the Connection object.
2	close()	Closes the connection with the HADB server.
3	commit()	Applies all changes made since the most recent commit or rollback.
4	createStatement()	Creates a Statement object for sending an SQL statement to the HADB server.
5	createStatement(int resultSetType, int resultSetConcurrency)	
6	createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	
7	getAutoCommit()	Acquires the current automatic commit mode for this Connection object.
8	getCatalog()	Acquires the current catalog name for this Connection object.
9	getHADBConnectionID()	Acquires the connection ID that is assigned to this Connection object.
10	getHADBConnectionSerialNum()	Acquires the connection sequence number that is assigned to this Connection object.
11	getHADBOrderMode()	Acquires for this Connection object the sort order for character string data in a SELECT statement in which the ORDER BY clause is specified.
12	getHADBSQLHashFltSize()	Acquires the size of the hash filter area that is set for this Connection object.
13	getHADBSQLHashTblSize()	Acquires the size of the hash table area that is set for this Connection object.

No.	Method in the Connection interface	Function
14	`getHADBSQLMaxRthdNum()`	Acquires the maximum number of SQL processing real threads that is set for this `Connection` object.
15	`getHADBTransactionID()`	Acquires the transaction ID of the transaction that is being executed.
16	`getHoldability()`	Acquires the current holdability of the `ResultSet` object that is created by using this `Connection` object.
17	`getMetaData()`	Creates a `DatabaseMetaData` object.
18	`getSchema()`	Acquires the current schema name for this `Connection` object.
19	`getTransactionIsolation()`	Acquires the current transaction isolation level for this `Connection` object.
20	`getTypeMap()`	Acquires the `Map` object related to this `Connection` object.
21	`getWarnings()`	Acquires as an `SQLWarning` object a warning reported by a call related to this `Connection` object.
22	`isClosed()`	Returns a value indicating whether this `Connection` object is closed.
23	`isReadOnly()`	Acquires a value indicating whether this `Connection` object is in read-only mode.
24	`isValid(int timeout)`	Acquires the current connection status.
25	`nativeSQL(String sql)`	Converts escape clauses in a specified SQL statement to a format that can be executed by HADB.
26	`prepareStatement(String sql)`	Creates a `PreparedStatement` object for sending an SQL statement with parameters to the HADB server.
27	`prepareStatement(String sql, int resultSetType, int resultSetConcurrency)`	
28	`prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)`	
29	`rollback()`	Undoes all changes made by the transaction and releases all locks currently held by the `Connection` object.
30	`setAutoCommit(boolean autoCommit)`	Sets the automatic commit mode for this connection.
31	`setCatalog(String catalog)`	Sets the passed catalog name and selects a database work subspace for the `Connection` object.
32	`setHADBAuditInfo(int pos,String userinfo)`	Sets the user-specific connection information (user-added information).
33	`setHADBOrderMode(int mode)`	Sets for this `Connection` object the sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified.
34	`setHADBSQLHashFltSize(int areaSize)`	Sets for this `Connection` object the size of the hash filter area.
35	`setHADBSQLHashTblSize(int areaSize)`	Sets for this `Connection` object the size of the hash table area.
36	`setHADBSQLMaxRthdNum(int rthdNum)`	Sets for this `Connection` object the maximum number of SQL processing real threads.
37	`setHoldability(int holdability)`	Sets the holdability of the `ResultSet` object that is created by using this `Connection` object.

No.	Method in the Connection interface	Function
38	`setReadOnly(boolean readOnly)`	Sets this `Connection` object in the read-only mode. Sets the transaction access mode.
39	`setSchema(String schema)`	Sets the name of the schema to access.
40	`setTransactionIsolation(int level)`	Sets the transaction isolation level for this `Connection` object.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` might be thrown.

(3) Required package name and class name

The package and class names required in order to use the `Connection` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbConnection`

8.2.2 clearWarnings()

(1) Function

This method clears all warnings reported to the `Connection` object.

Once this method has been called, the return value of the `getWarnings` method is `null` until a new warning is reported for this `Connection` object.

(2) Format

```
public synchronized void clearWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.3 close()

(1) Function

This method closes the connection with the HADB server.

When a normal connection is in effect, this method disconnects the HADB server, disables the corresponding objects, and releases any unneeded resources.

(2) Format

```
public synchronized void close() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

None.

(6) Notes

- If this method is executed in a pooling environment, the physical connection is not closed. In this case, the `close` method of the `PooledConnection` object must be used to close the physical connection.
- If execution of this method in a pooling environment results in a fatal error and connection pooling becomes unavailable, `ConnectionEventListener.connectionErrorOccurred` does not occur.
- If this method is called by a `Connection` object that is already closed, this method does nothing.
- If an error occurred during row retrieval processing, an unsettled transaction is rolled back without being committed. Connection to the HADB server is closed normally and resources are released.

8.2.4 commit()

(1) Function

This method applies all changes made since the most recent commit or rollback.

If this method is called while the automatic commit mode is enabled, the method performs `commit` processing without throwing an exception.

(2) Format

```
public synchronized void commit() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- A database access error occurs.
- A row retrieval error was detected in an extension of the commit processing.

(6) Notes

- If an exception occurs due to detection of a row retrieval error, the transaction is rolled back without being committed.
- If commit processing fails, the HADB server terminates abnormally.

8.2.5 createStatement()

(1) Function

This method creates a `Statement` object for sending an SQL statement to the HADB server.

(2) Format

```
public synchronized Statement createStatement() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the `Statement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- Creation of the `Statement` object resulted in an error.
- A database access error occurs.

(6) Notes

The holdability of a `ResultSet` object generated from the `Statement` object created by this method is always `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

8.2.6 createStatement(int resultSetType, int resultSetConcurrency)

(1) Function

This method creates a `Statement` object for sending an SQL statement to the HADB server.

(2) Format

```
public synchronized Statement createStatement(int resultSetType, int resultSetConcurr
ency) throws SQLException
```

(3) Arguments

`int resultSetType`

Specifies a result set type.

`int resultSetConcurrency`

Specifies the concurrent processing mode.

(4) Return value

The method returns the `Statement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- Creation of the `Statement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the result set type.
- A value other than a `ResultSet` literal is specified for the concurrent processing mode.
- A database access error occurs.

(6) Notes

- The holdability of a `ResultSet` object generated from the `Statement` object created by this method is always `ResultSet.HOLD_CURSORS_OVER_COMMIT`.
- If `ResultSet.TYPE_SCROLL_SENSITIVE` is specified for the result set type, this JDBC driver changes it to `ResultSet.TYPE_SCROLL_INSENSITIVE`, and then sets an `SQLWarning`.
- For the concurrent processing mode, the JDBC driver supports only `ResultSet.CONCUR_READ_ONLY`. If `ResultSet.CONCUR_UPDATABLE` is specified, the JDBC driver changes it to `ResultSet.CONCUR_READ_ONLY`, and then sets an `SQLWarning`.

8.2.7 createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)

(1) Function

This method creates a `Statement` object for sending an SQL statement to the HADB server.

(2) Format

```
public synchronized Statement createStatement(int resultSetType, int resultSetConcurr
ency, int resultSetHoldability) throws SQLException
```

(3) Arguments

`int resultSetType`

Specifies a result set type.

`int resultSetConcurrency`

Specifies the concurrent processing mode.

`int resultSetHoldability`

Specifies the holdability of the `ResultSet` object.

(4) Return value

The method returns the `Statement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- Creation of the `Statement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the result set type.
- A value other than a `ResultSet` literal is specified for concurrent processing mode.
- A value other than a `ResultSet` literal is specified for the holdability of the `ResultSet` object.
- A database access error occurs.

(6) Notes

- If `ResultSet.TYPE_SCROLL_SENSITIVE` is specified for the result set type, this JDBC driver changes it to `ResultSet.TYPE_SCROLL_INSENSITIVE`, and then sets an `SQLWarning`.
- For concurrent processing mode, the JDBC driver supports only `ResultSet.CONCUR_READ_ONLY`. If `ResultSet.CONCUR_UPDATABLE` is specified, the JDBC driver changes it to `ResultSet.CONCUR_READ_ONLY`, and then sets an `SQLWarning`.
- For the holdability of the `ResultSet` object, the JDBC driver supports only `ResultSet.HOLD_CURSORS_OVER_COMMIT`. If `ResultSet.CLOSE_CURSORS_AT_COMMIT` is

specified, the JDBC driver changes it to `ResultSet.HOLD_CURSORS_OVER_COMMIT`, and then sets an `SQLWarning`.

8.2.8 getAutoCommit()

(1) Function

This method acquires the current automatic commit mode for this `Connection` object.

(2) Format

```
public synchronized boolean getAutoCommit() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the current automatic commit mode for this `Connection` object.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.9 getCatalog()

(1) Function

This method acquires the current catalog name for this `Connection` object.

(2) Format

```
public synchronized String getCatalog() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `null`.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.10 getHADBConnectionID()

(1) Function

This method acquires the connection ID that is assigned to this `Connection` object.

(2) Format

```
public int getHADBConnectionID() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the connection ID that is assigned to this `Connection` object.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.11 getHADBConnectionSerialNum()

(1) Function

This method acquires the connection sequence number that is assigned to this `Connection` object.

(2) Format

```
public int getHADBConnectionSerialNum() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the connection sequence number that is assigned to this `Connection` object.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.12 getHADBOrderMode()

(1) Function

This method acquires for this `Connection` object the current sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified.

(2) Format

```
public int getHADBOrderMode() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the current sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified. One of the following values is returned:

- `AdbConnection.HADB_SQL_ORDER_MODE_BYTE`
- `AdbConnection.HADB_SQL_ORDER_MODE_ISO`

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.13 getHADBSQLHashFltSize()

(1) Function

This method acquires the size of the hash filter area that is set for this `Connection` object.

For details about how to use this method (how to change the size of the hash filter area for each SQL statement to be executed), see (7) Examples in 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

(2) Format

```
public int getHADBSQLHashFltSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the size of the hash filter area (in megabytes) that is set for this `Connection` object.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.14 getHADBSQLHashTblSize()

(1) Function

This method acquires the size of the hash table area that is set for this `Connection` object.

For details about how to use this method (how to change the size of the hash table area for each SQL statement to be executed), see (7) Examples in 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

(2) Format

```
public int getHADBSQLHashTblSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the size of the hash table area (in megabytes) that is set for this `Connection` object.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.15 getHADBSQLMaxRthdNum()

(1) Function

This method acquires the maximum number of SQL processing real threads that is set for this `Connection` object.

For details about how to use this method (how to change the maximum number of SQL processing real threads for each SQL statement to be executed), see (7) Examples in 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

(2) Format

```
public int getHADBSQLMaxRthdNum() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the maximum number of SQL processing real threads that are set for this `Connection` object.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.16 getHADBTransactionID()

(1) Function

This method acquires the transaction ID of the transaction that is being executed.

(2) Format

```
public long getHADBTransactionID() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the transaction ID of the transaction that is being executed when the `getHADBTransactionID` method is issued.

Note that if the `getHADBTransactionID` method is issued before executing the SQL statement, this method returns `0`.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.17 getHoldability()

(1) Function

This method acquires the current holdability of the `ResultSet` object that is created by using this `Connection` object.

(2) Format

```
public synchronized int getHoldability() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.18 getMetaData()

(1) Function

This method creates a `DatabaseMetaData` object.

(2) Format

```
public synchronized DatabaseMetaData getMetaData() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the `DatabaseMetaData` object.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.19 getSchema()

(1) Function

This method acquires the current schema name for this `Connection` object.

(2) Format

```
public synchronized String getSchema() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `null`.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.20 getTransactionIsolation()

(1) Function

This method acquires the current transaction isolation level for this `Connection` object.

(2) Format

```
public synchronized int getTransactionIsolation() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the current transaction isolation level, which is one of the following:

- `Connection.TRANSACTION_READ_COMMITTED`
  This value is returned when `READ COMMITTED` is applied as the transaction isolation level.
- `Connection.TRANSACTION_REPEATABLE_READ`
  This value is returned when `REPEATABLE READ` is applied as the transaction isolation level.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.21 getTypeMap()

(1) Function

This method acquires the `Map` object related to this `Connection` object.

(2) Format

```
public synchronized java.util.Map getTypeMap() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns an empty `java.util.Map` object.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.22 getWarnings()

(1) Function

This method acquires as an `SQLWarning` object a warning reported by a call related to this `Connection` object.

This method acquires the `SQLWarning` object held by the corresponding `Connection` object. By executing the `getNextWarning` method of the acquired `SQLWarning` object, you can acquire the next warning.

(2) Format

```
public synchronized SQLWarning getWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the first `SQLWarning` object. If there is no `SQLWarning` object, the method returns `null`.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.23 isClosed()

(1) Function

This method returns a value indicating whether this `Connection` object is closed.

The HADB server connection is closed when the `close` method is called or when a specific fatal error has occurred. This method is guaranteed to return `true` only when it is executed after a `close` method.

This method cannot be used to determine whether the HADB server connection is valid.

(2) Format

```
public synchronized boolean isClosed() throws SQLException
```

(3) Arguments

None.

(4) Return value

If this `Connection` object is closed, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

None.

8.2.24 isReadOnly()

(1) Function

This method acquires a value indicating whether this `Connection` object is in read-only mode.

(2) Format

```
public synchronized boolean isReadOnly() throws SQLException
```

(3) Arguments

None.

(4) Return value

If this `Connection` object is in read-only mode, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.25 isValid(int timeout)

(1) Function

This method acquires the current connection status.

(2) Format

```
public synchronized boolean isValid(int timeout) throws SQLException
```

(3) Arguments

`int timeout`

Specifies the wait time (in seconds), in the range from 0 to 65,535.
If zero is specified, there will be no limit to the wait time.
If 65,536 or a greater value is specified, `65,535` is assumed.

(4) Return value

If the method verifies that the connection is alive, it returns `true`. If the `Connection` object is closed, or if the wait time specified in the argument has expired and a timeout has occurred, the method returns `false`.

(5) Exceptions

If `-1` or a smaller value is specified in the argument, the JDBC driver throws an `SQLException`.

8.2.26 nativeSQL(String sql)

(1) Function

This method converts escape clauses in a specified SQL statement to a format that can be executed by HADB.

(2) Format

```
public synchronized String nativeSQL(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies an SQL statement.

(4) Return value

The method returns an SQL statement that can be executed by HADB.

If `null` is specified for `sql`, the method returns `null`. If an empty string is specified for `sql`, the method returns an empty string.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- The format of an escape clause in the specified SQL statement is invalid for the following reason:
  - `{` and a keyword are specified, but `}` is missing.
- The specified SQL statement exceeds 16,000,000 characters.

(6) Syntax rules for escape clause

This method converts any escape clauses in the specified SQL statement to a format that can be executed by HADB, and then returns the SQL statement. The following are the syntax rules for an escape clause:

```
escape-clause ::= escape-sequence-for-date-or-time-or-time-stamp
                | escape-sequence-for-escape-character-in-LIKE-predicate
                | escape-sequence-for-outer-join
                | scalar-function-escape-sequence
```

```
escape-sequence-for-date-or-time-or-time-stamp ::= date-escape-sequence
                                                 | time-escape-sequence
                                                 | time-stamp-escape-sequence
date-escape-sequence ::=
          escape-start-code d default-character-string-representation-of-date-data#1 e
scape-end-code
time-escape-sequence ::=
          escape-start-code t default-character-string-representation-of-time-data#2 e
scape-end-code
time-stamp-escape-sequence ::=
          escape-start-code ts default-character-string-representation-of-time-stamp-
data#3 escape-end-code
escape-sequence-for-escape-character-in-LIKE-predicate ::=
          escape-start-code escape escape-character escape-end-code
escape-sequence-for-outer-join ::= escape-start-code oj joined-table escape-end-code
scalar-function-escape-sequence ::= escape-start-code fn scalar-function escape-end-c
ode
scalar-function ::= scalar-function-in-default-format#4
escape-start-code ::= '{'
escape-end-code ::= '}'
```

#1

Character string representation '*YYYY-MM-DD*'

#2

Character string representation '*hh*:*mm*:*ss*[.*f*...]'

#3

Character string representation '*YYYY-MM-DD hh*:*mm*:*ss*[.*f*...]'

#4

For details about the scalar function in the default format, see 7.8 Scalar functions that can be specified in the escape clause.

Note that an escape clause cannot be specified in an underlined part. Because the JDBC driver does not perform syntax analysis on the underlined parts, they will remain the same after conversion and will be subject to syntax analysis by the HADB server.

The following keywords can be used in escape sequences. These keywords are not case sensitive.

1. `d` in a date escape sequence
2. `t` in a time escape sequence
3. `ts` in a time stamp escape sequence
4. `escape` in an escape sequence of an escape character of a LIKE predicate
5. `oj` in an outer join escape sequence
6. `fn` in a scalar function escape sequence

The escape clause entry rules are as follows:

- The space can be used as the delimiter character in an escape clause.
- The delimiter can be inserted following an escape start code, following a keyword, and before an escape end code.
- You can specify multiple escape clauses in a single SQL statement.

- The JDBC driver converts the escape clauses in an SQL statement to a format that can be executed by HADB. Note that only the part of each escape clause that is enclosed in curly brackets is converted. The driver converts nothing outside the escape clauses.

The following table shows the escape clause conversion rules.

Table 8-5: Escape clause conversion rules

Escape clause	Before conversion	After conversion
Date	*escape-start-code* `d` *default-character-string-representation-of-date-data escape-end-code*	*default-character-string-representation-of-date-data*
Time	*escape-start-code* `t` *default-character-string-representation-of-time-data escape-end-code*	*default-character-string-representation-of-time-data*
Time stamp	*escape-start-code* `ts` *default-character-string-representation-of-time-stamp-data escape-end-code*	*default-character-string-representation-of-time-stamp-data*
LIKE	*escape-start-code* `escape` *escape-character escape-end-code*	`escape` *escape-character*
Outer join	*escape-start-code* `oj` *joined-table escape-end-code*	*joined-table*
Scalar function	*escape-start-code* `fn` *scalar-function escape-end-code*	*scalar-function-in-HADB-format*#

#

The JDBC driver converts a scalar function in the default format to the HADB format.

The table below shows the conversion formats of scalar functions whose default format differs from the HADB server format.

In general, the JDBC driver does not check the number of arguments in scalar functions.

Table 8-6: Conversion formats of scalar functions whose default format differs from the HADB server format

Scalar functions	Format before conversion	Format after conversion (HADB format)
Mathematical function	`CEILING(number)`	`CEIL(number)`
	`LOG(float)`	`LN(float)`
	`LOG10(float)`	`LOG(10,float)`
	`RAND([number, number])`	`RANDOM([number, number])`
	`TRUNCATE(number[, places])`	`TRUNC(number[, places])`
String function	`CHAR(code)`	`CHR(code)`
	`LCASE(string)`	`LOWER(string)`
	`OCTET_LENGTH(string)`	`LENGTHB(string)`
	`SUBSTRING(string, start[ ,length])`	`SUBSTR(string, start[ ,length])`
	`UCASE(string)`	`UPPER(string)`
Time and date functions	`CURDATE()`	`CURRENT_DATE`
	`CURRENT_DATE()`	`CURRENT_DATE`
	`CURRENT_TIME()`	`CURRENT_TIME`
	`CURRENT_TIMESTAMP()`	`CURRENT_TIMESTAMP`
	`CURTIME()`	`CURRENT_TIME`

Scalar functions	Format before conversion	Format after conversion (HADB format)
	NOW()	CURRENT_TIMESTAMP
System function	USER()	CURRENT_USER

8.2.27 prepareStatement(String sql)

(1) Function

This method creates a PreparedStatement object for sending an SQL statement with parameters to the HADB server.

(2) Format

```
public synchronized PreparedStatement prepareStatement(String sql) throws SQLExceptio
n
```

(3) Arguments

String sql

Specifies the SQL statement that is to be executed.

(4) Return value

The method returns the PreparedStatement object.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- The Connection object is closed.
- Creation of the PreparedStatement object resulted in an error.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

(6) Notes

The holdability of a ResultSet object generated from the PreparedStatement object created by this method is always ResultSet.HOLD_CURSORS_OVER_COMMIT.

8.2.28 prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

(1) Function

This method creates a `PreparedStatement` object for sending an SQL statement with parameters to the HADB server.

(2) Format

```
public synchronized PreparedStatement prepareStatement(String sql, int resultSetType,
 int resultSetConcurrency) throws SQLException
```

(3) Arguments

`String sql`
Specifies the SQL statement that is to be executed.

`int resultSetType`
Specifies a result set type.

`int resultSetConcurrency`
Specifies the concurrent processing mode.

(4) Return value

The method returns the `PreparedStatement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- Creation of the `PreparedStatement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the result set type.
- A value other than a `ResultSet` literal is specified for the concurrent processing mode.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

(6) Notes

- If `ResultSet.TYPE_SCROLL_SENSITIVE` is specified for the result set type, this JDBC driver changes it to `ResultSet.TYPE_SCROLL_INSENSITIVE`, and then sets an `SQLWarning`.
- For the concurrent processing mode, the JDBC driver supports only `ResultSet.CONCUR_READ_ONLY`. If `ResultSet.CONCUR_UPDATABLE` is specified, the JDBC driver changes it to `ResultSet.CONCUR_READ_ONLY`, and then sets an `SQLWarning`.
- The holdability of a `ResultSet` object generated from the `PreparedStatement` object created by this method is always `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

8.2.29 prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)

(1) Function

This method creates a `PreparedStatement` object for sending an SQL statement with parameters to the HADB server.

(2) Format

```
public synchronized PreparedStatement prepareStatement(String sql, int resultSetType,
 int resultSetConcurrency, int resultSetHoldability) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statement that is to be executed.

`int resultSetType`

Specifies a result set type.

`int resultSetConcurrency`

Specifies the concurrent processing mode.

`int resultSetHoldability`

Specifies the holdability of the `ResultSet` object.

(4) Return value

The method returns the `PreparedStatement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- Creation of the `PreparedStatement` object resulted in an error.
- A value other than a `ResultSet` literal is specified for the result set type.
- A value other than a `ResultSet` literal is specified for the concurrent processing mode.
- A value other than a `ResultSet` literal is specified for the holdability of the `ResultSet` object.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

(6) Notes

- If `ResultSet.TYPE_SCROLL_SENSITIVE` is specified for the result set type, this JDBC driver changes it to `ResultSet.TYPE_SCROLL_INSENSITIVE`, and then sets an `SQLWarning`.

- For the concurrent processing mode, the JDBC driver supports only `ResultSet.CONCUR_READ_ONLY`. If `ResultSet.CONCUR_UPDATABLE` is specified, the JDBC driver changes it to `ResultSet.CONCUR_READ_ONLY`, and then sets an `SQLWarning`.
- For the holdability of the `ResultSet` object, the JDBC driver supports only `ResultSet.HOLD_CURSORS_OVER_COMMIT`. If `ResultSet.CLOSE_CURSORS_AT_COMMIT` is specified, the JDBC driver changes it to `ResultSet.HOLD_CURSORS_OVER_COMMIT`, and then sets an `SQLWarning`.

8.2.30 rollback()

(1) Function

This method undoes all changes made by the transaction and releases all locks currently held by the `Connection` object.

If you call this method while the automatic commit mode is enabled, the method performs rollback processing without throwing an exception.

(2) Format

```
public synchronized void rollback() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- A database access error occurs.

(6) Notes

- If rollback processing is successful, the `ResultSet` object is invalidated.
- If rollback processing fails, the HADB server terminates abnormally.

8.2.31 setAutoCommit(boolean autoCommit)

(1) Function

This method sets the automatic commit mode for this connection.

(2) Format

```
public synchronized void setAutoCommit(boolean autoCommit) throws SQLException
```

(3) Arguments

`boolean autoCommit`

Specifies `true` to enable the automatic commit mode or `false` to disable it.

(4) Return value

None.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

(6) Notes

- When the automatic commit mode is enabled, an SQL statement is committed automatically when its processing is completed. Therefore, each SQL statement is treated as one transaction. When the automatic commit mode is disabled, an SQL statement is not completed until the `commit` or `rollback` method is executed. By default, the automatic commit mode is enabled.
- Automatic commit is performed upon completion of an SQL statement. If the SQL statement returns a `ResultSet` object, the SQL statement is completed when the `ResultSet` object is closed.
- A transaction that is executing when this method is called will not be committed.

8.2.32 setCatalog(String catalog)

(1) Function

This method sets the name of the catalog that is to be passed and selects a database work subspace for the `Connection` object.

(2) Format

```
public synchronized void setCatalog(String catalog) throws SQLException
```

(3) Arguments

`String catalog`

This argument is ignored, if specified.

(4) Return value

None.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.33 setHADBAuditInfo(int pos,String userinfo)

(1) Function

This method sets user-added information, such as the account information of applications that access the HADB server. The user-added information that is set has effect until it is revoked.

The user-added information set by using this function is output as an audit trail at the following times:

- When an SQL statement that includes a `Statement` object generated by using the `Connection` object is executed
- When an SQL statement that includes a `PreparedStatement` object generated by using the `Connection` object is executed
- When the `Connection` object is closed

(2) Format

```
public synchronized void setHADBAuditInfo(int pos,String userinfo) throws SQLExceptio
n
```

(3) Arguments

`int pos`

Specifies which column in the audit trail the user-added information specified by `userinfo` is to be output to. Specify one of the following values:

`1`: Specify this value to output the user-added information specified by `userinfo` to user-added information 1 in the audit trail.

`2`: Specify this value to output the user-added information specified by `userinfo` to user-added information 2 in the audit trail.

`3`: Specify this value to output the user-added information specified by `userinfo` to user-added information 3 in the audit trail.

`String userinfo`

Specifies user-added information.

The user-added information specified here is converted in the character encoding that is used on the HADB server. Make sure that the size of the specified user-added information does not exceed 100 bytes after the character encoding is converted.

Note that a null character (`0x00`) cannot be used.

To revoke the specification of the user-added information, specify `null`.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- The value specified in `pos` is not a value in the range from 1 to 3.
- Character encoding conversion of the user-added information fails.
- The size of the user-added information exceeds 100 bytes after the character encoding is converted.
- The user-added information includes a null character (`0x00`).
- A transaction has already started.

(6) Notes

- This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.
- If the `Connection` object is pooled and then reused, the user-added information that was set by using the `setHADBAuditInfo` method is not reused. In such a case, the status changes to the status that existed before the `setHADBAuditInfo` method was executed.

8.2.34 setHADBOrderMode(int mode)

(1) Function

This method sets for this `Connection` object the sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified.

The information set in this method corresponds to the `adb_clt_sql_order_mode` operand in the client definition.

(2) Format

```
public void setHADBOrderMode(int mode) throws SQLException
```

(3) Arguments

`int mode`

Specifies the sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified. Specify one of the following values:

- `AdbConnection.HADB_SQL_ORDER_MODE_BYTE`
  Sort character string data by bytecode.
- `AdbConnection.HADB_SQL_ORDER_MODE_ISO`
  Sort character string data by sort code (ISO/IEC 14651:2011 compliance).

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- A transaction has already started.
- The specified sort order is not one of the following:
  - `AdbConnection.HADB_SQL_ORDER_MODE_BYTE`
  - `AdbConnection.HADB_SQL_ORDER_MODE_ISO`

(6) Notes

- The sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified is determined in the priority order shown below (the smaller the number, the higher the priority). For example, if 1 and 2 are both specified, 1 takes effect.
  1. Sort order specified with the `setHADBOrderMode` method
  2. Sort order specified with the `adb_clt_sql_order_mode` system property
  3. Value of the `adb_clt_sql_order_mode` property specified in the `info` argument of the `getConnection` method of the `DriverManager` class
  4. Value of `adb_clt_sql_order_mode` property specified in the `url` argument of the `getConnection` method of the `DriverManager` class
  5. Sort order specified with the `adb_sql_order_mode` server definition operand
- When the `setHADBOrderMode` method is used to specify the sort order for character string data in a `SELECT` statement in which the `ORDER BY` clause is specified, that sort order setting is not inherited when the `Connection` object has been pooled and then reused. In such a case, the sort order remains the same as it was before the `setHADBOrderMode` method was executed.
- This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

8.2.35 setHADBSQLHashFltSize(int areaSize)

(1) Function

This method sets for this `Connection` object the size of the hash filter area.

The value set by this method is compatible with the `adb_sql_exe_hashflt_area_size` operand in the client definition.

> **Important**
>
> For the `Statement` or `PreparedStatement` object generated from the `Connection` object for which this method is specified, the size of the hash filter area that is set by this method is applied until the `Connection` object is closed.

For details about how to use this method (how to change the size of the hash filter area for each SQL statement to be executed), see (7) Examples in 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

(2) Format

```
public void setHADBSQLHashFltSize(int areaSize) throws SQLException
```

(3) Arguments

`int areaSize`

Specifies in megabytes the size of the hash filter area to be set.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- An invalid value is specified in `areaSize`.

(6) Notes

- This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.
- For the size of the hash filter area specified by using the `setHADBSQLHashFltSize` method, the previously set value is not inherited if the `Connection` object is pooled and then reused. This is the same case as when the `setHADBSQLHashFltSize` method has not been executed.
- Even if the `setHADBSQLHashFltSize` method is used to set the size of the hash filter area again, the new setting is not applied to the `Statement` or `PreparedStatement` object that has already been generated.

- For details about how the size of the hash filter area is determined, see the explanation of the operand adb_sql_exe_hashflt_area_size in 2.2.3 Operands related to performance.

8.2.36 setHADBSQLHashTblSize(int areaSize)

(1) Function

This method sets for this `Connection` object the size of the hash table area.

The value set by this method is compatible with the `adb_sql_exe_hashtbl_area_size` operand in the client definition.

> **Important**
>
> For the `Statement` or `PreparedStatement` object generated from the `Connection` object for which this method is specified, the size of the hash table area that is set by this method is applied until the `Connection` object is closed.

For details about how to use this method (how to change the size of the hash table area for each SQL statement to be executed), see (7) Examples in 8.2.37 setHADBSQLMaxRthdNum(int rthdNum).

(2) Format

```
public void setHADBSQLHashTblSize(int areaSize) throws SQLException
```

(3) Arguments

`int areaSize`

Specifies in megabytes the size of the hash table area to be set.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- An invalid value is specified in `areaSize`.

(6) Notes

- This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

- For the size of the hash table area specified by using the `setHADBSQLHashTblSize` method, the previously set value is not inherited if the `Connection` object is pooled and then reused. This is the same case as when the `setHADBSQLHashTblSize` method has not been executed.
- Even if the `setHADBSQLHashTblSize` method is used to set the size of the hash table area again, the new setting is not applied to the `Statement` or `PreparedStatement` object that has already been generated.
- For details about how the size of the hash table area is determined, see the explanation of the adb_sql_exe_hashtbl_area_size operand in 2.2.3 Operands related to performance.

8.2.37 setHADBSQLMaxRthdNum(int rthdNum)

(1) Function

This method sets for this `Connection` object the maximum number of SQL processing real threads.

The value set by this method is compatible with the `adb_sql_exe_max_rthd_num` operand in the client definition.

> **Important**
>
> For the `Statement` or `PreparedStatement` object generated from the `Connection` object for which this method is specified, the maximum number of SQL processing real threads that is set by this method is applied until the `Connection` object is closed.

(2) Format

```
public void setHADBSQLMaxRthdNum(int rthdNum) throws SQLException
```

(3) Arguments

`int rthdNum`

Specifies the maximum number of SQL processing real threads to be set.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- An invalid value is specified in `rthdNum`.

(6) Notes

- This is an HADB-specific method provided by the `AdbConnection` interface. For details about the execution method, see 12.2 Wrapper interface.

- For the maximum number of SQL processing real threads specified by using the `setHADBSQLMaxRthdNum` method, the previously set value is not inherited if the `Connection` object is pooled and then reused. This is the same case as when the `setHADBSQLMaxRthdNum` method has not been executed.
- Even if the `setHADBSQLMaxRthdNum` method is used to set the maximum number of SQL processing real threads again, the new setting is not applied to the `Statement` or `PreparedStatement` object that has already been generated.
- For details about how the maximum number of SQL processing real threads is determined, see the explanation of the adb_sql_exe_max_rthd_num operand in 2.2.3 Operands related to performance.

(7) Examples

In the following example, the `setHADBSQLMaxRthdNum` and `getHADBSQLMaxRthdNum` methods are used to change the maximum number of SQL processing real threads for each SQL statement to be executed.

Example 1:

```
   :
   :
Connection cnct = ds.getConnection();
AdbConnection con = cnct.unwrap(com.hitachi.hadb.jdbc.AdbConnection.class);

// Back up the default of the maximum number of SQL processing real threads.
int default_sql_exe_rthd_num = con.getHADBSQLMaxRthdNum();

// Change the maximum number of SQL processing real threads to 0.
con.setHADBSQLMaxRthdNum(0);

// Generate a PreparedStatement object.
// All SQL statements executed with this PreparedStatement object operate under
// the condition where the maximum number of SQL processing real threads is 0.
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM MASTER.SQL_USERS WHE
RE USER_NAME=?");

// Reset the maximum number of SQL processing real threads to the default.
con.setHADBSQLMaxRthdNum(default_sql_exe_rthd_num);
// This operation does not affect the preceding pstmt.

pstmt.setString(1, "FOO");
ResultSet rs = pstmt.executeQuery(); // Execute an SQL statement under the
// condition where the maximum number of SQL processing real threads is 0.
while (rs.next()) {
   :
   :
}
rs.close();
pstmt.close();
   :
   :
```

Example 2:

In the following example, each SQL statement is executed with a different maximum number of SQL processing real threads.

```
   :
   :
Connection cnct = ds.getConnection();
AdbConnection con = cnct.unwrap(com.hitachi.hadb.jdbc.AdbConnection.class);

// Back up the default of the maximum number of SQL processing real threads.
```

```
int default_sql_exe_rthd_num = con.getHADBSQLMaxRthdNum();

// Change the maximum number of SQL processing real threads to 0.
con.setHADBSQLMaxRthdNum(0);

// Generate a PreparedStatement object.
// All SQL statements executed with pstmt1 operate under the condition
// where the maximum number of SQL processing real threads is 0.
PreparedStatement pstmt1 = con.prepareStatement("SELECT * FROM MASTER.SQL_USERS WH
ERE USER_NAME=?");

// Change the maximum number of SQL processing real threads to 4.
con.setHADBSQLMaxRthdNum(4); // This operation does not affect
// the preceding pstmt1.

// Generate a PreparedStatement object.
// All SQL statements executed with pstmt2 operate under the condition
// where the maximum number of SQL processing real threads is 4.
PreparedStatement pstmt2 = con.prepareStatement("SELECT * FROM MASTER.SQL_TABLE_PR
IVILEGES WHERE GRANTOR=?");

// Reset the maximum number of SQL processing real threads to the default.
con.setHADBSQLMaxRthdNum(default_sql_exe_rthd_num);
// This operation does not affect the preceding pstmt1 and pstmt2.

pstmt1.setString(1, "HOGE");
ResultSet rs1 = pstmt1.executeQuery(); // Execute an SQL statement under the
// condition where the maximum number of SQL processing real threads is 0.
while (rs1.next()) {
   :
   :
}
rs1.close();
pstmt1.close();

pstmt2.setString(1, "FOO");
ResultSet rs2 = pstmt2.executeQuery(); // Execute an SQL statement under the
// condition where the maximum number of SQL processing real threads is 4.
while (rs2.next()) {
   :
   :
}
rs2.close();
pstmt2.close();
   :
   :
```

Note

- In the preceding examples, the processing to handle exceptions and other errors is omitted.
- The preceding examples will also be useful for reference purposes when you change the size of the hash table area for each SQL statement to be executed by using the `setHADBSQLHashTblSize` and `getHADBSQLHashTblSize` methods.
- The preceding examples will also be useful for reference purposes when you change the size of the hash filter area for each SQL statement to be executed by using the `setHADBSQLHashFltSize` and `getHADBSQLHashFltSize` methods.

8.2.38 setHoldability(int holdability)

(1) Function

This method sets the holdability of the `ResultSet` object that is created by using this `Connection` object.

(2) Format

```
public synchronized void setHoldability(int holdability) throws SQLException
```

(3) Arguments

`int holdability`

This argument is ignored, if specified. The JDBC driver always assumes that `ResultSet.HOLD_CURSORS_OVER_COMMIT` is specified.

(4) Return value

None.

(5) Exceptions

If this `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.39 setReadOnly(boolean readOnly)

(1) Function

This method sets this `Connection` object in the read-only mode. The method sets the transaction access mode.

(2) Format

```
public synchronized void setReadOnly(boolean readOnly) throws SQLException
```

(3) Arguments

`boolean readOnly`

Specify `true` to set the `Connection` object in the read-only mode; otherwise, specify `false`.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- A transaction has already started.
- A `ResultSet` object created by this `Connection` object with `HOLD_CURSORS_OVER_COMMIT` specified for holdability has not been closed.

(6) Notes

- The transaction access mode is determined in the priority order described below. A smaller number represents a higher priority (for example, 1 has a higher priority than 2).
  1. Transaction access mode specified with the `setReadOnly` method
  2. Transaction access mode specified with the `adb_clt_trn_access_mode` system property
  3. `adb_clt_trn_access_mode` property value specified in the `info` argument of the `getConnection` method of the `DriverManager` class
  4. Value of `adb_clt_trn_access_mode` specified in the `url` argument of the `getConnection` method of the `DriverManager` class
- In the case of the transaction access mode specified with the `setReadOnly` method, if the `Connection` object is pooled and then is reused, the previous transaction access mode is not inherited. This is the same status as when the `setReadOnly` method has not been executed.

8.2.40 setSchema(String schema)

(1) Function

This method sets the name of the schema to access. This method does not set a schema name. The value specified in the argument of this method is ignored.

(2) Format

```
public synchronized void setSchema(String schema) throws SQLException
```

(3) Arguments

`String schema`

Specifies the schema name. However, any value you specify is ignored.

(4) Return value

None.

(5) Exceptions

If the `Connection` object is closed, the JDBC driver throws an `SQLException`.

8.2.41 setTransactionIsolation(int level)

(1) Function

This method sets the transaction isolation level for this `Connection` object.

(2) Format

```
public synchronized void setTransactionIsolation(int level) throws SQLException
```

(3) Arguments

`int level`

Specifies the transaction isolation level to be applied. Specify one of the following values:

- `Connection.TRANSACTION_READ_COMMITTED`
  Specifies that `READ COMMITTED` is to be applied as the transaction isolation level.
- `Connection.TRANSACTION_REPEATABLE_READ`
  Specifies that `REPEATABLE READ` is to be applied as the transaction isolation level.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed.
- A transaction has already started.
- A `ResultSet` object created by this `Connection` object with `HOLD_CURSORS_OVER_COMMIT` specified for holdability has not been closed.
- The value specified for the transaction isolation level is neither of the following:
  - `Connection.TRANSACTION_READ_COMMITTED`
  - `Connection.TRANSACTION_REPEATABLE_READ`

(6) Notes

- The transaction isolation level is determined in the priority order described below. A smaller number represents a higher priority (for example, 1 has a higher priority than 2).
  1. Transaction isolation level specified with the `setTransactionIsolation` method
  2. Transaction isolation level specified in the `adb_clt_trn_iso_lv` system property
  3. `adb_clt_trn_iso_lv` property value specified in the `info` argument of the `getConnection` method of the `DriverManager` class
  4. Value of `adb_clt_trn_iso_lv` specified in the `url` argument of the `getConnection` method of the `DriverManager` class

5. Transaction isolation level specified in the `adb_sys_trn_iso_lv` server definition operand

- The transaction isolation level specified with the `setTransactionIsolation` method is not inherited when the `Connection` object has been pooled and then reused. In such a case, it remains the same as it was before the `setTransactionIsolation` method was executed.

8.3 Statement interface

This section explains the methods provided by the `Statement` interface.

8.3.1 List of the methods in the Statement interface

(1) Main functions of the Statement interface

The `Statement` interface provides the following main functions:

- Execution of SQL statements
- Creation of a result set (`ResultSet` object) for retrieval results
- Return of the number of updated rows as the result of updating
- Specification of the maximum number of rows to be retrieved
- Specification of a retrieval limit time

(2) Methods in the Statement interface that are supported by HADB

The following table lists and describes the methods in the `Statement` interface that are supported by HADB.

Table 8-7: Methods in the Statement interface

No.	Method in the Statement interface	Function
1	`addBatch(String sql)`	Adds SQL statements to the `Statement` object's batch.
2	`cancel()`	Cancels the SQL statements executing in the corresponding object and in objects using the same connection as that object.
3	`clearBatch()`	Clears all SQL statements registered in this `Statement` object's batch.
4	`clearWarnings()`	Clears all warnings that have been reported for this `Statement` object.
5	`close()`	Closes the `Statement` object and any `ResultSet` object created from this `Statement` object.
6	`closeOnCompletion()`	Closes this `Statement` object when all result sets that depend on the `Statement` object are closed.
7	`execute(String sql)`	Executes an SQL statement.
8	`executeBatch()`	Executes the SQL statements registered in a batch and returns the number of updated rows as `int` data in an array.
9	`executeLargeBatch()`	Executes the SQL statements registered in a batch and returns the number of updated rows as `long` data in an array.
10	`executeLargeUpdate(String sql)`	Executes an SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as `long` data.
11	`executeQuery(String sql)`	Executes a retrieval SQL statement and returns a `ResultSet` object containing the retrieval result.
12	`executeUpdate(String sql)`	Executes an SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as `int` data.
13	`getConnection()`	Returns the `Connection` object that created the `Statement` object.

No.	Method in the Statement interface	Function
14	`getFetchDirection()`	Acquires the default fetch direction for a result set that is created from this `Statement` object.
15	`getFetchSize()`	Acquires the default fetch size (number of retrieval result rows to be transferred from the HADB server to the HADB client in the batch mode) for a `ResultSet` object that is created from the `Statement` object.
16	`getHADBSQLSerialNum()`	Acquires the SQL statement sequence number that is assigned to this `Statement` object.
17	`getHADBStatementHandle()`	Acquires the statement handle that is assigned to this `Statement` object.
18	`getLargeMaxRows()`	Acquires the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object as `long` data.
19	`getLargeUpdateCount()`	Returns the number of updated rows as `long` data.
20	`getMaxFieldSize()`	Acquires the maximum number of bytes for a `CHAR` or `VARCHAR` column of a `ResultSet` object that is created by this `Statement` object.
21	`getMaxRows()`	Acquires the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object as `int` data.
22	`getMoreResults()`	Moves to the next result set.
23	`getQueryTimeout()`	Acquires the timeout time set for SQL processing in the `setQueryTimeout` method.
24	`getResultSet()`	Acquires retrieval results as a `ResultSet` object.
25	`getResultSetConcurrency()`	Acquires the concurrent processing mode for a `ResultSet` object that is created from this `Statement` object.
26	`getResultSetHoldability()`	Acquires the holdability of the `ResultSet` object that is created from this `Statement` object.
27	`getResultSetType()`	Acquires the result set type of a `ResultSet` object that is created from this `Statement` object.
28	`getUpdateCount()`	Returns the number of updated rows as `int` data.
29	`getWarnings()`	Acquires the first warning that is reported by a call related to this `Statement` object.
30	`isClosed()`	Acquires a value indicating whether this `Statement` object is closed.
31	`isCloseOnCompletion()`	Acquires a value that indicates whether this `Statement` object is closed when all result sets that depend on the object are closed.
32	`isPoolable()`	Acquires a value indicating whether this `Statement` object can be pooled.
33	`setCursorName(String name)`	Specifies the SQL cursor name to be used by the `execute` method of the next `Statement` object.
34	`setEscapeProcessing(boolean enable)`	Specifies whether escape syntax analysis by this `Statement` object is to be enabled or disabled.
35	`setFetchDirection(int direction)`	Specifies the fetch direction for a result set that is created from this `Statement` object.
36	`setFetchSize(int rows)`	Specifies the default fetch size (number of retrieval result rows to be transferred from the HADB server to the HADB client in the batch mode) for a `ResultSet` object that is created from this `Statement` object.

No.	Method in the Statement interface	Function
37	`setLargeMaxRows(long max)`	Sets the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as `long` data.
38	`setMaxFieldSize(int max)`	Specifies the maximum number of bytes for a `CHAR` or `VARCHAR` column in a `ResultSet` object that is created from this `Statement` object.
39	`setMaxRows(int max)`	Sets the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as `int` data.
40	`setQueryTimeout(int seconds)`	Specifies the SQL processing timeout value.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` might be thrown.

(3) Required package name and class name

The package and class names required in order to use the `Statement` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbStatement`

8.3.2 addBatch(String sql)

(1) Function

This method adds SQL statements to the `Statement` object's batch. You can add a maximum of 2,147,483,647 SQL statements.

(2) Format

```
public synchronized void addBatch(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statements to be added.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.

- The `Connection` object that created the `Statement` object is closed.
- An attempt was made to add more than 2,147,483,647 SQL statements.
- `null` or a character string with a length of zero is specified as the SQL statements.
- A specified SQL statement exceeds 16,000,000 characters.

8.3.3 cancel()

(1) Function

This method cancels the SQL statements executing in the corresponding object and in objects using the same connection as that object.

You can use this method to cancel executing SQL statements asynchronously.

(2) Format

```
public void cancel() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

(6) Notes

- If the corresponding `Statement` object is not executing any SQL statements, but another object is executing SQL statements on the same connection object, this method performs asynchronous cancellation.
- If the corresponding `Statement` object is not executing any SQL statements and no other object is executing SQL statements on the same connection object, this method does not perform cancellation processing.
- If asynchronous cancellation of SQL statements is successful, the transaction is rolled back.

8.3.4 clearBatch()

(1) Function

This method clears all SQL statements registered in this `Statement` object's batch.

(2) Format

```
public synchronized void clearBatch() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.5 clearWarnings()

(1) Function

This method clears all warnings that have been reported for this `Statement` object.

(2) Format

```
public synchronized void clearWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

None.

8.3.6 close()

(1) Function

This method closes the `Statement` object and any `ResultSet` object created from this `Statement` object.

(2) Format

```
public synchronized void close() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

If a database access error occurs, the JDBC driver throws an `SQLException`.

8.3.7 closeOnCompletion()

(1) Function

This method closes the `Statement` object when all result sets that depend on the `Statement` object have been closed. This method is invalid when running the `Statement` object does not generate a result set.

The effect on the `Statement` object is the same regardless of how many times this method is called.

Calling this method affects subsequent executions of `Statement` objects, and any `Statement` objects with dependent result sets that are currently open.

(2) Format

```
public synchronized void closeOnCompletion() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.8 execute(String sql)

(1) Function

This method executes an SQL statement. You can use the `getResultSet` and `getUpdateCount` methods (or the `getLargeUpdateCount` method) to acquire the `ResultSet` object and the number of updated rows.

The following table shows the return values of the `getResultSet` and `getUpdateCount` methods (or the `getLargeUpdateCount` method) depending on the type of the SQL statement that was executed.

Table 8-8: Return values of the getResultSet and getUpdateCount methods (or the `getLargeUpdateCount` method) depending on the type of the SQL statement that was executed

Type of the SQL statement that was executed	Return value of the getResultSet method	Return value of the getUpdateCount method or getLargeUpdateCount method
Retrieval SQL statement	`ResultSet` object obtained as the execution result	`-1`
Non-retrieval SQL statement	`null`	`0` or a greater value#
SQL execution resulting in an error	`null`	`-1`

#

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `getLargeUpdateCount` method instead of the `getUpdateCount` method. If you use the `getUpdateCount` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

Note

A retrieval SQL statement means a `SELECT` statement. Non-retrieval SQL statements include update SQL statements (such as the `UPDATE` statement) and definition SQL statements (such as `CREATE TABLE`).

An update SQL statement means `INSERT`, `UPDATE`, `DELETE`, `PURGE CHUNK`, and `TRUNCATE TABLE` statements.

(2) Format

```
public synchronized boolean execute(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statement that is to be executed.

(4) Return value

If a retrieval SQL statement was executed, this method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- `null` or a character string with a length of zero was specified in the `sql` argument.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

8.3.9 executeBatch()

(1) Function

This method executes the SQL statements registered in a batch and returns the number of updated rows as `int` data in an array.

The method clears all the SQL statements registered in the batch after executing all of them. If an error occurs during processing, the method still clears all the SQL statements registered in the batch.

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeBatch` method instead of the `executeBatch` method. If you use the `executeBatch` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized int[] executeBatch() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the number of updated rows for each SQL statement that was executed, as `int` data in an array. The elements of the array are in the order the SQL statements were registered into the batch. If no SQL statements are registered in the batch, or if the first SQL statement in the batch resulted in an error, the method returns an array containing no elements.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.

- The `Connection` object that created the `Statement` object is closed.

The method throws a `BatchUpdateException` (subclass of `SQLException`) in the following cases:

- A retrieval SQL statement was executed in the batch.
- A database access error occurs.

8.3.10 executeLargeBatch()

(1) Function

This method executes the SQL statements registered in a batch and returns an array of update counts in `long` format.

This method clears the SQL statements in the batch after executing all of them. All SQL statements will be cleared even if an error occurs during processing.

If there is a possibility that the update count might exceed `Integer.MAX_VALUE`, use the `executeLargeBatch` method instead of `executeBatch`. If you use the `executeBatch` method, it will return `0` if the update count exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized long[] executeLargeBatch() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns a `long` array of the numbers of updated rows for each of the executed SQL statements. The elements of the array are in the order in which the SQL statements are registered in the batch. If there are no SQL statements registered in the batch or the first SQL statement in the batch results in an error, the method returns an array containing zero elements.

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.3.9 executeBatch().

8.3.11 executeLargeUpdate(String sql)

(1) Function

This method executes an SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as a `long` value.

If it is possible that the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeUpdate(String sql)` method instead of `executeUpdate(String sql)`. If you use the `executeUpdate(String sql)` method, it will return `0` if the update count exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized long executeLargeUpdate(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statement (other than a retrieval SQL statement) to be executed.

(4) Return value

If an `INSERT`, `UPDATE`, or `DELETE` statement was executed, the method returns the number of updated rows as `long` data. If any other SQL statement was executed, the method returns `0`.

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.3.13 executeUpdate(String sql).

8.3.12 executeQuery(String sql)

(1) Function

This method executes a retrieval SQL statement and returns a `ResultSet` object containing the retrieval result.

(2) Format

```
public synchronized ResultSet executeQuery(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statement (retrieval SQL statement) that is to be executed.

(4) Return value

The method returns a `ResultSet` object containing the retrieval result.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

- A non-retrieval SQL statement (such as the `INSERT` statement) was executed.
- `null` or a character string with a length of zero was specified in the `sql` argument.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

8.3.13 executeUpdate(String sql)

(1) Function

This method executes a SQL statement (other than a retrieval SQL statement), and returns the number of updated rows as `int` data. If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeUpdate(String sql)` method instead of the `executeUpdate(String sql)` method. If you use the `executeUpdate(String sql)` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized int executeUpdate(String sql) throws SQLException
```

(3) Arguments

`String sql`

Specifies the SQL statement (non-retrieval SQL statement) that is to be executed.

(4) Return value

If an `INSERT`, `UPDATE`, or `DELETE` statement was executed, this method returns the number of updated rows as `int` data. If any other SQL statement was executed, the method returns `0`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A retrieval SQL statement (`SELECT` statement) was executed.
- `null` or a character string with a length of zero was specified in the `sql` argument.
- A database access error occurs.
- The specified SQL statement exceeds 16,000,000 characters.

8.3.14 getConnection()

(1) Function

This method returns the `Connection` object that created the `Statement` object.

(2) Format

```
public synchronized Connection getConnection() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `Connection` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.15 getFetchDirection()

(1) Function

This method acquires the default fetch direction for a result set that is created from this `Statement` object.

(2) Format

```
public synchronized int getFetchDirection() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `ResultSet.FETCH_FORWARD`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.

- The `Connection` object that created the `Statement` object is closed.

8.3.16 getFetchSize()

(1) Function

This method acquires the default fetch size (number of retrieval result rows to be transferred from the HADB server to the HADB client in the batch mode) for a `ResultSet` object that is created from the `Statement` object.

(2) Format

```
public synchronized int getFetchSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the default fetch size (number of retrieval result rows to be transferred from the HADB server to the HADB client in the batch mode) for a `ResultSet` object that is created from this `Statement` object.

If `0` is specified in the `setFetchSize` method, the value of `adb_clt_fetch_size` in the system properties, user properties, or URL connection properties is applied as the fetch size, but the return value is `0`. The following table shows the relationship between the fetch size and the return value.

Table 8-9: Relationship between the fetch size and the return value

Setting by the setFetchSize method (m)	Return value
`0`	`0`
$1 \leq m \leq 65,535$	m

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.17 getHADBSQLSerialNum()

(1) Function

This method acquires the SQL statement sequence number that is assigned to this `Statement` object.

(2) Format

```
public long getHADBSQLSerialNum() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the SQL statement sequence number that is assigned to this `Statement` object.

This method returns `0` if executed before an SQL statement is executed.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

(6) Notes

This is an HADB-specific method provided by the `AdbStatement` interface. For details about the execution method, see 12.2 Wrapper interface.

8.3.18 getHADBStatementHandle()

(1) Function

This method acquires the statement handle that is assigned to this `Statement` object.

(2) Format

```
public int getHADBStatementHandle() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the statement handle that is assigned to this `Statement` object.

This method returns `0` if executed before an SQL statement is executed.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

(6) Notes

This is an HADB-specific method provided by the `AdbStatement` interface. For details about the execution method, see 12.2 Wrapper interface.

8.3.19 getLargeMaxRows()

(1) Function

This method specifies the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as a `long` value. Any rows in excess of this value are not stored in the `ResultSet` object. You will not be notified that these rows have not been stored.

If you specify a value in the `setLargeMaxRows` method that exceeds `Integer.MAX_VALUE`, use the `getLargeMaxRows` method instead of `getMaxRows`. When you use the `getMaxRows` method, if the value of `Integer.MAX_VALUE` is exceeded, `0` is returned.

(2) Format

```
public synchronized long getLargeMaxRows() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as a `long` value. The value it returns is the value set by the `setMaxRows` or the `setLargeMaxRows` method. A return value of `0` means that a maximum number of rows has not been set.

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.3.22 getMaxRows().

8.3.20 getLargeUpdateCount()

(1) Function

This method returns the number of updated rows as a `long` value.

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `getLargeUpdateCount` method instead of `getUpdateCount`. If you use the `getUpdateCount` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized long getLargeUpdateCount() throws SQLException
```

(3) Arguments

None.

(4) Return value

For details about return values, see (4) Return value in 8.3.29 getUpdateCount().

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.3.29 getUpdateCount().

8.3.21 getMaxFieldSize()

(1) Function

This method acquires the maximum number of bytes for a `CHAR` or `VARCHAR` column of a `ResultSet` object that is created by this `Statement` object. Any bytes in excess of this value are discarded.

(2) Format

```
public synchronized int getMaxFieldSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the maximum number of bytes for a `CHAR` or `VARCHAR` column. The method returns the value set by the `setMaxFieldSize` method. A value of `0` means that no maximum number of bytes has been set.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.22 getMaxRows()

(1) Function

This method acquires the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as `int` data. Any rows in excess of this value are not stored in the `ResultSet` object (and without notification).

If the value specified in the `setLargeMaxRows` method exceeds `Integer.MAX_VALUE`, use the `getLargeMaxRows` method instead of the `getMaxRows` method. If you use the `getMaxRows` method, it will return `0` if the number of rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized int getMaxRows() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as `int` data. The value set in the `setMaxRows` or `setLargeMaxRows` method is returned. A value of `0` means that no maximum number of rows has been set.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.23 getMoreResults()

(1) Function

This method moves to the next result set.

(2) Format

```
public synchronized boolean getMoreResults() throws SQLException
```

(3) Arguments

None.

(4) Return value

If there is another result set, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.24 getQueryTimeout()

(1) Function

This method acquires the timeout time set for SQL processing in the `setQueryTimeout` method.

(2) Format

```
public synchronized int getQueryTimeout() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the timeout value (in seconds) that was set by the `setQueryTimeout` method. If the `setQueryTimeout` method has not been executed, the method returns `0`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.25 getResultSet()

(1) Function

This method acquires retrieval results as a `ResultSet` object.

(2) Format

```
public synchronized ResultSet getResultSet() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the `ResultSet` object held by the `Statement` object. If there are no retrieval results in the `ResultSet` object, the method returns `null`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.26 getResultSetConcurrency()

(1) Function

This method acquires the concurrent processing mode for a `ResultSet` object that is created from this `Statement` object.

(2) Format

```
public synchronized int getResultSetConcurrency() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `ResultSet.CONCUR_READ_ONLY`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.27 getResultSetHoldability()

(1) Function

This method acquires the holdability of the `ResultSet` object that is created from this `Statement` object.

(2) Format

```
public synchronized int getResultSetHoldability() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.28 getResultSetType()

(1) Function

This method acquires the result set type of a `ResultSet` object that is created from this `Statement` object.

(2) Format

```
public synchronized int getResultSetType() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns `ResultSet.TYPE_FORWARD_ONLY` or `ResultSet.TYPE_SCROLL_INSENSITIVE`.

`ResultSet.TYPE_FORWARD_ONLY`
The only direction the cursor can move is forward.

`ResultSet.TYPE_SCROLL_INSENSITIVE`
The cursor can be scrolled, but changes to the underlying values are not reflected in the result set.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.

- The `Connection` object that created the `Statement` object is closed.

8.3.29 getUpdateCount()

(1) Function

This method returns the number of updated rows as `int` data.

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `getLargeUpdateCount` method instead of the `getUpdateCount`method. If you use the `getUpdateCount` method, `0` is returned if `Integer.MAX_VALUE` is exceeded.

(2) Format

```
public synchronized int getUpdateCount() throws SQLException
```

(3) Arguments

None.

(4) Return value

The following table describes the details of the return value.

Table 8-10: Details of the return values of getUpdateCount and getLargeUpdateCount methods

Statement object's method execution status				Return value of getUpdateCount or getLargeUpdateCount method
No `execute`*XXX* method has been executed.				`-1`
An `execute`*XXX* method has been executed.	A `getMoreResults` method was executed after the last `execute`*XXX* method was executed.			`-1`
	The last `execute`*XXX* method executed resulted in an error.			`-1`
	The last method to be executed was an `executeBatch` or `executeLargeBatch` method			`-1`
	The last method to be executed was an `execute`*XXX* method other than an `executeBatch` method or `executeLargeBatch` method	A retrieval SQL statement was executed at the end.		`-1`
		A non-retrieval SQL statement was executed at the end.	`INSERT`, `UPDATE`, `DELETE`	Number of updated rows#
			Other	`0`

#

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `getLargeUpdateCount` method instead of the `getUpdateCount` method. If you use the `getUpdateCount` method, it returns `0` if `Integer.MAX_VALUE` is exceeded.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.30 getWarnings()

(1) Function

This method acquires the first warning reported by a call related to this `Statement` object. If there is more than one warning, the subsequent warnings are chained to the first warning and can be acquired by calling the `getNextWarning` method of the `SQLWarning` object for the immediately preceding warning that was acquired.

(2) Format

```
public synchronized SQLWarning getWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the first `SQLWarning` object. If there is no `SQLWarning` object, the method returns `null`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.31 isClosed()

(1) Function

This method acquires a value indicating whether this `Statement` object is closed.

(2) Format

```
public synchronized boolean isClosed() throws SQLException
```

(3) Arguments

None.

(4) Return value

If this `Statement` object is closed, the method returns `true`; if it is not closed, the method returns `false`.

(5) Exceptions

None.

8.3.32 isCloseOnCompletion()

(1) Function

This method acquires a value that indicates whether the `Statement` object will be closed when all result sets that depend on the `Statement` have been closed.

(2) Format

```
public synchronized boolean isCloseOnCompletion() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns `true` when the `Statement` object is closed after all result sets that depend on the `Statement` object have been closed. It returns `false` if the object is not closed.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.33 isPoolable()

(1) Function

This method acquires a value indicating whether this `Statement` object can be pooled.

(2) Format

```
public synchronized boolean isPoolable() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.34 setCursorName(String name)

(1) Function

This method specifies the SQL cursor name to be used by the `execute` method of the next `Statement` object.

(2) Format

```
public synchronized void setCursorName(String name) throws SQLException
```

(3) Arguments

`String` `name`
Specifies an SQL cursor name.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.35 setEscapeProcessing(boolean enable)

(1) Function

This method specifies whether escape syntax analysis by this `Statement` object is to be enabled or disabled.

(2) Format

```
public synchronized void setEscapeProcessing(boolean enable) throws SQLException
```

(3) Arguments

`boolean enable`

Specifies `true` to enable escape syntax analysis and `false` to disable it.

If this method is not executed, `true` is assumed.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.

8.3.36 setFetchDirection(int direction)

(1) Function

This method specifies the fetch direction for a result set that is created from this `Statement` object.

(2) Format

```
public synchronized void setFetchDirection(int direction) throws SQLException
```

(3) Arguments

`int direction`

Specifies the fetch direction. Only `ResultSet.FETCH_FORWARD` can be specified.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A value other than `ResultSet.FETCH_FORWARD` was specified for `direction`.

8.3.37 setFetchSize(int rows)

(1) Function

This method specifies the default fetch size (number of retrieval result rows to be transferred from the HADB server to the HADB client in the batch mode) for a `ResultSet` object that is created from this `Statement` object.

(2) Format

```
public synchronized void setFetchSize(int rows) throws SQLException
```

(3) Arguments

`int rows`

Specifies the number of rows to be transferred in the batch mode, in the range from 0 to 65,535.

If the specified value is `1` or greater, the JDBC driver transfers the specified number of rows of data from the HADB server to the HADB client in the batch mode.

If `0` is specified or this method is not executed, the value of `adb_clt_fetch_size` in the system properties, user properties, or URL connection properties is applied.

The following table shows the relationship between the `setFetchSize` method setting and the `adb_clt_fetch_size` property setting.

Table 8-11: Relationship between the setFetchSize method setting and the adb_clt_fetch_size property setting

setFetchSize method setting (m)	adb_clt_fetch_size property setting (n)	Number of rows to be transferred in the batch mode
0	$1 \leq n \leq 65,535$	*n*
	Not specified	1
$1 \leq m \leq 65,535$	$1 \leq n \leq 65,535$	*m*
	Not specified	*m*

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A value outside the range from 0 to 65,535 was specified in `rows`.
- The value specified in `rows` is greater than the maximum number of rows that can be stored (`setMaxRows` method setting).
- The value specified in `rows` is greater than the maximum number of rows that can be stored (`setLargeMaxRows` method setting).

(6) Notes

The following table shows the priority order for determining the number of rows that the JDBC driver requests the HADB server to transfer in a single transmission.

Table 8-12: Priorities for number of rows that the JDBC driver requests the HADB server to transfer in one transmission

Priority	Number of rows that the JDBC driver requests the HADB server to transfer in one transmission
1	Value specified in the argument of the `setFetchSize` method of the `ResultSet` class
2	Value specified in the argument of the `setFetchSize` method of the `Statement` class
3	Value specified in the `adb_clt_fetch_size` system property
4	`adb_clt_fetch_size` property value specified in the `info` argument of the `getConnection` method of the `DriverManager` class
5	Value of `adb_clt_fetch_size` specified in the `url` argument of the `getConnection` method of the `DriverManager` class

If the retrieval result is larger than the number of transfer rows shown in the table above, the JDBC driver requests transfer to the HADB server as many times as necessary until retrieval is completed (or until all retrieval requests from the application program have been processed).

8.3.38 setLargeMaxRows(long max)

(1) Function

This method sets the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as a `long` value. Any rows in excess of this value are not stored in the `ResultSet` object. You will not be notified that these rows have not been stored.

The setting you specify in this method does not apply to `ResultSet` objects that have already been created.

(2) Format

```
public synchronized void setLargeMaxRows(long max) throws SQLException
```

(3) Arguments

`long max`

Specifies the maximum number of rows that can be stored.

If you specify `0`, no maximum is set. If the result set type is `ResultSet.TYPE_SCROLL_INSENSITIVE`, the maximum number of rows that can be stored is set to `Integer.MAX_VALUE` if you specify `0`.

If you do not execute this method, HADB operates as if `0` were specified (no maximum is set).

(4) Return value

None.

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.3.40 setMaxRows(int max).

8.3.39 setMaxFieldSize(int max)

(1) Function

This method specifies the maximum number of bytes for a `CHAR` or `VARCHAR` column in a `ResultSet` object that is created from this `Statement` object. Any bytes in excess of this value are discarded.

The setting specified by this method is not applied to `ResultSet` objects that have already been created.

(2) Format

```
public synchronized void setMaxFieldSize(int max) throws SQLException
```

(3) Arguments

`int max`

Specifies the maximum number of bytes to be applied to each `CHAR` and `VARCHAR` column.

If `0` is specified, no maximum number of bytes is set.

If this method is not executed, `0` (no maximum number of bytes is set) is assumed.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A value less than `0` is specified for `max`.

8.3.40 setMaxRows(int max)

(1) Function

This method sets the maximum number of rows that can be stored in a `ResultSet` object created from this `Statement` object, as `int` data. Any rows in excess of this value are not stored in the `ResultSet` object (and without notification).

The setting specified by this method is not applied to `ResultSet` objects that have already been created.

(2) Format

```
public synchronized void setMaxRows(int max) throws SQLException
```

(3) Arguments

`int max`

Specifies the maximum number of rows that can be stored.

If `0` is specified, no maximum number of rows that can be stored is set. If the result set type is `ResultSet.TYPE_SCROLL_INSENSITIVE`, the maximum number of rows that can be stored is set to `Integer.MAX_VALUE` even if `0` is specified here.

If this method is not executed, `0` (no maximum number of rows that can be stored is set) is assumed.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A value less than `0` is specified for `max`.

8.3.41 setQueryTimeout(int seconds)

(1) Function

This method specifies the SQL processing timeout value.

(2) Format

```
public synchronized void setQueryTimeout(int seconds) throws SQLException
```

(3) Arguments

`int seconds`

Specifies an SQL processing timeout value (in seconds), in the range from 0 to 65,535.

If `0` is specified or this method is not executed, the value of `adb_clt_rpc_sql_wait_time` in the system properties, user properties, or URL connection properties takes effect.

If this method is executed, HADB monitors the following wait times:

- How long the HADB client waits for the HADB server to respond to a processing request
  If this wait time is exceeded, a timeout error whose `SQLCODE` is `-732` (`KFAA30732-E`) is returned to the application. When this occurs, processing of the SQL statement is canceled, and the transaction is rolled back. Then, the application is disconnected from the HADB server.
- How long to wait to secure processing real threads if a shortage occurs when multiple `SELECT` statements are executed concurrently in the same connection
  If this wait time is exceeded, HADB returns a timeout error whose `SQLCODE` is `-1071570` (`KFAA71570-E`) to the application. When this happens, processing of the SQL statement is canceled but the transaction is not rolled back. Nor is the application disconnected from the HADB server.

For details about the purpose of monitoring wait times using this method, see (4) Note about executing multiple SELECT statements concurrently in the same connection in 7.4.1 How to retrieve data.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Statement` object is closed.
- The `Connection` object that created the `Statement` object is closed.
- A value less than `0` is specified for `seconds`.

(6) Notes

If 65,536 (maximum value of `adb_clt_rpc_sql_wait_time` property) or a greater value is specified for `seconds`, the specification of this method is ignored.

8.3.42 Notes about the Statement interface

(1) Notes about executing executeXXX methods

If you execute an `execute`*XXX* method before the `ResultSet` object created by the corresponding `Statement` object has been closed, the JDBC driver closes the previous `ResultSet` object that was created. If an attempt is made to use the previously created `ResultSet` object to acquire retrieval results after the `execute`*XXX* method has been executed, the JDBC driver will throw an `SQLException`. The following shows an example that results in an `SQLException`.

■ **Example that results in an `SQLException`**

```
Statement st  = con.createStatement();
ResultSet rs1 = st.executeQuery("select * from tb1");
ResultSet rs2 = st.executeQuery("select * from tb2");
rs1.next();  // SQLException is thrown.
rs2.next();
```

(2) Closing the Statement object

After you have used a `Statement` object, make sure that you close the `Statement` object explicitly with the `close` method. When a `Statement` object is closed explicitly, the corresponding statement handle in HADB is released. If you do not close `Statement` objects, a shortage of statement handles might occur.

The statement handle is also released when a transaction is settled by issuing `COMMIT` or `ROLLBACK`. Therefore, if you settle transactions at appropriate intervals, you can prevent a shortage of statement handles.

8.4 PreparedStatement interface

This section explains the methods provided by the `PreparedStatement` interface.

8.4.1 List of the methods in the PreparedStatement interface

(1) Main functions of the PreparedStatement interface

The `PreparedStatement` interface provides the following main functions:

- Execution of SQL statements in which dynamic parameters are specified
- Specification of dynamic parameters
- 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 in the PreparedStatement interface that are supported by HADB

The following table lists and describes the methods in the `PreparedStatement` interface that are supported by HADB.

Table 8-13: Methods in the PreparedStatement interface

No.	Method in the PreparedStatement interface	Function
1	`addBatch()`	Adds the current parameter set to this `PreparedStatement` object's batch.
2	`clearParameters()`	Clears all values from the current parameter set that is specified.
3	`execute()`	Executes the preprocessed SQL statement.
4	`executeLargeUpdate()`	Executes a preprocessed SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as `long` data.
5	`executeQuery()`	Executes a preprocessed retrieval SQL statement and returns the `ResultSet` object that contains the execution results.
6	`executeUpdate()`	Executes a preprocessed SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as `int` data.
7	`getHADBSQLSerialNum()`	Acquires the SQL statement sequence number that is assigned to this `PreparedStatement` object.
8	`getHADBStatementHandle()`	Acquires the statement handle that is assigned to this `PreparedStatement` object.
9	`getMetaData()`	Returns the `ResultSetMetaData` object that stores information about the columns in the `ResultSet` object that is returned when this `PreparedStatement` object is executed.

No.	Method in the PreparedStatement interface	Function
10	`getParameterMetaData()`	Returns the `ParameterMetaData` object that contains meta information for the parameters in this `PreparedStatement` object.
11	`setAsciiStream(int parameterIndex, InputStream x, int length)`	Sets the value of a specified `InputStream` object as a dynamic parameter value.
12	`setBigDecimal(int parameterIndex, BigDecimal x)`	Sets a specified `BigDecimal` object as a dynamic parameter value.
13	`setBinaryStream(int parameterIndex, InputStream x, int length)`	Sets the value of a specified `InputStream` object as a dynamic parameter value.
14	`setBoolean(int parameterIndex, boolean x)`	Sets a specified `boolean` value as a dynamic parameter value.
15	`setByte(int parameterIndex, byte x)`	Sets a specified `byte` value as a dynamic parameter value.
16	`setBytes(int parameterIndex, byte[] x)`	Sets a specified `byte` array as a dynamic parameter value.
17	`setCharacterStream(int parameterIndex, Reader reader, int length)`	Sets a specified `Reader` object as a dynamic parameter value.
18	`setDate(int parameterIndex, Date x)`	Sets a specified `java.sql.Date` object as a dynamic parameter value.
19	`setDate(int parameterIndex, Date x, Calendar cal)`	Converts a `java.sql.Date` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.
20	`setDouble(int parameterIndex, double x)`	Sets a specified `double` value as a dynamic parameter value.
21	`setFloat(int parameterIndex, float x)`	Sets a specified `float` value as a dynamic parameter value.
22	`setInt(int parameterIndex, int x)`	Sets a specified `int` value as a dynamic parameter value.
23	`setLong(int parameterIndex, long x)`	Sets a specified `long` value as a dynamic parameter value.
24	`setNull(int parameterIndex,int sqlType)`	Sets the null value in a specified dynamic parameter.
25	`setObject(int parameterIndex, Object x)`	Sets the value of a specified object as a dynamic parameter value.
26	`setObject(int parameterIndex, Object x, int targetSqlType)`	
27	`setObject(int parameterIndex, Object x, int targetSqlType, int scale)`	
28	`setShort(int parameterIndex, short x)`	Sets a specified `short` value as a dynamic parameter value.
29	`setString(int parameterIndex, String x)`	Sets a specified `String` object as a dynamic parameter value.
30	`setTime(int parameterIndex, Time x)`	Sets a specified `java.sql.Time` object as a dynamic parameter value.
31	`setTime(int parameterIndex, Time x, Calendar cal)`	Converts a `java.sql.Time` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.
32	`setTimestamp(int parameterIndex, Timestamp x)`	Sets a specified `java.sql.Timestamp` object as a dynamic parameter value.
33	`setTimestamp(int parameterIndex, Timestamp x, Calendar cal)`	Converts a `java.sql.Timestamp` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` is thrown.

(3) Required package name and class name

The package and class names required in order to use the `PreparedStatement` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbPreparedStatement`

8.4.2 addBatch()

(1) Function

This method adds the current parameter set to this `PreparedStatement` object's batch. You can add a maximum of 2,147,483,647 parameter sets.

(2) Format

```
public synchronized void addBatch() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- No value is set for at least one dynamic parameter.
- More than 2,147,483,647 items have been registered in the batch.

8.4.3 clearParameters()

(1) Function

This method clears all values from the current parameter set that is specified.

(2) Format

```
public synchronized void clearParameters() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.

8.4.4 execute()

(1) Function

This method executes the preprocessed SQL statement.

You can use the `getResultSet` and `getUpdateCount` methods (or the `getLargeUpdateCount` method) of the `PreparedStatement` object to obtain the `ResultSet` object and the number of updated rows as execution results.

For the return values of the `getResultSet` method and `getUpdateCount` method (or `getLargeUpdateCount` method) after execution of the `execute` method, see Table 8-8: Return values of the getResultSet and getUpdateCount methods (or the `getLargeUpdateCount` method) depending on the type of the SQL statement that was executed.

(2) Format

```
public synchronized boolean execute() throws SQLException
```

(3) Arguments

None.

(4) Return value

If a retrieval SQL statement was executed, this method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- No value is set in at least one dynamic parameter.
- A database access error occurs.

8.4.5 executeLargeUpdate()

(1) Function

This method executes an SQL statement (other than a retrieval SQL statement) that has undergone preprocessing, and returns the number of updated rows as a `long` value.

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeUpdate()` method instead of `executeUpdate()`. If you use the `executeUpdate()` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized long executeLargeUpdate() throws SQLException
```

(3) Arguments

None.

(4) Return value

If an `INSERT`, `UPDATE`, or `DELETE` statement was executed, the method returns the number of updated rows as `long` data. If any other SQL statement was executed, the method returns `0`.

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.4.7 executeUpdate().

8.4.6 executeQuery()

(1) Function

This method executes the preprocessed retrieval SQL statement and returns the `ResultSet` object that contains the execution results.

(2) Format

```
public synchronized ResultSet executeQuery() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the `ResultSet` object that contains the execution results.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A non-retrieval SQL statement (such as an `INSERT` statement) was executed.
- No value is set in at least one dynamic parameter.
- A database access error occurs.

8.4.7 executeUpdate()

(1) Function

This method executes a preprocessed SQL statement (other than a retrieval SQL statement) and returns the number of updated rows as `int` data.

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeUpdate()` method instead of the `executeUpdate()` method. If you use the `executeUpdate()` method, it will return `0` if the number of updated rows exceeds `Integer.MAX_VALUE`.

(2) Format

```
public synchronized int executeUpdate() throws SQLException
```

(3) Arguments

None.

(4) Return value

If an `INSERT`, `UPDATE`, or `DELETE` statement was executed, the method returns the number of updated rows as `int` data. If any other SQL statement was executed, the method returns `0`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this object is closed.

- A retrieval SQL statement was executed.
- No value is set in at least one dynamic parameter.
- A database access error occurs.

8.4.8 getHADBSQLSerialNum()

(1) Function

This method acquires the SQL statement sequence number that is assigned to this `PreparedStatement` object.

(2) Format

```
public long getHADBSQLSerialNum() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the SQL statement sequence number that is assigned to this `PreparedStatement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.

(6) Notes

This is an HADB-specific method provided by the `AdbPreparedStatement` interface. For details about the execution method, see 12.2 Wrapper interface.

8.4.9 getHADBStatementHandle()

(1) Function

This method acquires the statement handle that is assigned to this `PreparedStatement` object.

(2) Format

```
public int getHADBStatementHandle() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the statement handle that is assigned to this `PreparedStatement` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.

(6) Notes

This is an HADB-specific method provided by the `AdbPreparedStatement` interface. For details about the execution method, see 12.2 Wrapper interface.

8.4.10 getMetaData()

(1) Function

This method returns the `ResultSetMetaData` object that stores information about the columns in the `ResultSet` object that is returned when this `PreparedStatement` object is executed.

(2) Format

```
public synchronized ResultSetMetaData getMetaData() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns meta information for this `PreparedStatement` object as a `ResultSetMetaData` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.

8.4.11 getParameterMetaData()

(1) Function

This method returns the `ParameterMetaData` object that contains meta information for the parameters in this `PreparedStatement` object. The returned `ParameterMetaData` object is meta information for parameters acquired from the server when the `Connection.prepareStatement()` method is executed.

(2) Format

```
public synchronized ParameterMetaData getParameterMetaData() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns meta information for this `PreparedStatement` object as a `ParametertMetaData` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.

8.4.12 setAsciiStream(int parameterIndex, InputStream x, int length)

(1) Function

This method sets the value of a specified `InputStream` object as a dynamic parameter value.

(2) Format

```
public synchronized void setAsciiStream(int parameterIndex, InputStream x, int length
) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`InputStream x`

Specifies the `java.io.InputStream` object that contains the value to be set in the specified dynamic parameter.

`int length`

Specifies the number of bytes to be set.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A value less than `0` was specified for `length`.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

(6) Notes

The `setAsciiStream` method does not execute the `close` method on `x`, even after input from `x` has been completed.

8.4.13 setBigDecimal(int parameterIndex, BigDecimal x)

(1) Function

This method sets a specified `BigDecimal` object as a dynamic parameter value.

(2) Format

```
public synchronized void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLEx
ception
```

(3) Arguments

`int parameterIndex`
Specifies the number of a dynamic parameter.

`BigDecimal x`
Specifies the `java.math.BigDecimal` object that is to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.

- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.14 setBinaryStream(int parameterIndex, InputStream x, int length)

(1) Function

Sets the value of a specified `InputStream` object as a dynamic parameter value.

(2) Format

```
public synchronized void setBinaryStream(int parameterIndex, InputStream x, int lengt
h) throws SQLException
```

(3) Arguments

`int parameterIndex`
Specifies the number of a dynamic parameter.

`InputStream x`
Specifies the `java.io.InputStream` object that contains the value to be set in the specified dynamic parameter.

`int length`
Specifies the number of bytes to be set.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A value less than `0` was specified for `length`.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

(6) Notes

The `setBinaryStream` method does not execute the close method on `x`, even after input from `x` has been completed.

8.4.15 setBoolean(int parameterIndex, boolean x)

(1) Function

This method sets a specified `boolean` value as a dynamic parameter value.

(2) Format

```
public synchronized void setBoolean(int parameterIndex, boolean x) throws SQLExceptio
n
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`boolean x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.

8.4.16 setByte(int parameterIndex, byte x)

(1) Function

This method sets a specified `byte` value as a dynamic parameter value.

(2) Format

```
public synchronized void setByte(int parameterIndex, byte x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`byte x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.

8.4.17 setBytes(int parameterIndex, byte[] x)

(1) Function

This method sets a specified `byte` array as a dynamic parameter value.

(2) Format

```
public synchronized void setBytes(int parameterIndex, byte[] x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`byte[] x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.

- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.18 setCharacterStream(int parameterIndex, Reader reader, int length)

(1) Function

This method sets a specified `Reader` object as a dynamic parameter value.

(2) Format

```
public synchronized void setCharacterStream(int parameterIndex, Reader reader, int le
ngth) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Reader reader`

Specifies the `java.io.Reader` object that contains the value to be set in the specified dynamic parameter.

`int length`

Specifies the number of characters.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A value less than `0` was specified for `length`.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- Encoding failed.

8.4.19 setDate(int parameterIndex, Date x)

(1) Function

This method sets a specified `java.sql.Date` object as a dynamic parameter value.

(2) Format

```
public synchronized void setDate(int parameterIndex, Date x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Date x`

Specifies the `java.sql.Date` object that contains the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.20 setDate(int parameterIndex, Date x, Calendar cal)

(1) Function

This method converts a `java.sql.Date` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.

(2) Format

```
public synchronized void setDate(int parameterIndex, Date x, Calendar cal) throws SQL
Exception
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Date x`

Specifies the `java.sql.Date` object that contains the value to be set in the specified dynamic parameter.

`Calendar cal`

Specifies the calendar in which has been set the time zone for the value to be stored in the database. If `null` is specified, the Java Virtual Machine's (JVM) default time zone's calendar is applied.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.21 setDouble(int parameterIndex, double x)

(1) Function

This method sets a specified `double` value as a dynamic parameter value.

(2) Format

```
public synchronized void setDouble(int parameterIndex, double x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`double x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.

- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.22 setFloat(int parameterIndex, float x)

(1) Function

This method sets a specified `float` value as a dynamic parameter value.

(2) Format

```
public synchronized void setFloat(int parameterIndex, float x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`float x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.23 setInt(int parameterIndex, int x)

(1) Function

This method sets a specified `int` value as a dynamic parameter value.

(2) Format

```
public synchronized void setInt(int parameterIndex, int x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`int x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.24 setLong(int parameterIndex, long x)

(1) Function

This method sets a specified `long` value as a dynamic parameter value.

(2) Format

```
public synchronized void setLong(int parameterIndex, long x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`long x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.25 setNull(int parameterIndex,int sqlType)

(1) Function

This method sets the null value in a specified dynamic parameter.

(2) Format

```
public synchronized void setNull(int parameterIndex,int sqlType) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`int sqlType`

Specifies JDBC's SQL data type.

This argument is ignored, if specified.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.

8.4.26 setObject(int parameterIndex, Object x)

(1) Function

This method sets the value of a specified object as a dynamic parameter value.

(2) Format

```
public synchronized void setObject(int parameterIndex, Object x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Object x`

Specifies the object that contains the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.27 setObject(int parameterIndex, Object x, int targetSqlType)

(1) Function

This method sets the value of a specified object as a dynamic parameter value.

(2) Format

```
public synchronized void setObject(int parameterIndex, Object x, int targetSqlType) t
hrows SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Object x`

Specifies the object that contains the value to be set in the specified dynamic parameter.

`int targetSqlType`

Specifies JDBC's SQL data type.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- One of the following was set for `targetSqlType`:
  `Types.ARRAY`, `Types.BLOB`, `Types.CLOB`, `Types.JAVA_OBJECT`, `Types.REF`, or `Types.STRUCT`

8.4.28 setObject(int parameterIndex, Object x, int targetSqlType, int scale)

(1) Function

This method sets the value of a specified object as a dynamic parameter value.

(2) Format

```
public synchronized void setObject(int parameterIndex, Object x, int targetSqlType, i
nt scale) throws SQLException
```

(3) Arguments

`int parameterIndex`
Specifies the number of a dynamic parameter.

`Object x`
Specifies the object that contains the value to be set in the specified dynamic parameter.

`int targetSqlType`
Specifies JDBC's SQL data type.

`int scale`
Specifies scaling. This argument is ignored, if specified.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- One of the following was set for `targetSqlType`:
  `Types.ARRAY`, `Types.BLOB`, `Types.CLOB`, `Types.JAVA_OBJECT`, `Types.REF`, or `Types.STRUCT`

8.4.29 setShort(int parameterIndex, short x)

(1) Function

This method sets a specified `short` value as a dynamic parameter value.

(2) Format

```
public synchronized void setShort(int parameterIndex, short x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`short x`

Specifies the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.30 setString(int parameterIndex, String x)

(1) Function

This method sets a specified `String` object as a dynamic parameter value.

(2) Format

```
public synchronized void setString(int parameterIndex, String x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`String x`

Specifies the `String` object that contains the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.
- Encoding failed.

8.4.31 setTime(int parameterIndex, Time x)

(1) Function

This method sets a specified `java.sql.Time` object as a dynamic parameter value.

(2) Format

```
public synchronized void setTime(int parameterIndex, Time x) throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Time x`

Specifies the `java.sql.Time` object that contains the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.32 setTime(int parameterIndex, Time x, Calendar cal)

(1) Function

This method converts a `java.sql.Time` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.

(2) Format

```
public synchronized void setTime(int parameterIndex, Time x, Calendar cal) throws SQL
Exception
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Time x`

Specifies the `java.sql.Time` object that contains the value to be set in the specified dynamic parameter.

`Calendar cal`

Specifies the calendar in which has been set the time zone for the value to be stored in the database. If `null` is specified, the Java Virtual Machine's (JVM) default time zone's calendar is applied.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.33 setTimestamp(int parameterIndex, Timestamp x)

(1) Function

This method sets a specified `java.sql.Timestamp` object as a dynamic parameter value.

(2) Format

```
public synchronized void setTimestamp(int parameterIndex, Timestamp x) throws SQLExce
ption
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Timestamp x`

Specifies the `java.sql.Timestamp` object that contains the value to be set in the specified dynamic parameter.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.34 setTimestamp(int parameterIndex, Timestamp x, Calendar cal)

(1) Function

This method converts a `java.sql.Timestamp` object specified in local time to the equivalent value in a specified calendar's time zone, and then sets the resulting value as a dynamic parameter value.

(2) Format

```
public synchronized void setTimestamp(int parameterIndex, Timestamp x, Calendar cal)
throws SQLException
```

(3) Arguments

`int parameterIndex`

Specifies the number of a dynamic parameter.

`Timestamp x`

Specifies the `java.sql.Timestamp` object that contains the value to be set in the specified dynamic parameter.

`Calendar cal`

Specifies the calendar in which has been set the time zone for the value to be stored in the database. If `null` is specified, the Java Virtual Machine's (JVM) default time zone's calendar is applied.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `PreparedStatement` object is closed.
- The `Connection` object that created this `PreparedStatement` object is closed.
- A nonexistent dynamic parameter number was specified.
- This method does not support the HADB data type specified in the dynamic parameter.
- The specified value is outside the range of data types for the column or in a format that cannot be converted.

8.4.35 Notes about the PreparedStatement interface

The `PreparedStatement` interface is a subinterface of the `Statement` interface. For this reason, the notes for the `Statement` interface also apply to the `PreparedStatement` interface.

This section describes additional notes that apply to the `PreparedStatement` interface.

(1) Dynamic parameter setup

- For details about whether mapping is possible with a `set`*XXX* method, see (3) Mapping when a dynamic parameter is specified in 7.6.1 Mapping data types.
- If the column number specified in a `set`*XXX* method does not exist, the JDBC driver throws an `SQLException`.
- Overflow occurs when the value specified in a `set`*XXX* method exceeds the value range that can be represented by the data type of the corresponding dynamic parameter, resulting in an `SQLException`. For the combinations of a `set`*XXX* method and HADB data type that can cause overflow to occur, see 7.6.3 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 dynamic parameters to be specified are the same.
  - The `close` method is executed for the target `PreparedStatement` object.

(2) Value specifications for dynamic parameters of HADB's DECIMAL type

This subsection describes the operations that are executed when a `set`*XXX* method is used to specify a value for a dynamic parameter of HADB's `DECIMAL` type, and when the precision and scaling of the dynamic parameter do not match the precision and scaling of the specified value.

- When the precision of the specified value is greater than the actual precision: the JDBC driver throws an `SQLException`.
- When the precision of the specified value is smaller than the actual precision: the JDBC driver increases the precision of the specified value.
- When the scaling of the specified value is greater than the actual scaling: the JDBC driver truncates the specified scaling.
- When the scaling of the specified value is smaller than the actual scaling: the JDBC driver increases the specified scaling by adding zeros.

(3) Value specifications for dynamic parameters of HADB's TIME and TIMESTAMP types

If a data type with a high fractional second precision is specified for a data type with a low fractional second precision, the differential fractional second precision is discarded. On the other hand, if a data type with a low fractional second precision is specified for a data type with a high fractional second precision, the resulting value is extended with zeros padded for the differential fractional second precision.

(4) Value specifications for dynamic parameters of HADB's CHAR and VARCHAR types

When a `set`*XXX* method is used to specify a value for a dynamic parameter of HADB's `CHAR` or `VARCHAR` type, and when the length of the value after conversion to a character string representation is greater than the defined length of the dynamic parameter, the JDBC driver throws an `SQLException`.

(5) Objects that can be specified with setObject

Objects of the following types can be specified for the `x` argument of the `setObject` method:

- 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.Boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

8.5 ResultSet interface

This section explains the methods provided by the `ResultSet` interface.

8.5.1 List of the methods in the ResultSet interface

(1) Main functions of the ResultSet interface

The `ResultSet` interface provides the following main functions:

- Movement of the cursor within a result set in units of rows
- Return of results data
- Notification of whether the retrieval result is the null value

(2) Methods in the ResultSet interface that are supported by HADB

The following table lists and describes the methods in the `ResultSet` interface that are supported by HADB.

Table 8-14: Methods in the ResultSet interface

No.	Method in the ResultSet interface	Function
1	`absolute(int row)`	Moves the cursor to a specified row in the `ResultSet` object.
2	`afterLast()`	Moves the cursor to the location immediately following the last row in the `ResultSet` object.
3	`beforeFirst()`	Moves the cursor to the location immediately preceding the first row in the `ResultSet` object.
4	`clearWarnings()`	Clears all warnings concerning this `ResultSet` object that have been reported.
5	`close()`	Closes the cursor that has been opened for the `ResultSet` object and releases JDBC resources.
6	`findColumn(String columnName)`	Returns the column number corresponding to a specified column name.
7	`first()`	Moves the cursor to the first row in the `ResultSet` object.
8	`getAsciiStream(int columnIndex)`	Acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object.
9	`getAsciiStream(String columnName)`	
10	`getBigDecimal(int columnIndex)`	Acquires in a `java.math.BigDecimal` object the value in a specified column in the current row of the `ResultSet` object.
11	`getBigDecimal(String columnName)`	
12	`getBinaryStream(int columnIndex)`	Acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object.
13	`getBinaryStream(String columnName)`	
14	`getBoolean(int columnIndex)`	Acquires as `boolean` in the Java programming language the value in a specified column in the current row of the `ResultSet` object.
15	`getBoolean(String columnName)`	
16	`getByte(int columnIndex)`	Acquires as `byte` in the Java programming language the value in a specified column in the current row of the `ResultSet` object.

No.	Method in the ResultSet interface	Function
17	getByte(String columnName)	
18	getBytes(int columnIndex)	Acquires as a byte array in the Java programming language the value in a specified column in the current row of the ResultSet object.
19	getBytes(String columnName)	
20	getCharacterStream(int columnIndex)	Acquires in a java.io.Reader object the value in a specified column in the current row of the ResultSet object.
21	getCharacterStream(String columnName)	
22	getConcurrency()	Acquires this ResultSet object's concurrent processing mode.
23	getCursorName()	Acquires the name of the SQL cursor used by this ResultSet object.
24	getDate(int columnIndex)	Acquires in a java.sql.Date object the value in a specified column in the current row of the ResultSet object.
25	getDate(int columnIndex, Calendar cal)	
26	getDate(String columnName)	
27	getDate(String columnName, Calendar cal)	
28	getDouble(int columnIndex)	Acquires as double in the Java programming language the value in a specified column in the current row of the ResultSet object.
29	getDouble(String columnName)	
30	getFetchDirection()	Acquires this ResultSet object's fetch direction.
31	getFetchSize()	Acquires this ResultSet object's fetch size.
32	getFloat(int columnIndex)	Acquires as float in the Java programming language the value in a specified column in the current row of the ResultSet object.
33	getFloat(String columnName)	
34	getHoldability()	Acquires a value indicating the status of the holding functionality for this ResultSet object.
35	getInt(int columnIndex)	Acquires as int in the Java programming language the value in a specified column in the current row of the ResultSet object.
36	getInt(String columnName)	
37	getLong(int columnIndex)	Acquires as long in the Java programming language the value in a specified column in the current row of the ResultSet object.
38	getLong(String columnName)	
39	getMetaData()	Acquires this ResultSet object's meta information.
40	getObject(int columnIndex)	Acquires as Object in the Java programming language the value in a specified column in the current row of the ResultSet object.
41	getObject(String columnName)	
42	getObject(int columnIndex,Class<T> type)	Acquires the value for a specified column in the current row of the ResultSet object, and converts it to a Java data type.
43	getObject(String columnLabel,Class<T> type)	
44	getRow()	Acquires the current row number.
45	getShort(int columnIndex)	Acquires as short in the Java programming language the value in a specified column in the current row of the ResultSet object.
46	getShort(String columnName)	
47	getStatement()	Acquires the Statement object that created this ResultSet object.

No.	Method in the ResultSet interface	Function
48	`getString(int columnIndex)`	Acquires as `String` in the Java programming language the value in a specified column in the current row of the `ResultSet` object.
49	`getString(String columnName)`	
50	`getTime(int columnIndex)`	Acquires in a `java.sql.Time` object the value in a specified column in the current row of the `ResultSet` object.
51	`getTime(int columnIndex, Calendar cal)`	
52	`getTime(String columnName)`	
53	`getTime(String columnName, Calendar cal)`	
54	`getTimestamp(int columnIndex)`	Acquires in a `java.sql.Timestamp` object the value in a specified column in the current row of the `ResultSet` object.
55	`getTimestamp(int columnIndex, Calendar cal)`	
56	`getTimestamp(String columnName)`	
57	`getTimestamp(String columnName, Calendar cal)`	
58	`getType()`	Returns the `ResultSet` object's type.
59	`getWarnings()`	Acquires the first warning reported by a call related to this `ResultSet` object.
60	`isAfterLast()`	Acquires a value indicating whether the cursor is located after the last row in the `ResultSet` object.
61	`isBeforeFirst()`	Acquires a value indicating whether the cursor is located before the first row in the `ResultSet` object.
62	`isClosed()`	Acquires a value indicating whether this `ResultSet` object is closed.
63	`isFirst()`	Acquires a value indicating whether the cursor is located on the first row in the `ResultSet` object.
64	`isLast()`	Acquires a value indicating whether the cursor is located on the last row in the `ResultSet` object.
65	`last()`	Moves the cursor to the last row of the `ResultSet` object.
66	`next()`	Moves the cursor to the next row.
67	`previous()`	Moves the cursor to the immediately preceding row.
68	`relative(int rows)`	Moves the cursor.
69	`setFetchDirection(int direction)`	Specifies the fetch direction for the `ResultSet` object.
70	`setFetchSize(int rows)`	Specifies the fetch size (number of rows to be fetched) when the `ResultSet` object is retrieved.
71	`wasNull()`	Returns a value indicating whether the last column value acquired is the null value.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` might be thrown.

(3) Required package name and class name

The package and class names required in order to use the `ResultSet` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbResultSet`

8.5.2 absolute(int row)

(1) Function

This method moves the cursor to a specified row in the `ResultSet` object.

(2) Format

```
public synchronized boolean absolute(int row) throws SQLException
```

(3) Arguments

`int row`

Specifies the number of the row to which the cursor is to be moved. A positive number indicates that the row numbers in the result set are to be counted from the beginning, and a negative number indicates that the row numbers are to be counted from the end of the result set.

(4) Return value

If the cursor position resulting from the `absolute` method call is before the first row or after the last row, the method returns `false`; otherwise, the method returns `true`.

The following table shows the destination of the cursor and the return value when the `absolute` method is executed.

Table 8-15: Location to which the cursor is moved and return value when the absolute method is executed

Number of rows in result set#	Specified row value	Destination of cursor	Return value
0	Non-zero value	Remains before the first row	`false`
n	*n* < `row`	After the last row	`false`
	1 ≤ `row` ≤ *n*	`row`	`true`
	-*n* ≤ `row` ≤ -1	(*n* + 1) + `row`	`true`
	`row` < -*n*	Before the first row	`false`

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

If the actual number of rows is greater than the `setLargeMaxRows` value, the `setLargeMaxRows` value takes effect.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- A value of `0` was specified for `row`.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.3 afterLast()

(1) Function

This method moves the cursor to the location immediately following the last row in the `ResultSet` object. The following table shows the location to which the cursor is moved when this method is executed.

Table 8-16: Location to which the cursor is moved when the afterLast method is executed

Number of rows in result set#	Current cursor position	Destination of cursor
`0`	Before the first row	Remains before the first row.
n	Before the first row	After the last row
	1 ≤ current row ≤ *n*	After the last row
	After the last row	Remains after the last row.

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

If the actual number of rows is greater than the `setLargeMaxRows` value, the `setLargeMaxRows` value takes effect.

(2) Format

```
public synchronized void afterLast() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.4 beforeFirst()

(1) Function

This method moves the cursor to the location immediately preceding the first row in the `ResultSet` object. The following table shows the location to which the cursor is moved when this method is executed.

Table 8-17: Location to which the cursor is moved when the beforeFirst method is executed

Number of rows in result set#	Current cursor position	Destination of cursor
0	Before the first row	Remains before the first row.
n	Before the first row	Remains before the first row.
	1 ≤ current row ≤ *n*	Before the first row
	After the last row	Before the first row

#

If the actual number of rows is greater than the `setMaxRows` value, the `setMaxRows` value takes effect.

If the actual number of rows is greater than the `setLargeMaxRows` value, the `setLargeMaxRows` value takes effect.

(2) Format

```
public synchronized void beforeFirst() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.5 clearWarnings()

(1) Function

This method clears all warnings concerning this `ResultSet` object that have been reported.

(2) Format

```
public synchronized void clearWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

If the `ResultSet` object has become invalid due to transaction settlement, the JDBC driver throws an `SQLException`.

8.5.6 close()

(1) Function

This method closes the cursor that has been opened for the `ResultSet` object and releases JDBC resources.

(2) Format

```
public synchronized void close() throws SQLException
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

If a database access error occurs, the method throws an `SQLException`.

8.5.7 findColumn(String columnName)

(1) Function

This method returns the column number corresponding to a specified column name.

(2) Format

```
public synchronized int findColumn(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name. The column number corresponding to this column name is returned.

The following notes apply to specifying a column name:

- Column names are not case sensitive.
- Because the entire character string that is specified is assumed to be the column name, any double quotation mark (") specified in the character string is assumed to be part of the column name.
- If more than one column has the specified column name, the smallest column number corresponding to that column name is returned.

(4) Return value

This method returns the column number corresponding to the specified column name.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object was invalidated by transaction settlement.

- A nonexistent column name was specified.
- `null` or a character string with a length of zero was specified for `columnName`.

8.5.8 first()

(1) Function

This method moves the cursor to the first row in the `ResultSet` object.

(2) Format

```
public synchronized boolean first() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the number of rows in the result set is 0, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.9 getAsciiStream(int columnIndex)

(1) Function

This method acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

The method does not convert the value to ASCII characters.

(2) Format

```
public synchronized InputStream getAsciiStream(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns a `java.io.InputStream` object that contains the retrieval result. The following table shows the relationship between the retrieval result and the return value.

Table 8-18: Relationship between the retrieval result and the return value (getAsciiStream method)

HADB data type	Retrieval result	Return value

`CHAR`
`VARCHAR`
`BINARY`
`VARBINARY`
	Null value	`null`

	Other than the above	`java.io.InputStream` object containing the retrieval result
Other	Not applicable	`SQLException` is thrown.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.10 getAsciiStream(String columnName)

(1) Function

This method acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

The method does not convert the value to ASCII characters.

(2) Format

```
public synchronized InputStream getAsciiStream(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns a `java.io.InputStream` object that contains the retrieval result. For details about the relationship between the retrieval result and the return value, see Table 8-18: Relationship between the retrieval result and the return value (getAsciiStream method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.11 getBigDecimal(int columnIndex)

(1) Function

This method acquires in a `java.math.BigDecimal` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized BigDecimal getBigDecimal(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value that corresponds to the specified column number in the current row. The following table shows the relationship between the retrieval result and the return value.

Table 8-19: Relationship between the retrieval result and the return value (getBigDecimal method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	null
	[*space*]*integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*]	java.math.BigDecimal object containing the retrieval result (the value without the spaces at the beginning and end of the character string is used as the java.math.BigDecimal object)
	Other than the above	SQLException is thrown.
SMALLINT	Null value	null
	Other than the above	java.math.BigDecimal object containing the retrieval result
INTEGER	Null value	null
	Other than the above	java.math.BigDecimal object containing the retrieval result
DECIMAL	Null value	null
	Other than the above	java.math.BigDecimal object containing the retrieval result
DOUBLE PRECISION	Null value	null
	Other than the above	java.math.BigDecimal object containing the retrieval result
BOOLEAN#	Null value	null
	true	java.math.BigDecimal object obtained based on BigDecimal(1)
	false	java.math.BigDecimal object obtained based on BigDecimal(0)
Other	Not applicable	SQLException is thrown.

#

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.
  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.
- The Connection used to create the Statement object that created this ResultSet object has been closed.
- The ResultSet object has become invalid due to transaction settlement.

- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `BigDecimal`.
- An error occurred in the JDBC driver.

8.5.12 getBigDecimal(String columnName)

(1) Function

This method acquires in a `java.math.BigDecimal` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized BigDecimal getBigDecimal(String columnName) throws SQLException
```

(3) Arguments

`String columnName`
Specifies a column name.

(4) Return value

This method returns the column value that corresponds to the specified column name in the current row.

For details about the relationship between the retrieval result and the return value, see Table 8-19: Relationship between the retrieval result and the return value (getBigDecimal method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `BigDecimal`.
- An error occurred in the JDBC driver.

8.5.13 getBinaryStream(int columnIndex)

(1) Function

This method acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized InputStream getBinaryStream(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

The method returns a `java.io.InputStream` object containing the retrieval result. The following table shows the relationship between the retrieval result and the return value.

Table 8-20: Relationship between the retrieval result and the return value (getBinaryStream method)

HADB data type	Retrieval result	Return value
`BINARY` `VARBINARY`	Null value	`null`
	Other than the above	`java.io.InputStream` object containing the retrieval result
Other	Not applicable	`SQLException` is thrown.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.14 getBinaryStream(String columnName)

(1) Function

This method acquires in a `java.io.InputStream` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized InputStream getBinaryStream(String columnName) throws SQLExceptio
n
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

The method returns a java.io.InputStream object containing the retrieval result. For details about the relationship between the retrieval result and the return value, see Table 8-20: Relationship between the retrieval result and the return value (getBinaryStream method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.15 getBoolean(int columnIndex)

(1) Function

This method acquires as `boolean` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized boolean getBoolean(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns `true` or `false`. The following table shows the relationship between the retrieval result and the return value.

Table 8-21: Relationship between the retrieval result and the return value (getBoolean method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`false`
	[*space*]`1`[*space*]	`true`
	Other than the above	`false`
`SMALLINT`	Null value	`false`
	`0`	`false`
	Other than the above	`true`
`INTEGER`	Null value	`false`
	`0`	`false`
	Other than the above	`true`
`DECIMAL`	Null value	`false`
	`0[.00...0]`	`false`
	Other than the above	`true`
`DOUBLE PRECISION`	Null value	`false`
	`0.0` or `-0.0`	`false`
	Other than the above	`true`
`BOOLEAN`#	Null value	`false`
	Non-null value	*retrieval-result*
Other	Not applicable	`SQLException` is thrown.

#

`BOOLEAN`-type data exists when the `ResultSet` object was created from `DatabaseMetadata`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.

- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.16 getBoolean(String columnName)

(1) Function

This method acquires as `boolean` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized boolean getBoolean(String columnName) throws SQLException
```

(3) Arguments

`String` `columnName`
Specifies a column name.

(4) Return value

This method returns `true` or `false`. For details about the relationship between the retrieval result and the return value, see Table 8-21: Relationship between the retrieval result and the return value (getBoolean method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.17 getByte(int columnIndex)

(1) Function

This method acquires as `byte` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized byte getByte(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-22: Relationship between the retrieval result and the return value (getByte method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and `Byte.MIN_VALUE` or greater, and `Byte.MAX_VALUE` or less	Retrieval result converted to a `byte` value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `Byte.MAX_VALUE` or less than `Byte.MIN_VALUE`	`SQLException` is thrown.
	[*space*] [+] `Infinity` [*space*]	
	[*space*] `-Infinity` [*space*]	
	[*space*] [+|-] `NaN` [*space*]	
	Other than the above	
`SMALLINT`	Null value	`0`
	`Byte.MIN_VALUE` or greater and `Byte.MAX_VALUE` or less	Retrieval result converted to a `byte` value
	Other than the above	`SQLException` is thrown.
`INTEGER`	Null value	`0`
	`Byte.MIN_VALUE` or greater and `Byte.MAX_VALUE` or less	Retrieval result converted to a `byte` value
	Other than the above	`SQLException` is thrown.

HADB data type	Retrieval result	Return value
DECIMAL	Null value	0
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the retrieval result converted to a byte value
	Other than the above	SQLException is thrown.
DOUBLE PRECISION	Null value	0
	Byte.MIN_VALUE or greater and Byte.MAX_VALUE or less	Integer part of the retrieval result converted to a byte value
	Other than the above	SQLException is thrown.
BOOLEAN#	Null value	0
	true	1
	false	0
Other	Not applicable	SQLException is thrown.

#

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.
- The Connection used to create the Statement object that created this ResultSet object has been closed.
- The ResultSet object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as byte.
- An error occurred in the JDBC driver.

8.5.18 getByte(String columnName)

(1) Function

This method acquires as byte in the Java programming language the value in a specified column in the current row of the ResultSet object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized byte getByte(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-22: Relationship between the retrieval result and the return value (getByte method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `byte`.
- An error occurred in the JDBC driver.

8.5.19 getBytes(int columnIndex)

(1) Function

This method acquires as a `byte` array in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized byte[] getBytes(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-23: Relationship between the retrieval result and the return value (getBytes method)

HADB data type	Retrieval result	Return value
BINARY VARBINARY	Null value	null
	Other than the above	Retrieval result converted to a byte value
ROW	The retrieval result will never be the null value.	Retrieval result converted to a byte value
Other	Not applicable	SQLException is thrown.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.
  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.
- The Connection used to create the Statement object that created this ResultSet object has been closed.
- The ResultSet object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.20 getBytes(String columnName)

(1) Function

This method acquires as a byte array in the Java programming language the value in a specified column in the current row of the ResultSet object. The column whose value is to be acquired is specified in the argument.

The byte value indicates the row value returned by the JDBC driver.

(2) Format

```
public synchronized byte[] getBytes(String columnName) throws SQLException
```

(3) Arguments

String columnName

Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-23: Relationship between the retrieval result and the return value (getBytes method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.21 getCharacterStream(int columnIndex)

(1) Function

This method acquires in a `java.io.Reader` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized Reader getCharacterStream(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns a `java.io.Reader` object containing the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-24: Relationship between the retrieval result and the return value (getCharacterStream method)

HADB data type	Retrieval result	Return value

`CHAR`
`VARCHAR`
`BINARY`
`VARBINARY`
	Null value	`null`

	Other than the above	`java.io.Reader` object containing the retrieval result
Other	Not applicable	`SQLException` is thrown.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- Encoding failed.
- An error occurred in the JDBC driver.

8.5.22 getCharacterStream(String columnName)

(1) Function

This method acquires in a `java.io.Reader` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized Reader getCharacterStream(String columnName) throws SQLException
```

(3) Arguments

`String` `columnName`
Specifies a column name.

(4) Return value

This method returns a `java.io.Reader` object containing the column value. For details about the relationship between the retrieval result and the return value, see Table 8-24: Relationship between the retrieval result and the return value (getCharacterStream method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- Encoding failed.
- An error occurred in the JDBC driver.

8.5.23 getConcurrency()

(1) Function

This method acquires this `ResultSet` object's concurrent processing mode.

(2) Format

```
public synchronized int getConcurrency() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `ResultSet.CONCUR_READ_ONLY`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.24 getCursorName()

(1) Function

This method acquires the name of the SQL cursor used by this `ResultSet` object.

(2) Format

```
public synchronized String getCursorName() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a null character string.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.25 getDate(int columnIndex)

(1) Function

This method acquires in a `java.sql.Date` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Date getDate(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns a `java.sql.Date` object containing the column value. The following table shows the relationship between the retrieval result and the return value.

For details about the conversion for `DATE`, `TIME`, `TIMESTAMP`, or character string type (`CHAR` or `VARCHAR`), see (2) Data conversion process during execution of a getXXX method (for DATE, TIME, TIMESTAMP, or character string type) in 7.6.2 Data conversion process.

Table 8-25: Relationship between the retrieval result and the return value (getDate method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`null`

HADB data type	Retrieval result	Return value
	[*space*]*formatted-date*[#][*space*]	Retrieval result converted to a `java.sql.Date` object without the spaces at the beginning and end of the retrieval result
	Other than the above	`SQLException` is thrown.
`DATE`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Date` object
`TIME`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Date` object
`TIMESTAMP`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Date` object
Other	Not applicable	`SQLException` is thrown.

#

formatted-date means a character string in the date format *YYYY-MM-DD*.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

8.5.26 getDate(int columnIndex, Calendar cal)

(1) Function

This method acquires in a `java.sql.Date` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate date as a millisecond value.

(2) Format

```
public synchronized java.sql.Date getDate(int columnIndex, Calendar cal) throws SQLEx
ception
```

(3) Arguments

`int columnIndex`

Specifies a column number.

`Calendar cal`

Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns a `java.sql.Date` object containing the column value. For details about the relationship between the retrieval result and the return value, see Table 8-25: Relationship between the retrieval result and the return value (getDate method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

8.5.27 getDate(String columnName)

(1) Function

This method acquires in a `java.sql.Date` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Date getDate(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns a `java.sql.Date` object containing the column value. For details about the relationship between the retrieval result and the return value, see Table 8-25: Relationship between the retrieval result and the return value (getDate method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

8.5.28 getDate(String columnName, Calendar cal)

(1) Function

This method acquires in a `java.sql.Date` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate date as a millisecond value.

(2) Format

```
public synchronized java.sql.Date getDate(String columnName, Calendar cal) throws SQL
Exception
```

(3) Arguments

`String columnName`

Specifies a column name.

`Calendar cal`

Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns a `java.sql.Date` object containing the column value. For details about the relationship between the retrieval result and the return value, see Table 8-25: Relationship between the retrieval result and the return value (getDate method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Date`.
- An error occurred in the JDBC driver.

8.5.29 getDouble(int columnIndex)

(1) Function

This method acquires as `double` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized double getDouble(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-26: Relationship between the retrieval result and the return value (getDouble method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`0.0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and – `Double.MAX_VALUE` or greater, and `Double.MIN_VALUE` or less, and `Double.MIN_VALUE` or greater, and `Double.MAX_VALUE` or less	Retrieval result converted to a `double` value

HADB data type	Retrieval result	Return value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `Double.MAX_VALUE`	`Infinity`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and less than `-Double.MAX_VALUE`	`-Infinity`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and less than `Double.MIN_VALUE`, and greater than 0	`0.0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `-Double.MIN_VALUE`, and less than 0	`-0.0`
	[*space*]`-Infinity`[*space*]	`-Infinity`
	[*space*]`[+]Infinity`[*space*]	`Infinity`
	[*space*]`[+|-]NaN`[*space*]	`NaN`
	Other than the above (cannot be converted to a `double` value)	`SQLException` is thrown.
`SMALLINT`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `double` value
`INTEGER`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `double` value
`DECIMAL`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `double` value
`DOUBLE PRECISION`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `double` value
`BOOLEAN`#	Null value	`0.0`
	`true`	`1.0`
	`false`	`0.0`
Other	Not applicable	`SQLException` is thrown.

#

`BOOLEAN`-type data exists when the `ResultSet` object was created from `DatabaseMetadata`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `double`.
- An error occurred in the JDBC driver.

8.5.30 getDouble(String columnName)

(1) Function

This method acquires as `double` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized double getDouble(String columnName) throws SQLException
```

(3) Arguments

`String columnName`
Specifies a column name.

(4) Return value

This method returns the column value.

For details about the relationship between the retrieval result and the return value, see Table 8-26: Relationship between the retrieval result and the return value (getDouble method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.

- The column value cannot be acquired as `double`.
- An error occurred in the JDBC driver.

8.5.31 getFetchDirection()

(1) Function

This method acquires this `ResultSet` object's fetch direction.

(2) Format

```
public synchronized int getFetchDirection() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `ResultSet.FETCH_FORWARD`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.32 getFetchSize()

(1) Function

This method acquires this `ResultSet` object's fetch size.

(2) Format

```
public synchronized int getFetchSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the current fetch size of this `ResultSet` object. The method returns the value set by the `setFetchSize` method. If no value has been set by the `setFetchSize` method, the method returns `0`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.33 getFloat(int columnIndex)

(1) Function

This method acquires as `float` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized float getFloat(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`
Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-27: Relationship between the retrieval result and the return value (getFloat method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`0.0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and one of the following: • `-Float.MAX_VALUE` or greater and `-Float.MIN_VALUE` or less • `Float.MIN_VALUE` or greater and `Float.MAX_VALUE` or less	Retrieval result converted to a `float` value

HADB data type	Retrieval result	Return value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `Float.MAX_VALUE`	`Infinity`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and less than `-Float.MAX_VALUE`	`-Infinity`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and less than `Float.MIN_VALUE`, and greater than `0`	`0.0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `-Float.MIN_VALUE`, and less than 0	`-0.0`
	[*space*]`-Infinity`[*space*]	`-Infinity`
	[*space*]`[+]Infinity`[*space*]	`Infinity`
	[*space*]`[+|-]NaN`[*space*]	`NaN`
	Other than the above (cannot be converted to a `float` value)	`SQLException` is thrown.
`SMALLINT`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `float` value
`INTEGER`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `float` value
`DECIMAL`	Null value	`0.0`
	Other than the above	Retrieval result converted to a `float` value
`DOUBLE PRECISION`	Null value	`0.0`
	One of the following: • `-Float.MAX_VALUE` or greater and `-Float.MIN_VALUE` or less • `Float.MIN_VALUE` or greater and `Float.MAX_VALUE` or less	Retrieval result converted to a `float` value
	Greater than `Float.MAX_VALUE`	`Infinity`
	Less than `-Float.MAX_VALUE`	`-Infinity`
	Less than `Float.MIN_VALUE` and greater than 0	`0.0`
	Greater than `-Float.MIN_VALUE` and less than 0	`-0.0`
`BOOLEAN`#	Null value	`0.0`
	`true`	`1.0`
	`false`	`0.0`

HADB data type	Retrieval result	Return value
Other	Not applicable	`SQLException` is thrown.

#

`BOOLEAN`-type data exists when the `ResultSet` object was created from `DatabaseMetadata`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `float`.
- An error occurred in the JDBC driver.

8.5.34 getFloat(String columnName)

(1) Function

This method acquires as `float` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized float getFloat(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value.

For details about the relationship between the retrieval result and the return value, see Table 8-27: Relationship between the retrieval result and the return value (getFloat method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `float`.
- An error occurred in the JDBC driver.

8.5.35 getHoldability()

(1) Function

This method acquires a value indicating the status of the holding functionality for this `ResultSet` object.

(2) Format

```
public synchronized int getHoldability() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
- The `Statement` object that created this `ResultSet` object is closed.
- The `Connection` object that created the `Statement` object that created this `ResultSet` object is closed.

8.5.36 getInt(int columnIndex)

(1) Function

This method acquires as `int` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized int getInt(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-28: Relationship between the retrieval result and the return value (getInt method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`0`
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and `Integer.MIN_VALUE` or greater, and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than `Integer.MAX_VALUE` or less than `Integer.MIN_VALUE` or less	`SQLException` is thrown.
	[*space*]`-Infinity`[*space*]	
	[*space*][`+`]`Infinity`[*space*]	
	[*space*][`+`|`-`]`NaN`[*space*]	
	Other than the above (cannot be converted to a `double` value)	
`SMALLINT`	Null value	`0`
	Other than the above	Retrieval result converted to an `int` value
`INTEGER`	Null value	`0`
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Retrieval result converted to an `int` value
	Other than the above	`SQLException` is thrown.
`DECIMAL`	Null value	`0`

HADB data type	Retrieval result	Return value
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	Other than the above	`SQLException` is thrown.
`DOUBLE PRECISION`	Null value	`0`
	`Integer.MIN_VALUE` or greater and `Integer.MAX_VALUE` or less	Integer part of the retrieval result converted to an `int` value
	Other than the above	`SQLException` is thrown.
`BOOLEAN`#	Null value	`0`
	`true`	`1`
	`false`	`0`
Other	Not applicable	`SQLException` is thrown.

#

`BOOLEAN`-type data exists when the `ResultSet` object was created from `DatabaseMetadata`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `int`.
- An error occurred in the JDBC driver.

8.5.37 getInt(String columnName)

(1) Function

This method acquires as `int` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized int getInt(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value.

For details about the relationship between the retrieval result and the return value, see Table 8-28: Relationship between the retrieval result and the return value (getInt method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `int`.
- An error occurred in the JDBC driver.

8.5.38 getLong(int columnIndex)

(1) Function

This method acquires as `long` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized long getLong(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-29: Relationship between the retrieval result and the return value (getLong method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	0
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and Long.MIN_VALUE or greater, and Long.MAX_VALUE or less	Integer part of the retrieval result converted to a long value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than Long.MAX_VALUE or less than Long.MIN_VALUE	SQLException is thrown.
	[*space*]-Infinity[*space*]	
	[*space*][+]Infinity[*space*]	
	[*space*][+|-]NaN[*space*]	
	Other than the above (cannot be converted to a double value or BigDecimal object)	
SMALLINT	Null value	0
	Other than the above	Retrieval result converted to a long value
INTEGER	Null value	0
	Other than the above	Retrieval result converted to a long value
DECIMAL	Null value	0
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the retrieval result converted to a long value
DOUBLE PRECISION	Null value	0
	Long.MIN_VALUE or greater and Long.MAX_VALUE or less	Integer part of the retrieval result converted to a long value
	Other than the above	SQLException is thrown.
BOOLEAN#	Null value	0
	true	1
	false	0
Other	Not applicable	SQLException is thrown.

#

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.

- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `long`.
- An error occurred in the JDBC driver.

8.5.39 getLong(String columnName)

(1) Function

This method acquires as `long` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized long getLong(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value.

For details about the relationship between the retrieval result and the return value, see Table 8-29: Relationship between the retrieval result and the return value (getLong method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `long`.
- An error occurred in the JDBC driver.

8.5.40 getMetaData()

(1) Function

This method acquires this `ResultSet` object's meta information.

(2) Format

```
public synchronized ResultSetMetaData getMetaData() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns meta information for this `ResultSet` object as a `ResultSetMetaData` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.41 getObject(int columnIndex)

(1) Function

This method acquires as `Object` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized Object getObject(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value as a Java object.

This Java object has the default Java object type that corresponds to the column's SQL type based on the mapping of built-in types specified in the JDBC specifications.

The following table shows the relationship between the retrieval result and the return value.

Table 8-30: Relationship between the retrieval result and the return value (getObject method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	null
	Other than the above	*retrieval-result*
SMALLINT	Null value	null
	Other than the above	Integer object created by using the retrieval result
INTEGER	Null value	null
	Other than the above	Long object created by using the retrieval result
DECIMAL	Null value	null
	Other than the above	*retrieval-result*
DOUBLE PRECISION	Null value	null
	Other than the above	Double object created by using the retrieval result
DATE	Null value	null
	Other than the above	java.sql.Date object created by using the retrieval result
TIME	Null value	null
	Other than the above	java.sql.Time object created by using the retrieval result
TIMESTAMP	Null value	null
	Other than the above	java.sql.Timestamp object created by using the retrieval result
BINARY VARBINARY	Null value	null
	Other than the above	*retrieval-result*
ROW	Non-null value#1	*retrieval-result*
BOOLEAN#2	Null value	null
	Non-null value	Boolean object created by using the retrieval result

#1

The retrieval result will never be the null value.

#2

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.

  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.

- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- An error occurred in the JDBC driver.

8.5.42 getObject(String columnName)

(1) Function

This method acquires as `Object` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized Object getObject(String columnName) throws SQLException
```

(3) Arguments

`String` `columnName`
Specifies a column name.

(4) Return value

This method returns the column value as a Java object. For details about the relationship between the retrieval result and the return value, see Table 8-30: Relationship between the retrieval result and the return value (getObject method).

This Java object has the default Java object type that corresponds to the column's SQL type based on the mapping of built-in types specified in the JDBC specifications.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- An error occurred in the JDBC driver.

8.5.43 getObject(int columnIndex,Class<T> type)

(1) Function

This method acquires the value in a specified column of the current row of a `ResultSet` object, and converts it to the Java data type of the specified class. The column whose value is to be acquired is specified in the argument of the method.

(2) Format

```
public synchronized <T> T getObject(int columnIndex,Class<T> type) throws SQLExceptio
n
```

(3) Arguments

`int columnIndex`**:**

Specifies the column number.

`Class<T> type`**:**

Specifies the class that represents the Java data type after conversion. The value of the column specified by `columnIndex` is converted to the Java data type of the specified class.

The following table lists the conversions that are possible between data types. If you specify a combination that is not listed in the table, an error occurs.

Table 8-31: Combinations of HADB data types and Java data types

HADB data type	Java data type (value specified for <T>)
CHAR	String
VARCHAR	
SMALLINT	Integer
INTEGER	Long
DECIMAL	java.math.BigDecimal
DOUBLE PRECISION	Double
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BINARY	byte[]
VARBINARY	
ROW	
BOOLEAN#	Boolean

#

`BOOLEAN` type data will exist if the `Resultset` object was created from `DatabaseMetadata`.

(4) Return value

The value in the column is returned as an object of the specified class. The following table lists the relationship between retrieval results and return values.

Table 8-32: Relationship between retrieval results and return values (getObject method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	null
	Other	String object created from retrieval result
SMALLINT	Null value	null
	Other	Integer object created from retrieval result
INTEGER	Null value	null
	Other	Long object created from retrieval result
DECIMAL	Null value	null
	Other	java.math.BigDecimal object created from retrieval result
DOUBLE PRECISION	Null value	null
	Other	Double object created from retrieval result
DATE	Null value	null
	Other	java.sql.Date object created from retrieval result
TIME	Null value	null
	Other	java.sql.Time object created from retrieval result
TIMESTAMP	Null value	null
	Other	java.sql.Timestamp object created from retrieval result
BINARY VARBINARY	Null value	null
	Other	*Retrieval result*
ROW	Non-null value[#1]	*Retrieval result*
BOOLEAN[#2]	Null value	null
	Other	Boolean object created from retrieval result

#1

A retrieval result cannot be a null value.

#2

BOOLEAN type data will exist if the Resultset object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- The ResultSet object is closed.

  This includes situations in which the ResultSet object is closed because the Statement object that created the ResultSet object is closed.

- The `Connection` object that created the `Statement` object that created the `ResultSet` object is closed.
- The `ResultSet` object was invalidated by transaction settlement.
- A nonexistent column number was specified.
- An error occurred in the JDBC driver.
- `null` was specified for `type`.
- The value specified for `type` is not one of the permitted combinations.

8.5.44 getObject(String columnLabel,Class<T> type)

(1) Function

This method acquires the value in a specified column of the current row of a `ResultSet` object, and converts it to the Java data type of the specified class. The column whose value is to be acquired is specified in the argument of the method.

(2) Format

```
public synchronized <T> T getObject(String columnLabel,Class<T> type) throws SQLExcep
tion
```

(3) Arguments

`String columnLabel`

Specifies the column name.

`Class<T> type`

Specifies the class that represents the Java data type after conversion. The value of the column specified by `columnLabel` is converted to the Java data type of the specified class.

Table 8-31: Combinations of HADB data types and Java data types lists the conversions that are possible between data types. If you specify a combination that is not listed in Table 8-31: Combinations of HADB data types and Java data types, an error occurs.

(4) Return value

The value in the column is returned as an object of the specified class. For details about the relationship between the retrieval result and the return value, see Table 8-32: Relationship between retrieval results and return values (getObject method).

(5) Exceptions

For details about exceptions, see (5) Exceptions in 8.5.43 getObject(int columnIndex,Class<T> type).

8.5.45 getRow()

(1) Function

This method acquires the current row number, such as `1` for the first row, `2` for the second row, and so on. If the row is before the first row or after the last row, the number that is acquired is `0`.

(2) Format

```
public synchronized int getRow() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the current row number. If the current row number is greater than `Integer.MAX_VALUE`, the method returns `Integer.MAX_VALUE`.

If the maximum number of retrieved rows exceeds 2,147,483,647, the method returns `2147483647`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.46 getShort(int columnIndex)

(1) Function

This method acquires as `short` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized short getShort(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`
Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-33: Relationship between the retrieval result and the return value (getShort method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	0
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and Short.MIN_VALUE or greater, and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value
	[*space*] *integer-in-character-string-representation*, *decimal-number-in-character-string-representation*, or *floating-point-number-in-character-string-representation* [*space*], and greater than Short.MAX_VALUE or less than Short.MIN_VALUE	SQLException is thrown.
	[*space*]-Infinity[*space*]	
	[*space*][+]Infinity[*space*]	
	[*space*][+|-]NaN[*space*]	
	Other than the above (cannot be converted to a double value)	
SMALLINT	Null value	0
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Retrieval result converted to a short value
	Other than the above	SQLException is thrown.
INTEGER	Null value	0
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Retrieval result converted to a short value
	Other than the above	SQLException is thrown.
DECIMAL	Null value	0
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value
	Other than the above	SQLException is thrown.
DOUBLE PRECISION	Null value	0
	Short.MIN_VALUE or greater and Short.MAX_VALUE or less	Integer part of the retrieval result converted to a short value
	Other than the above	SQLException is thrown.
BOOLEAN#	Null value	0
	true	1
	false	0
Other	Not applicable	SQLException is thrown.

#

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `short`.
- An error occurred in the JDBC driver.

8.5.47 getShort(String columnName)

(1) Function

This method acquires as `short` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized short getShort(String columnName) throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-33: Relationship between the retrieval result and the return value (getShort method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.

- The data type cannot be acquired by this method.
- The column value cannot be acquired as `short`.
- An error occurred in the JDBC driver.

8.5.48 getStatement()

(1) Function

This method acquires the `Statement` object that created this `ResultSet` object.

(2) Format

```
public synchronized Statement getStatement() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the `Statement` object that created the `ResultSet` object. If the result set was created by a method in `DatabaseMetaData`, this method returns `null`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.49 getString(int columnIndex)

(1) Function

This method acquires as `String` in the Java programming language the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized String getString(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

Table 8-34: Relationship between the retrieval result and the return value (getString method)

HADB data type	Retrieval result	Return value
CHAR VARCHAR	Null value	null
	Other than the above	*retrieval-result*
SMALLINT	Null value	null
	Other than the above	String object containing the retrieval result in character string representation
INTEGER	Null value	null
	Other than the above	String object containing the retrieval result in character string representation
DECIMAL	Null value	null
	Other than the above	String object containing the retrieval result in character string representation
DOUBLE PRECISION	Null value	null
	Other than the above	String object containing the retrieval result in character string representation
DATE	Null value	null
	Other than the above	String object of a character string in *YYYY-MM-DD* format
TIME	Null value	null
	Other than the above	String object of a character string in *hh*:*mm*:*ss*[.*f*...] format
TIMESTAMP	Null value	null
	Other than the above	String object of a character string in *YYYY-MM-DD*Δ*hh*:*mm*:*ss*[.*f*...] format (Δ indicates a space)
BINARY VARBINARY	Null value	null
	Other than the above	*retrieval-result*
BOOLEAN#	Null value	null
	true	String object of the character string "true"
	false	String object of the character string "false"

#

BOOLEAN-type data exists when the ResultSet object was created from DatabaseMetadata.

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.
  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.
- The Connection used to create the Statement object that created this ResultSet object has been closed.
- The ResultSet object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- Encoding failed.
- The data type cannot be acquired by this method.
- An error occurred in the JDBC driver.

8.5.50 getString(String columnName)

(1) Function

This method acquires as String in the Java programming language the value in a specified column in the current row of the ResultSet object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized String getString(String columnName) throws SQLException
```

(3) Arguments

String columnName
  Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-34: Relationship between the retrieval result and the return value (getString method).

(5) Exceptions

The JDBC driver throws an SQLException in the following cases:

- This ResultSet object is closed.
  This includes the case where the ResultSet object was closed because the Statement object that created this ResultSet object was closed.
- The Connection used to create the Statement object that created this ResultSet object has been closed.
- The ResultSet object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- Encoding failed.
- The data type cannot be acquired by this method.

- An error occurred in the JDBC driver.

8.5.51 getTime(int columnIndex)

(1) Function

This method acquires in a `java.sql.Time` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Time getTime(int columnIndex) throws SQLException
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

For details about the conversion for `DATE`, `TIME`, `TIMESTAMP`, or character string type (`CHAR` or `VARCHAR`), see (2) Data conversion process during execution of a getXXX method (for DATE, TIME, TIMESTAMP, or character string type) in 7.6.2 Data conversion process.

Table 8-35: Relationship between the retrieval result and the return value (getTime method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`null`
	[*space*] *formatted-time*# [*space*]	Retrieval result converted to a `java.sql.Time` object without the spaces at the beginning and end of the retrieval result
	Other than the above	`SQLException` is thrown.
`TIME`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Time` object
`TIMESTAMP`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Time` object
Other	Not applicable	`SQLException` is thrown.

#

formatted-time is a character string in the time format `'`*hh*`:`*mm*`:`*ss*`[.`*f*`...]'`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

8.5.52 getTime(int columnIndex, Calendar cal)

(1) Function

This method acquires in a `java.sql.Time` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate time as a millisecond value.

(2) Format

```
public synchronized java.sql.Time getTime(int columnIndex, Calendar cal) throws SQLEx
ception
```

(3) Arguments

`int columnIndex`
Specifies a column number.

`Calendar cal`
Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-35: Relationship between the retrieval result and the return value (getTime method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

8.5.53 getTime(String columnName)

(1) Function

This method acquires in a `java.sql.Time` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Time getTime(String columnName) throws SQLException
```

(3) Arguments

`String columnName`
Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-35: Relationship between the retrieval result and the return value (getTime method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

8.5.54 getTime(String columnName, Calendar cal)

(1) Function

This method acquires in a `java.sql.Time` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate time as a millisecond value.

(2) Format

```
public synchronized java.sql.Time getTime(String columnName, Calendar cal) throws SQL
Exception
```

(3) Arguments

`String columnName`

Specifies a column name.

`Calendar cal`

Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-35: Relationship between the retrieval result and the return value (getTime method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Time`.
- An error occurred in the JDBC driver.

8.5.55 getTimestamp(int columnIndex)

(1) Function

This method acquires in a `java.sql.Timestamp` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Timestamp getTimestamp(int columnIndex) throws SQLExcept
ion
```

(3) Arguments

`int columnIndex`

Specifies a column number.

(4) Return value

This method returns the column value. The following table shows the relationship between the retrieval result and the return value.

For details about the conversion for `DATE`, `TIME`, `TIMESTAMP`, or character string type (`CHAR` or `VARCHAR`), see (2) Data conversion process during execution of a getXXX method (for DATE, TIME, TIMESTAMP, or character string type) in 7.6.2 Data conversion process.

Table 8-36: Relationship between the retrieval result and the return value (getTimestamp method)

HADB data type	Retrieval result	Return value
`CHAR` `VARCHAR`	Null value	`null`
	[*space*]*formatted-time-stamp*[#][*space*]	Retrieval result converted to a `java.sql.Timestamp` object without the spaces at the beginning and end of the retrieval result
	Other than the above	`SQLException` is thrown.
`DATE`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Timestamp` object
`TIME`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Timestamp` object
`TIMESTAMP`	Null value	`null`
	Other than the above	Retrieval result converted to a `java.sql.Timestamp` object
Other	Not applicable	`SQLException` is thrown.

#

formatted-time-stamp is a character string in the time stamp format `'`*YYYY*−*MM*−*DD hh*:*mm*:*ss*`[.`*f*`...]'`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

8.5.56 getTimestamp(int columnIndex, Calendar cal)

(1) Function

This method acquires in a `java.sql.Timestamp` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate time stamp as a millisecond value.

(2) Format

```
public synchronized java.sql.Timestamp getTimestamp(int columnIndex, Calendar cal) th
rows SQLException
```

(3) Arguments

`int columnIndex`
Specifies a column number.

`Calendar cal`
Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-36: Relationship between the retrieval result and the return value (getTimestamp method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

- A nonexistent column number was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

8.5.57 getTimestamp(String columnName)

(1) Function

This method acquires in a `java.sql.Timestamp` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

(2) Format

```
public synchronized java.sql.Timestamp getTimestamp(String columnName) throws SQLExce
ption
```

(3) Arguments

`String columnName`

Specifies a column name.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-36: Relationship between the retrieval result and the return value (getTimestamp method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

8.5.58 getTimestamp(String columnName, Calendar cal)

(1) Function

This method acquires in a `java.sql.Timestamp` object the value in a specified column in the current row of the `ResultSet` object. The column whose value is to be acquired is specified in the argument.

This method uses a specified calendar to create the appropriate time stamp as a millisecond value.

(2) Format

```
public synchronized java.sql.Timestamp getTimestamp(String columnName, Calendar cal)
throws SQLException
```

(3) Arguments

`String columnName`

Specifies a column name.

`Calendar cal`

Specifies the calendar in which the time zone for the values stored in the database has been set.

(4) Return value

This method returns the column value. For details about the relationship between the retrieval result and the return value, see Table 8-36: Relationship between the retrieval result and the return value (getTimestamp method).

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A nonexistent column name was specified.
- The data type cannot be acquired by this method.
- The column value cannot be acquired as `java.sql.Timestamp`.
- An error occurred in the JDBC driver.

8.5.59 getType()

(1) Function

This method returns the `ResultSet` object's type.

(2) Format

```
public synchronized int getType() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns one of the following values:

`ResultSet.TYPE_FORWARD_ONLY`
The only direction the cursor can move is forward.

`ResultSet.TYPE_SCROLL_INSENSITIVE`
The cursor can be scrolled, but changes to the underlying values are not reflected in the result set.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.60 getWarnings()

(1) Function

This method acquires the first warning reported by a call related to this `ResultSet` object. If there is more than one warning, the subsequent warnings are chained to the first warning and can be acquired by calling the `getNextWarning` method of the `SQLWarning` object for the immediately preceding warning that was acquired.

(2) Format

```
public synchronized SQLWarning getWarnings() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the first `SQLWarning` object. If there is no `SQLWarning` object, the method returns `null`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.61 isAfterLast()

(1) Function

This method acquires a value indicating whether the cursor is located after the last row in the `ResultSet` object.

(2) Format

```
public synchronized boolean isAfterLast() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor is located after the last row, the method returns `true`; if not, or the result set contains no rows, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.62 isBeforeFirst()

(1) Function

This method acquires a value indicating whether the cursor is located before the first row in the `ResultSet` object.

(2) Format

```
public synchronized boolean isBeforeFirst() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor is located before the first row, the method returns `true`; if not, or the result set contains no rows, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.63 isClosed()

(1) Function

This method acquires a value indicating whether this `ResultSet` object is closed.

This method is guaranteed to return `true` only when it is executed after the `close` method has been executed.

(2) Format

```
public synchronized boolean isClosed() throws SQLException
```

(3) Arguments

None.

(4) Return value

If this `ResultSet` object is closed, the method returns `true`; if it is not closed, the method returns `false`.

(5) Exceptions

None.

8.5.64 isFirst()

(1) Function

This method acquires a value indicating whether the cursor is located on the first row in the `ResultSet` object.

(2) Format

```
public synchronized boolean isFirst() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor is located on the first row, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.65 isLast()

(1) Function

This method acquires a value indicating whether the cursor is located on the last row in the `ResultSet` object.

(2) Format

```
public synchronized boolean isLast() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor is located on the last row, the method returns `true`; if not, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.66 last()

(1) Function

This method moves the cursor to the last row of the `ResultSet` object.

(2) Format

```
public synchronized boolean last() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor is moved to the last row, the method returns `true`; if the result set contains no rows, the method returns `false`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.67 next()

(1) Function

This method moves the cursor to the next row. If the current cursor position is before the first row, the method moves the cursor to the first row; if it is on the last row, the method moves the cursor to the location after the last row.

(2) Format

```
public synchronized boolean next() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor position resulting from execution of this method is before the first row or after the last row, the method returns `false`; otherwise, the method returns `true`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.68 previous()

(1) Function

This method moves the cursor to the immediately preceding row.

(2) Format

```
public synchronized boolean previous() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the cursor position resulting from execution of this method is before the first row, the method returns `false`; otherwise, the method returns `true`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.69 relative(int rows)

(1) Function

This method moves the cursor.

(2) Format

```
public synchronized boolean relative(int rows) throws SQLException
```

(3) Arguments

`int rows`

Specifies the number of rows (from the current row) by which the cursor is to be moved.
If a positive value is specified, the cursor moves forward. If a negative value is specified, the cursor moves backwards.

(4) Return value

If the cursor position resulting from execution of this method is before the first row or after the last row, the method returns `false`; otherwise, the method returns `true`.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The type of this `ResultSet` object is `ResultSet.TYPE_FORWARD_ONLY`.

- The current position cannot be acquired.
- The cursor's current position is not on a valid row.
- The `ResultSet` object has become invalid due to transaction settlement.
- A database access error occurs.

8.5.70 setFetchDirection(int direction)

(1) Function

This method specifies the fetch direction for the `ResultSet` object.

(2) Format

```
public synchronized void setFetchDirection(int direction) throws SQLException
```

(3) Arguments

`int direction`

Specifies the fetch direction.

Only `ResultSet.FETCH_FORWARD` can be specified.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- The value specified for `direction` is not `ResultSet.FETCH_FORWARD`.

8.5.71 setFetchSize(int rows)

(1) Function

This method specifies the fetch size (number of rows to be fetched) when the `ResultSet` object is retrieved.

(2) Format

```
public synchronized void setFetchSize(int rows) throws SQLException
```

(3) Arguments

`int rows`

Specifies the number of rows to be fetched, in the range from 0 to 65,535.

If `0` is specified, retrieval is performed based on the value of `adb_clt_fetch_size` in the system properties, user properties, or URL connection properties.

If this method is omitted, the JDBC driver uses the number of rows specified in the `Statement` object for retrieval. If no number of rows is specified in the `Statement` object, or this `ResultSet` object has not been created from a `Statement` object, the JDBC driver uses the value of the `adb_clt_fetch_size` property for retrieval.

(4) Return value

None.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.

  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.
- A value outside the range from 0 to 65,535 was specified in `rows`.
- The value specified in `rows` is greater than the maximum number of rows that can be stored (the value set by the `setMaxRows` method of the `Statement` object that created this `ResultSet` object).
- The value specified in `rows` is greater than the number of rows that can be stored (the value set by the `setMaxLargeRows` method of the `Statement` object that created this `ResultSet` object).

(6) Notes

For notes, see (6) Notes in 8.3.37 setFetchSize(int rows).

8.5.72 wasNull()

(1) Function

This method returns a value indicating whether the last column value acquired is the null value.

(2) Format

```
public synchronized boolean wasNull() throws SQLException
```

(3) Arguments

None.

(4) Return value

If the last column value acquired is `NULL`, the method returns `true`; otherwise, the method returns `false`.

This method returns `false` before any column value has been acquired by a `get`*XXX* method.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- This `ResultSet` object is closed.
  This includes the case where the `ResultSet` object was closed because the `Statement` object that created this `ResultSet` object was closed.
- The `Connection` used to create the `Statement` object that created this `ResultSet` object has been closed.
- The `ResultSet` object has become invalid due to transaction settlement.

8.5.73 Fields supported by the ResultSet interface

The following table lists the fields supported by the `ResultSet` interface.

Table 8-37: Fields supported by the ResultSet interface

Field	Remarks
`public static final int FETCH_FORWARD`	--
`public static final int FETCH_REVERSE`	--
`public static final int FETCH_UNKNOWN`	--
`public static final int TYPE_FORWARD_ONLY`	--
`public static final int TYPE_SCROLL_INSENSITIVE`	--
`public static final int TYPE_SCROLL_SENSITIVE`	When this value is specified, the JDBC driver assumes that `TYPE_SCROLL_INSENSITIVE` was specified.
`public static final int CONCUR_READ_ONLY`	--
`public static final int CONCUR_UPDATABLE`	When this value is specified, the JDBC driver assumes that `CONCUR_READ_ONLY` was specified.
`public static final int HOLD_CURSORS_OVER_COMMIT`	Even if this value is specified, the `ResultSet` object might become invalid after the transaction is settled.
`public static final int CLOSE_CURSORS_AT_COMMIT`	When this value is specified, the JDBC driver assumes that `HOLD_CURSORS_OVER_COMMIT` was specified.

Legend:

--: None.

8.5.74 Notes about the ResultSet interface

(1) Value acquisition using a getXXX method

- For details about whether mapping is possible with a `get`*XXX* method, see (2) Mapping during retrieval data acquisition in 7.6.1 Mapping data types.
- If a nonexistent column number or column name is specified in a `get`*XXX* method, the JDBC driver throws an `SQLException`.
- If a value specified in a `get`*XXX* method cannot represent the actual value (for example, if `getShort` is used to acquire the `INTEGER`-type value `40,000`), overflow occurs and an `SQLException` results. For the combinations of a `get`*XXX* method and HADB data type that can cause overflow to occur, see 7.6.3 Overflow handling.

(2) Data mapping (conversion)

For details about whether mapping is possible with a `get`*XXX* method that is to be used for retrieval data acquisition, see (2) Mapping during retrieval data acquisition in 7.6.1 Mapping data types. If a `get`*XXX* method is called for a JDBC SQL data type that cannot be mapped, the JDBC driver throws an `SQLException`.

(3) Memory size used when the result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE

If the result set type is `ResultSet.TYPE_SCROLL_INSENSITIVE` or `ResultSet.TYPE_SCROLL_SENSITIVE` when the following methods in the `ResultSet` interface are executed, the JDBC driver allocates memory for accumulating the retrieval results:

- `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 is variable length, the memory object is set to the actual size of the retrieved data.

(4) next, absolute, relative, last, and afterLast methods

When the `next` method is executed, the JDBC driver retrieves and accumulates data from the database as described in the following table.

Table 8-38: Data retrieved and accumulated from the database during execution of the next method

Condition	Result set type	
	ResultSet.TYPE_FORWARD_ONLY	ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE
The data on the current row, which was moved by the `next` method, has not been read into the JDBC driver.	The JDBC driver acquires the moved current row from the connected database.	The JDBC driver acquires the moved current row from the connected database, then reads and accumulates the row in its memory.
The data on the current row, which was moved by the `next` method, has been read into the JDBC driver.		The JDBC driver does not retrieve data from the connected database.

When the `absolute`, `relative`, `last`, or `afterLast` method is executed, the JDBC driver retrieves and accumulates data from the database as described in the following table.

Table 8-39: Data retrieved and accumulated from the database during execution of the absolute, relative, last, or afterLast method

Condition	Result set type is ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE
Between the first row and the specified row# of the retrieval results there is data that the JDBC driver has not read.	The JDBC driver retrieves the rows that were not read from the database and accumulates them in its memory.
The JDBC driver has read all the data contained on the first row through the specified row# of the retrieval results.	The JDBC driver does not retrieve data from the database.

Note

If the result set type is `ResultSet.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.

(5) Notes about the getAsciiStream and getCharacterStream methods

The JDBC driver does not implicitly close objects returned by the `getAsciiStream` and `getCharacterStream` methods. The program that called these methods must execute the `close` method.

(6) Maximum number of retrieved rows

The following table shows the number of rows that a `ResultSet` object can retrieve from the HADB server. The JDBC driver discards retrieval results in excess of the applicable number of rows shown in the following table.

Table 8-40: Number of rows that a ResultSet object can retrieve from the HADB server

ResultSet object	Result set type	
	ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE	Other type
`ResultSet` object created from a `Statement` object that executed one of the following methods: • `setMaxRows` • `setLargeMaxRows`	The number of retrieved rows is the number of rows specified by the following methods: • `setMaxRows` • `setLargeMaxRows`	The number of retrieved rows is the number of rows specified in the following methods: • `setMaxRows` • `setLargeMaxRows`

ResultSet object	Result set type	
	ResultSet.TYPE_SCROLL_INSENSITIVE or ResultSet.TYPE_SCROLL_SENSITIVE	Other type
	If a value is specified in the `setLargeMaxRows` method that exceeds `Integer.MAX_VALUE`, the maximum value will be `Integer.MAX_VALUE`.	
Other `ResultSet` object	The maximum value is `Integer.MAX_VALUE`.	There is no limit.

8.6 DatabaseMetaData interface

This section explains the methods provided by the DatabaseMetaData interface.

8.6.1 List of the methods in the DatabaseMetaData interface

(1) Main functions of the DatabaseMetaData Interface

The DatabaseMetaData interface provides the following main functions:

- Return of various information about the connected database
- Return of list information, such as lists of tables or columns (the information is stored in a ResultSet)

(2) Methods in the DatabaseMetaData interface that are supported by HADB

The following table lists and describes the methods in the DatabaseMetaData interface that are supported by HADB.

Table 8-41: Methods in the DatabaseMetaData interface

No.	Method in the DatabaseMetaData interface	Function
1	allProceduresAreCallable()	Acquires a value indicating whether all the procedures returned by the getProcedures method can be called by the current HADB user.
2	allTablesAreSelectable()	Acquires a value indicating whether all the tables returned by the getTables method can be used by the current HADB user.
3	autoCommitFailureClosesAllResultSets()	Returns a value indicating whether all open ResultSet objects are to be closed if an SQLException occurs while the automatic commit mode is enabled.
4	dataDefinitionCausesTransactionCommit()	Acquires a value indicating whether a data definition statement in a transaction is to forcibly commit the transaction.
5	dataDefinitionIgnoredInTransactions()	Acquires a value indicating whether data definition statements are ignored in transactions.
6	deletesAreDetected(int type)	Acquires a value indicating whether deletions of visible rows can be detected by calling the rowDeleted method of the ResultSet class.
7	doesMaxRowSizeIncludeBlobs()	Acquires a value indicating whether the getMaxRowSize method's return value contains the LONGVARCHAR or LONGVARBINARY SQL data type.
8	generatedKeyAlwaysReturned()	Acquires a value indicating whether a generated key will always be returned if the column names or indexes specified for the auto-generated key columns are valid and the statement succeeds.
9	getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	Acquires information related to a specified attribute of a specified type for user-defined types (UDTs) that can be used in specified schemas and catalogs.
10	getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	Acquires information about the optimum column set for a table in which rows are identified uniquely.

No.	Method in the DatabaseMetaData interface	Function
11	`getCatalogs()`	Acquires a catalog name.
12	`getCatalogSeparator()`	Acquires the separator between the catalog name and the table name.
13	`getCatalogTerm()`	Acquires a word recommended for `catalog`.
14	`getClientInfoProperties()`	Returns a list of the client information properties supported by the driver.
15	`getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)`	Acquires information about table column access permissions.
16	`getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)`	Acquires information about table columns.
17	`getConnection()`	Acquires the `Connection` instance that created this `DatabaseMetaData` instance.
18	`getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)`	Acquires cross-reference information between a specified referencing table and a specified referenced table.
19	`getDatabaseMajorVersion()`	Acquires the major version of the database (HADB server).
20	`getDatabaseMinorVersion()`	Acquires the minor version of the database (HADB server).
21	`getDatabaseProductName()`	Acquires the product name of the connected database (HADB server).
22	`getDatabaseProductVersion()`	Acquires the version of the connected database (HADB server).
23	`getDefaultTransactionIsolation()`	Acquires the default transaction isolation level for this database.
24	`getDriverMajorVersion()`	Acquires the JDBC driver's major version.
25	`getDriverMinorVersion()`	Acquires the JDBC driver's minor version.
26	`getDriverName()`	Acquires the JDBC driver's name.
27	`getDriverVersion()`	Acquires the JDBC driver's version.
28	`getExportedKeys(String catalog, String schema, String table)`	Acquires information about a referencing table's foreign keys.
29	`getExtraNameCharacters()`	Acquires all the special characters that can be used in an ID name that is not enclosed in double quotation marks (`"`).
30	`getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)`	Returns information about the function's parameters and the types that are returned.
31	`getFunctions(String catalog, String schemaPattern, String functionNamePattern)`	Returns information about the function.
32	`getIdentifierQuoteString()`	Acquires the character string used to enclose SQL identifiers.
33	`getImportedKeys(String catalog, String schema, String table)`	Acquires information about a referenced table's primary key.

No.	Method in the DatabaseMetaData interface	Function
34	`getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)`	Acquires information about indexes.
35	`getJDBCMajorVersion()`	Acquires the driver's JDBC major version.
36	`getJDBCMinorVersion()`	Acquires the driver's JDBC minor version.
37	`getMaxBinaryLiteralLength()`	Acquires the maximum number of hexadecimal characters that can be used in a binary literal.
38	`getMaxCatalogNameLength()`	Acquires the maximum length of a catalog name (number of characters).
39	`getMaxCharLiteralLength()`	Acquires the maximum length of an item of character string data (number of characters).
40	`getMaxColumnNameLength()`	Acquires the maximum length of a column name (number of characters).
41	`getMaxColumnsInGroupBy()`	Acquires the maximum number of grouping columns that can be specified in the `GROUP BY` clause.
42	`getMaxColumnsInIndex()`	Acquires the maximum number of columns that can comprise an index.
43	`getMaxColumnsInOrderBy()`	Acquires the maximum number of columns that can be specified in an `ORDER BY` clause.
44	`getMaxColumnsInSelect()`	Acquires the maximum number of selection expressions that can be specified in a selection list.
45	`getMaxColumnsInTable()`	Acquires the maximum number of columns that can be defined in a table.
46	`getMaxConnections()`	Acquires the maximum number of HADB clients that can connect concurrently to the HADB server.
47	`getMaxCursorNameLength()`	Acquires the maximum length of a cursor name (number of characters).
48	`getMaxIndexLength()`	Acquires the maximum length of an index key.
49	`getMaxLogicalLobSize()`	Acquires the maximum number of bytes this database allows as the logical size for a LOB.
50	`getMaxProcedureNameLength()`	Acquires the maximum length of a procedure name (number of characters).
51	`getMaxRowSize()`	Acquires the maximum length of a row (in bytes).
52	`getMaxSchemaNameLength()`	Acquires the maximum length of a schema name (number of characters).
53	`getMaxStatementLength()`	Acquires the maximum length of a character string that can be specified as an SQL statement.
54	`getMaxStatements()`	Acquires the maximum number of SQL statements that can be executed concurrently.
55	`getMaxTableNameLength()`	Acquires the maximum length of a table name (number of characters).
56	`getMaxTablesInSelect()`	Acquires the maximum number of tables that can be specified in a `SELECT` statement.
57	`getMaxUserNameLength()`	Acquires the maximum length of an authorization identifier (number of characters).

No.	Method in the DatabaseMetaData interface	Function
58	`getNumericFunctions()`	Acquires a list of the available mathematical functions (delimited by the comma).
59	`getPrimaryKeys(String catalog, String schema, String table)`	Acquires information about a specified table's primary key columns.
60	`getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)`	Acquires information about stored procedure parameters.
61	`getProcedures(String catalog, String schemaPattern, String procedureNamePattern)`	Acquires information about stored procedures.
62	`getProcedureTerm()`	Acquires a word recommended for `procedure`.
63	`getPseudoColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)`	Acquires a description of the pseudo or hidden columns in a particular table within the specified catalog and schema.
64	`getResultSetHoldability()`	Acquires the holdability of the `ResultSet` object.
65	`getRowIdLifetime()`	Indicates whether the `RowId` type is supported. If the `RowId` type is supported, the method also indicates the period during which the `RowId` object is valid.
66	`getSchemas()`	Acquires schema names.
67	`getSchemas(String catalog, String schemaPattern)`	Acquires schema names.
68	`getSchemaTerm()`	Acquires a word recommended for `schema`.
69	`getSearchStringEscape()`	Acquires the character string used as the escape sequence for wildcard characters.
70	`getSQLKeywords()`	Acquires a list (delimited by the comma) of all database-specific SQL keywords that are not SQL:2003 keywords.
71	`getSQLStateType()`	Acquires a value indicating whether `SQLSTATE` returned by the `getSQLState` method of the `SQLException` class is an X/Open (currently Open Group) SQL CLI or SQL:2003.
72	`getStringFunctions()`	Acquires a list of string functions (delimited by the comma).
73	`getSuperTables(String catalog, String schemaPattern, String tableNamePattern)`	Acquires information about table hierarchies defined in a specified schema.
74	`getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)`	Acquires information about the hierarchy of the user-defined types (UDT) that are defined in a specific schema.
75	`getSystemFunctions()`	Acquires the available system functions (delimited by the comma).
76	`getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)`	Acquires information about access privileges for a table.
77	`getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`	Acquires information about tables.
78	`getTableTypes()`	Acquires the table types.

No.	Method in the DatabaseMetaData interface	Function
79	`getTimeDateFunctions()`	Acquires a list of the available time and date functions (delimited by the comma).
80	`getTypeInfo()`	Acquires information about the default SQL types.
81	`getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)`	Acquires information about the user-defined types (UDTs).
82	`getURL()`	Acquires the URL that specifies information about the HADB server at the connection destination.
83	`getUserName()`	Acquires the authorization identifier used to connect to the HADB server.
84	`getVersionColumns(String catalog, String schema, String table)`	Acquires information about the table columns that are updated automatically when rows in the table are modified.
85	`insertsAreDetected(int type)`	Acquires a value indicating whether insertion of a visible row can be detected by calling the `rowInserted` method of the `ResultSet` class.
86	`isCatalogAtStart()`	Acquires a value indicating whether a catalog appears as the leading (or trailing) part of a fully qualified table name.
87	`isReadOnly()`	Acquires a value indicating whether the database is in the read-only mode.
88	`locatorsUpdateCopy()`	Acquires a value indicating whether a change was made to a copy of a LOB or directly to the LOB.
89	`nullPlusNonNullIsNull()`	Acquires a value indicating whether a join of a null value and a non-null value is treated as being a null value.
90	`nullsAreSortedAtEnd()`	Acquires a value indicating whether null values are sorted at the end regardless of the sort order.
91	`nullsAreSortedAtStart()`	Acquires a value indicating whether null values are sorted at the start regardless of the sort order.
92	`nullsAreSortedHigh()`	Acquires a value indicating whether null values are sorted in ascending order.
93	`nullsAreSortedLow()`	Acquires a value indicating whether null values are sorted low.
94	`othersDeletesAreVisible(int type)`	Acquires a value indicating whether a deletion performed externally is visible.
95	`othersInsertsAreVisible(int type)`	Acquires a value indicating whether an insertion performed externally is visible.
96	`othersUpdatesAreVisible(int type)`	Acquires a value indicating whether an updating performed externally is visible.
97	`ownDeletesAreVisible(int type)`	Acquires a value indicating whether a deletion of a result set itself is visible.
98	`ownInsertsAreVisible(int type)`	Acquires a value indicating whether an insertion of a result set itself is visible.
99	`ownUpdatesAreVisible(int type)`	Acquires a value indicating whether an updating of a result set itself is visible.
100	`storesLowerCaseIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all lowercase letters.

No.	Method in the DatabaseMetaData interface	Function
101	`storesLowerCaseQuotedIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all lowercase letters.
102	`storesMixedCaseIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in uppercase and lowercase letters.
103	`storesMixedCaseQuotedIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in uppercase and lowercase letters.
104	`storesUpperCaseIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all uppercase letters.
105	`storesUpperCaseQuotedIdentifiers()`	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all uppercase letters.
106	`supportsAlterTableWithAddColumn()`	Acquires a value indicating whether `ALTER TABLE` with added columns is supported.
107	`supportsAlterTableWithDropColumn()`	Acquires a value indicating whether `ALTER TABLE` with dropped columns is supported.
108	`supportsANSI92EntryLevelSQL()`	Acquires a value indicating whether the ANSI92 entry-level SQL grammar is supported.
109	`supportsANSI92FullSQL()`	Acquires a value indicating whether the ANSI92 full-level SQL grammar is supported.
110	`supportsANSI92IntermediateSQL()`	Acquires a value indicating whether the ANSI92 intermediate-level SQL grammar is supported.
111	`supportsBatchUpdates()`	Acquires a value indicating whether batch updating is supported.
112	`supportsCatalogsInDataManipulation()`	Acquires a value indicating whether catalog names can be used in data manipulation statements.
113	`supportsCatalogsInIndexDefinitions()`	Acquires a value indicating whether catalog names can be used in index definition statements.
114	`supportsCatalogsInPrivilegeDefinitions()`	Acquires a value indicating whether catalog names can be used in definition statements for granting privileges (`GRANT` statement) or revoking privileges (`REVOKE` statement).
115	`supportsCatalogsInProcedureCalls()`	Acquires a value indicating whether catalog names can be used in procedure call statements.
116	`supportsCatalogsInTableDefinitions()`	Acquires a value indicating whether catalog names can be used in table definition statements.
117	`supportsColumnAliasing()`	Acquires a value indicating whether aliases are supported for columns.
118	`supportsConvert()`	Acquires a value indicating whether the `CONVERT` function is supported for SQL types.
119	`supportsConvert(int fromType, int toType)`	Acquires a value indicating whether the `CONVERT` function is supported for specified SQL types.

No.	Method in the DatabaseMetaData interface	Function
120	supportsCoreSQLGrammar()	Acquires a value indicating whether the ODBC Core SQL grammar is supported.
121	supportsCorrelatedSubqueries()	Acquires a value indicating whether subqueries that contain external reference columns are supported.
122	supportsDataDefinitionAndDataManipulationTransactions()	Acquires a value indicating whether data definition statements and data manipulation statements are both supported in transactions.
123	supportsDataManipulationTransactionsOnly()	Acquires a value indicating whether only data manipulation statements are supported in transactions.
124	supportsDifferentTableCorrelationNames()	Acquires a value indicating whether the table names must be different from the correlation names when table correlation names are supported.
125	supportsExpressionsInOrderBy()	Acquires a value indicating whether value expressions are supported in an ORDER BY list.
126	supportsExtendedSQLGrammar()	Acquires a value indicating whether the ODBC Extended SQL grammar is supported.
127	supportsFullOuterJoins()	Acquires a value indicating whether nested full outer joins are supported.
128	supportsGetGeneratedKeys()	Acquires a value indicating whether automatic generation keys can be acquired after statements have executed.
129	supportsGroupBy()	Acquires a value indicating whether the GROUP BY clause form is supported.
130	supportsGroupByBeyondSelect()	Acquires a value indicating whether a column that is not in the SELECT statement can be used in the GROUP BY clause, provided that all columns in the SELECT statement are included in the GROUP BY clause.
131	supportsGroupByUnrelated()	Acquires a value indicating whether a column that is not in the SELECT statement can be used in the GROUP BY clause.
132	supportsIntegrityEnhancementFacility()	Acquires a value indicating whether the SQL Integrity Enhancement Facility is supported.
133	supportsLikeEscapeClause()	Acquires a value indicating whether the escape clause is supported in the LIKE clause.
134	supportsLimitedOuterJoins()	Acquires a value indicating whether limited support is provided for outer joins.
135	supportsMinimumSQLGrammar()	Acquires a value indicating whether the ODBC Minimum SQL grammar is supported.
136	supportsMixedCaseIdentifiers()	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being case sensitive, and then the results are stored in uppercase and lowercase letters.
137	supportsMixedCaseQuotedIdentifiers()	Acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being case sensitive, and then the results are stored in uppercase and lowercase letters.
138	supportsMultipleOpenResults()	Acquires a value indicating whether it is possible to have multiple ResultSet objects that have been returned by a CallableStatement object.

No.	Method in the DatabaseMetaData interface	Function
139	supportsMultipleResultSets()	Acquires a value indicating whether multiple ResultSet objects can be acquired from a single call of the execute method.
140	supportsMultipleTransactions()	Acquires a value indicating whether multiple transactions can be open at the same time (for different connections).
141	supportsNamedParameters()	Acquires a value indicating whether named parameters are supported for the CALL statement.
142	supportsNonNullableColumns()	Acquires a value indicating whether columns can be defined as non-null columns.
143	supportsOpenCursorsAcrossCommit()	Acquires a value indicating whether the cursor can remain open between commit operations.
144	supportsOpenCursorsAcrossRollback()	Acquires a value indicating whether the cursor can remain open between rollback operations.
145	supportsOpenStatementsAcrossCommit()	Acquires a value indicating whether the statement handle can remain open between commit operations.
146	supportsOpenStatementsAcrossRollback()	Acquires a value indicating whether the statement handle can remain open between rollback operations.
147	supportsOrderByUnrelated()	Acquires a value indicating whether a column that is not in the SELECT statement can be used in the ORDER BY clause.
148	supportsOuterJoins()	Acquires a value indicating whether some form of outer join is supported.
149	supportsPositionedDelete()	Acquires a value indicating whether positioned DELETE statements are supported.
150	supportsPositionedUpdate()	Acquires a value indicating whether positioned UPDATE statements are supported.
151	supportsRefCursors()	Acquires a value indicating whether the database supports REF CURSOR.
152	supportsResultSetConcurrency(int type, int concurrency)	Acquires a value indicating whether the combination of a specified result set type and a specified concurrent processing type is supported.
153	supportsResultSetHoldability(int holdability)	Acquires a value indicating whether holdability is supported for the specified ResultSet object.
154	supportsResultSetType(int type)	Acquires a value indicating whether a specified result set type is supported.
155	supportsSavepoints()	Acquires a value indicating whether save points are supported.
156	supportsSchemasInDataManipulation()	Acquires a value indicating whether schema names can be used in data manipulation statements.
157	supportsSchemasInIndexDefinitions()	Acquires a value indicating whether schema names can be used in index definition statements.
158	supportsSchemasInPrivilegeDefinitions()	Acquires a value indicating whether schema names can be used in definition statements for granting privileges (GRANT statement) or revoking privileges (REVOKE statement).
159	supportsSchemasInProcedureCalls()	Acquires a value indicating whether schema names can be used in procedure call statements.
160	supportsSchemasInTableDefinitions()	Acquires a value indicating whether schema names can be used in table definition statements.

No.	Method in the DatabaseMetaData interface	Function
161	supportsSelectForUpdate()	Acquires a value indicating whether SELECT FOR UPDATE statements are supported.
162	supportsStatementPooling()	Acquires a value indicating whether pooling of statement handles is supported.
163	supportsStoredFunctionsUsingCallSyntax()	Acquires a value indicating whether user-defined functions or vendor functions that use a stored procedure escape syntax are supported.
164	supportsStoredProcedures()	Acquires a value indicating whether stored procedure calls that use a stored procedure escape syntax are supported.
165	supportsSubqueriesInComparisons()	Acquires a value indicating whether subqueries are supported in comparison predicates.
166	supportsSubqueriesInExists()	Acquires a value indicating whether subqueries are supported in EXISTS predicates.
167	supportsSubqueriesInIns()	Acquires a value indicating whether subqueries are supported in IN predicates.
168	supportsSubqueriesInQuantifieds()	Acquires a value indicating whether subqueries are supported in quantified predicates.
169	supportsTableCorrelationNames()	Acquires a value indicating whether subqueries are supported in quantified predicates.
170	supportsTransactionIsolationLevel(int level)	Returns a value indicating whether a specified transaction isolation level is supported.
171	supportsTransactions()	Acquires a value indicating whether transactions are supported.
172	supportsUnion()	Acquires a value indicating whether SQL UNION is supported.
173	supportsUnionAll()	Acquires a value indicating whether SQL UNION ALL is supported.
174	updatesAreDetected(int type)	Acquires a value indicating whether updating performed on visible rows can be detected by calling the rowUpdated method of the ResultSet class.
175	usesLocalFilePerTable()	Acquires a value indicating whether a file is to be used for each table.
176	usesLocalFiles()	Acquires a value indicating whether tables are to be stored in local files.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an SQLException might be thrown.

(3) Required package name and class name

The package and class names required in order to use the DatabaseMetaData interface are as follows:

- Package name: com.hitachi.hadb.jdbc
- Class name: AdbDatabaseMetaData

(4) Special characters that can be specified in pattern character strings

Some methods in the `DatabaseMetaData` class use a `String` pattern character string as an argument. The following table shows the special characters that can be specified in such a pattern character string.

Table 8-42: Special characters that can be specified in pattern character strings

Special character that can be specified	Description
_ (underscore)	Any single character
%	A character string of any length, including no characters.
\	An escape character. Special characters that immediately follow the escape character in a pattern character string are treated as normal characters. The character \ is represented by the Shift-JIS code 0x5c (0x5c00 in UTF-16LE). In UTF-8, specify the character that appears as a backslash \.

8.6.2 allProceduresAreCallable()

(1) Function

This method acquires a value indicating whether all the procedures returned by the `getProcedures` method can be called by the current HADB user.

(2) Format

```
public synchronized boolean allProceduresAreCallable() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.3 allTablesAreSelectable()

(1) Function

This method acquires a value indicating whether all the tables returned by the `getTables` method can be used by the current HADB user.

(2) Format

```
public synchronized boolean allTablesAreSelectable() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.4 autoCommitFailureClosesAllResultSets()

(1) Function

This method returns a value indicating whether all open `ResultSet` objects are to be closed if an `SQLException` occurs while the automatic commit mode is enabled.

(2) Format

```
public synchronized boolean autoCommitFailureClosesAllResultSets() throws SQLExceptio
n
```

(3) Arguments

None.

(4) Return value

The method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.5 dataDefinitionCausesTransactionCommit()

(1) Function

This method acquires a value indicating whether a data definition statement in a transaction is to forcibly commit the transaction.

(2) Format

```
public synchronized boolean dataDefinitionCausesTransactionCommit() throws SQLExcepti
on
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.6 dataDefinitionIgnoredInTransactions()

(1) Function

This method acquires a value indicating whether data definition statements are ignored in transactions.

(2) Format

```
public synchronized boolean dataDefinitionIgnoredInTransactions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.7 deletesAreDetected(int type)

(1) Function

This method acquires a value indicating whether deletions of visible rows can be detected by calling the `rowDeleted` method of the `ResultSet` class.

(2) Format

```
public synchronized boolean deletesAreDetected(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.8 doesMaxRowSizeIncludeBlobs()

(1) Function

This method acquires a value indicating whether the `getMaxRowSize` method's return value contains the `LONGVARCHAR` or `LONGVARBINARY` SQL data type.

(2) Format

```
public synchronized boolean doesMaxRowSizeIncludeBlobs() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.9 generatedKeyAlwaysReturned()

(1) Function

This method acquires a value indicating whether a generated key will always be returned if the column names or indexes specified for the auto-generated key columns are valid and the statement succeeds.

(2) Format

```
public synchronized boolean generatedKeyAlwaysReturned() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If the `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.10 getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)

(1) Function

This method acquires information related to a specified attribute of a specified type for user-defined types (UDTs) that can be used in specified schemas and catalogs.

(2) Format

```
public synchronized ResultSet getAttributes(String catalog, String schemaPattern, Str
ing typeNamePattern, String attributeNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is ignored, if specified.

`String typeNamePattern`

Specifies a type name pattern. This argument is ignored, if specified.

`String attributeNamePattern`

Specifies an attribute name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-43: Format of the ResultSet object that is returned (getAttributes method)

Column No.	Type	Column name	Description
1	`String`	`TYPE_CAT`	Catalog name
2	`String`	`TYPE_SCHEM`	Schema name
3	`String`	`TYPE_NAME`	Type name
4	`String`	`ATTR_NAME`	Attribute name
5	`int`	`DATA_TYPE`	Attribute type of SQL type
6	`String`	`ATTR_TYPE_NAME`	Type name
7	`int`	`ATTR_SIZE`	Column size
8	`int`	`DECIMAL_DIGITS`	Decimal places
9	`int`	`NUM_PREC_RADIX`	Radix
10	`int`	`NULLABLE`	Whether the `NULL` value is permitted
11	`String`	`REMARKS`	Comment column
12	`String`	`ATTR_DEF`	Default value
13	`int`	`SQL_DATA_TYPE`	Not used
14	`int`	`SQL_DATETIME_SUB`	Not used
15	`int`	`CHAR_OCTET_LENGTH`	Maximum length (in bytes) of a `CHAR`-type column
16	`int`	`ORDINAL_POSITION`	Column number
17	`String`	`IS_NULLABLE`	Whether the `NULL` value is permitted
18	`String`	`SCOPE_CATALOG`	Catalog name for a table in the scope of reference attribute
19	`String`	`SCOPE_SCHEMA`	Schema name for a table in the scope of reference attribute
20	`String`	`SCOPE_TABLE`	Name for a table in the scope of reference attribute
21	`short`	`SOURCE_DATA_TYPE`	Source data type for individual types, user-defined `Ref` type, or `java.sql.Types` SQL type

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.11 getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)

(1) Function

This method acquires information about the optimum column set for a table in which rows are identified uniquely.

(2) Format

```
public synchronized ResultSet getBestRowIdentifier(String catalog, String schema, Str
ing table, int scope, boolean nullable) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schema`

Specifies a schema name. This argument is ignored, if specified.

`String table`

Specifies a table name. This argument is ignored, if specified.

`int scope`

Specifies a target scale. This argument is ignored, if specified.

`boolean nullable`

Specifies whether a column in which the null value can be specified is included. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-44: Format of the ResultSet object that is returned (getBestRowIdentifier method)

Column No.	Type	Column name	Description
1	short	SCOPE	Actual scale of the result
2	String	COLUMN_NAME	Column name
3	int	DATA_TYPE	SQL type
4	String	TYPE_NAME	Type name
5	int	COLUMN_SIZE	Precision
6	int	BUFFER_LENGTH	Not used
7	short	DECIMAL_DIGITS	Decimal places
8	short	PSEUDO_COLUMN	Whether this is a pseudo column

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.12 getCatalogs()

(1) Function

This method acquires a catalog name.

(2) Format

```
public synchronized ResultSet getCatalogs() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-45: Format of the ResultSet object that is returned (getCatalogs method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.13 getCatalogSeparator()

(1) Function

This method acquires the separator between the catalog name and the table name.

(2) Format

```
public synchronized String getCatalogSeparator() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a null character string.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.14 getCatalogTerm()

(1) Function

This method acquires a word recommended for `catalog`.

(2) Format

```
public synchronized String getCatalogTerm() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a null character string.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.15 getClientInfoProperties()

(1) Function

This method returns a list of the client information properties supported by the driver.

(2) Format

```
public synchronized ResultSet getClientInfoProperties() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a `ResultSet` object that contains no rows of retrieval results. The following table shows the format of the `ResultSet` object that is returned.

Table 8-46: Format of the ResultSet object that is returned

Column No.	Type	Column name	Description
1	String	NAME	Name of the client information property
2	int	MAX_LEN	Maximum length of the property's value
3	String	DEFAULT_VALUE	Property's default value
4	String	DESCRIPTION	Description of the property

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.16 getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)

(1) Function

This method acquires information about table column access permissions.

(2) Format

```
public synchronized ResultSet getColumnPrivileges(String catalog, String schema, Stri
ng table, String columnNamePattern) throws SQLException
```

(3) Arguments

String catalog

Specifies a catalog name. This argument is ignored, if specified.

String schema

Specifies a schema name. This argument is ignored, if specified.

String table

Specifies a table name. This argument is ignored, if specified.

String columnNamePattern

Specifies a column name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a ResultSet object that contains no retrieval result rows. The following table shows the format of the ResultSet object that is returned.

Table 8-47: Format of the ResultSet object that is returned (getColumnPrivileges method)

Column No.	Type	Column name	Description
1	String	TABLE_CAT	Catalog name
2	String	TABLE_SCHEM	Schema name

Column No.	Type	Column name	Description
3	String	TABLE_NAME	Table name
4	String	COLUMN_NAME	Column name
5	String	GRANTOR	User who grants access privilege
6	String	GRANTEE	User who receives access privilege
7	String	PRIVILEGE	Name of access privilege
8	String	IS_GRANTABLE	Whether a user who has received an access privilege can grant access privileges to other HADB users

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.17 getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)

(1) Function

This method acquires information about table columns.

The information about table columns that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the table column information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getColumns(String catalog, String schemaPattern, String
 tableNamePattern, String columnNamePattern) throws SQLException
```

(3) Arguments

String catalog

Specifies a catalog name. This argument is ignored, if specified.

String schemaPattern

Specifies a schema name pattern#. This value is case sensitive.

String tableNamePattern

Specifies a table name pattern#. This value is case sensitive.

String columnNamePattern

Specifies a column name pattern#. This value is case sensitive.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-48: Format of the ResultSet object that is returned (getColumns method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	The method always returns a null character string.
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Table name
4	`String`	`COLUMN_NAME`	Column name
5	`int`	`DATA_TYPE`	SQL type
6	`String`	`TYPE_NAME`	Type name
7	`int`	`COLUMN_SIZE`	Column size
8	`int`	`BUFFER_LENGTH`	The null value is always returned.
9	`int`	`DECIMAL_DIGITS`	Decimal places
10	`int`	`NUM_PREC_RADIX`	Radix • Approximate value: `2` • Exact numeric value: `10` • Non-numeric value: `0`
11	`int`	`NULLABLE`	Value indicating whether the `NULL` value is permitted: • `columnNoNulls`: The null value might not be permitted. • `columnNullable`: The null value can be used.
12	`String`	`REMARKS`	Comment column The method always returns a null character string.
13	`String`	`COLUMN_DEF`	The default value of the column is returned. • If the returned value is enclosed by single quotation marks (`'`), it means that the default value is a character string. • If `NULL` is specified as the default value of the column, the character string `NULL` is returned without single quotation marks (`'`). • If no default value is specified for the column, a null value is returned. • For details about other return values, see the description of the `DEFAULT_VALUE` column in the table *Content of SQL_COLUMNS* in the *HADB Setup and Operation Guide*.
14	`int`	`SQL_DATA_TYPE`	The null value is always returned.
15	`int`	`SQL_DATETIME_SUB`	The null value is always returned.
16	`int`	`CHAR_OCTET_LENGTH`	Maximum length (in bytes) of a `CHAR`-type column
17	`int`	`ORDINAL_POSITION`	Column number (beginning at 1)
18	`String`	`IS_NULLABLE`	Value indicating whether the `NULL` value is permitted: • `"NO"`: The null value cannot be used.

Column No.	Type	Column name	Description
			• "YES": The null value might be permitted.
19	String	SCOPE_CATALOG	The method always returns a null character string.
20	String	SCOPE_SCHEMA	The method always returns a null character string.
21	String	SCOPE_TABLE	The method always returns a null character string.
22	short	SOURCE_DATA_TYPE	The null value is always returned.
23	String	IS_AUTOINCREMENT	Whether columns are incremented automatically: • NO: Columns are not incremented automatically. NO is always returned.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

(6) Notes

The value acquired as column information of the viewed table might differ between the following two methods:

1. The DECIMAL_DIGITS value of the ResultSet object acquired by this method
2. The value acquired by the getScale method of the ResultSetMetaData interface

The first of these methods acquires the scaling value that was in effect when the viewed table was defined. The second of these methods acquires the scaling value that was in effect when the viewed table was retrieved. These two values will differ if the value specified for the adb_sql_prep_dec_div_rs_prior operand in the server definition or the client definition when the table was defined differs from the value specified in this operand when the table was retrieved.

8.6.18 getConnection()

(1) Function

This method acquires the Connection instance that created this DatabaseMetaData instance.

(2) Format

```
public synchronized Connection getConnection() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns a Connection object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.19 getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)

(1) Function

This method acquires cross-reference information between a specified referencing table and a specified referenced table.

The cross-reference information that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between privileges and the information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getCrossReference
       (String parentCatalog,String parentSchema, String parentTable,
        String foreignCatalog, String foreignSchema, String foreignTable)
       throws SQLException
```

(3) Arguments

`String parentCatalog`

Specifies the catalog name of the referenced table. This argument is ignored, if specified.

`String parentSchema`

Specifies the schema name pattern# of the referenced table.

`String parentTable`

Specifies the table name pattern# of the referenced table.

`String foreignCatalog`

Specifies the catalog name of the referencing table. This argument is ignored, if specified.

`String foreignSchema`

Specifies the schema name pattern# of the referencing table.

`String foreignTable`

Specifies the table name pattern# of the referencing table.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The following table shows the format of the `ResultSet` object that is returned.

Table 8-49: Format of the ResultSet object that is returned (getCrossReference method)

Column No.	Type	Column name	Description
1	String	PKTABLE_CAT	Catalog name of the referenced table The method always returns a null character string.
2	String	PKTABLE_SCHEM	Schema name of the referenced table
3	String	PKTABLE_NAME	Table name of the referenced table
4	String	PKCOLUMN_NAME	Column name of primary key
5	String	FKTABLE_CAT	Catalog name of the referencing table The method always returns a null character string.
6	String	FKTABLE_SCHEM	Schema name of the referencing table
7	String	FKTABLE_NAME	Table name of the referencing table
8	String	FKCOLUMN_NAME	Column name of foreign key
9	short	KEY_SEQ	Sequence number of foreign key
10	short	UPDATE_RULE	Operation that is applied to foreign keys when the primary key is updated • importedKeyNoAction: The primary key cannot be updated. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be updated.
11	short	DELETE_RULE	Operation that is applied to foreign keys when the primary key is deleted • importedKeyNoAction: The primary key cannot be deleted. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be deleted.
12	String	FK_NAME	Constraint name of referential constraints
13	String	PK_NAME	Index name of the primary key
14	short	DEFERRABILITY	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed • importedKeyNotDeferrable: Cannot be postponed.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.20 getDatabaseMajorVersion()

(1) Function

This method acquires the major version of the database (HADB server).

(2) Format

```
public synchronized int getDatabaseMajorVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the HADB server's major version.

For example, if the HADB server's version is 01-00, the method returns `1` of the `int` type.

This return value is the same as the server's major version displayed in the header of SQL traces output by the JDBC driver.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.21 getDatabaseMinorVersion()

(1) Function

This method acquires the minor version of the database (HADB server).

(2) Format

```
public synchronized int getDatabaseMinorVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the HADB server's minor version.

For example, if the HADB server's version is 01-00, the method returns `0` of the `int` type.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.22 getDatabaseProductName()

(1) Function

This method acquires the product name of the connected database (HADB server).

(2) Format

```
public synchronized String getDatabaseProductName() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns `"Hitachi Advanced Data Binder"`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.23 getDatabaseProductVersion()

(1) Function

This method acquires the version of the connected database (HADB server).

(2) Format

```
public synchronized String getDatabaseProductVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns the HADB server's version in the format "*vv*-*rr*" (example: `"01-00"`).

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.24 getDefaultTransactionIsolation()

(1) Function

This method acquires the default transaction isolation level for this database.

(2) Format

```
public synchronized int getDefaultTransactionIsolation() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `Connection.TRANSACTION_READ_COMMITTED` (the default transaction isolation level is `READ COMMITTED`).

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.25 getDriverMajorVersion()

(1) Function

This method acquires the JDBC driver's major version.

(2) Format

```
public synchronized int getDriverMajorVersion()
```

(3) Arguments

None.

(4) Return value

The method returns the JDBC driver's major version.

For example, if the JDBC driver's version is 01-00, the method returns `1`.

(5) Exceptions

None.

8.6.26 getDriverMinorVersion()

(1) Function

This method acquires the JDBC driver's minor version.

(2) Format

```
public synchronized int getDriverMinorVersion()
```

(3) Arguments

None.

(4) Return value

The method returns the JDBC driver's minor version.

For example, if the JDBC driver's version is 01-00, the method returns `0`.

(5) Exceptions

None.

8.6.27 getDriverName()

(1) Function

This method acquires the JDBC driver's name.

(2) Format

```
public synchronized String getDriverName() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns "`Hitachi Advanced Data Binder JDBC Driver`".

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.28 getDriverVersion()

(1) Function

This method acquires the JDBC driver's version.

(2) Format

```
public synchronized String getDriverVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns the JDBC driver's version in the format "*vv*-*rr*" (example: "`01-00`").

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.29 getExportedKeys(String catalog, String schema, String table)

(1) Function

This method acquires information about a referencing table's foreign keys.

The information about foreign keys that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between privileges and the information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getExportedKeys(String catalog, String schema, String t
able) throws SQLException
```

(3) Arguments

`String catalog`

Specifies the catalog name of the referenced table. This argument is ignored, if specified.

`String schema`

Specifies the schema name pattern# of the referenced table.

`String table`

Specifies the table name pattern[#] of the referenced table.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The following table shows the format of the `ResultSet` object that is returned.

Table 8-50: Format of the ResultSet object that is returned (getExportedKeys method)

Column No.	Type	Column name	Description
1	String	PKTABLE_CAT	Catalog name of the referenced table. The method always returns a null character string.
2	String	PKTABLE_SCHEM	Schema name of the referenced table
3	String	PKTABLE_NAME	Table name of the referenced table
4	String	PKCOLUMN_NAME	Column name of primary key
5	String	FKTABLE_CAT	Catalog name of the referencing table. The method always returns a null character string.
6	String	FKTABLE_SCHEM	Schema name of the referencing table
7	String	FKTABLE_NAME	Table name of the referencing table
8	String	FKCOLUMN_NAME	Column name of foreign key
9	short	KEY_SEQ	Sequence number of foreign key
10	short	UPDATE_RULE	Operation that is applied to foreign keys when the primary key is updated • importedKeyNoAction: The primary key cannot be updated. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be updated.
11	short	DELETE_RULE	Operation that is applied to foreign keys when the primary key is deleted • importedKeyNoAction: The primary key cannot be deleted. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be deleted.
12	String	FK_NAME	Constraint name of referential constraints
13	String	PK_NAME	Index name of the primary key
14	short	DEFERRABILITY	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed • importedKeyNotDeferrable: Cannot be postponed.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.30 getExtraNameCharacters()

(1) Function

This method acquires all the special characters that can be used in an ID name that is not enclosed in double quotation marks (").

(2) Format

```
public synchronized String getExtraNameCharacters() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns `"\@#"`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.31 getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)

(1) Function

This method returns information about the function's parameters and the types that are returned.

(2) Format

```
public synchronized ResultSet getFunctionColumns(String catalog, String schemaPattern
, String functionNamePattern, String columnNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is case sensitive.

`String functionNamePattern`

Specifies a function name pattern. This argument is case sensitive.

`String columnNamePattern`

Specifies a parameter name pattern. This argument is case sensitive.

(4) Return value

This method always returns a `ResultSet` object that contains no rows of retrieval results. The following table shows the format of the `ResultSet` object that is returned.

Table 8-51: Format of the ResultSet object that is returned

Column No.	Type	Column name	Description
1	`String`	`FUNCTION_CAT`	Catalog name
2	`String`	`FUNCTION_SCHEM`	Authorization identifier name
3	`String`	`FUNCTION_NAME`	Function name
4	`String`	`COLUMN_NAME`	Column or parameter name
5	`short`	`COLUMN_TYPE`	Column type or parameter
6	`int`	`DATA_TYPE`	Parameter's SQL type
7	`String`	`TYPE_NAME`	Parameter's SQL type name
8	`int`	`PRECISION`	Parameter's precision
9	`int`	`LENGTH`	Parameter size
10	`short`	`SCALE`	Parameter scaling (number of decimal places)
11	`short`	`RADIX`	Radix of parameter
12	`short`	`NULLABLE`	Whether the null value can be used
13	`String`	`REMARKS`	Comment related to the parameter
14	`int`	`CHAR_OCTET_LENGTH`	Maximum length of a binary or character-based parameter or column
15	`int`	`ORDINAL_POSITION`	Order of input and output parameters (beginning with 1) • For a function's return value, 0 is returned.
16	`String`	`IS_NULLABLE`	Whether the null value is permitted in the parameter or column
17	`String`	`SPECIFIC_NAME`	Name that uniquely identifies this function within the schema

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.32 getFunctions(String catalog, String schemaPattern, String functionNamePattern)

(1) Function

This method returns information about the function.

(2) Format

```
public synchronized ResultSet getFunctions(String catalog, String schemaPattern, Stri
ng functionNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is case sensitive.

`String functionNamePattern`

Specifies a function name pattern. This argument is case sensitive.

(4) Return value

This method always returns a `ResultSet` object that contains no rows of retrieval results The following table shows the format of the `ResultSet` object that is returned.

Table 8-52: Format of the ResultSet object that is returned

Column No.	Type	Column name	Description
1	`String`	`FUNCTION_CAT`	Catalog name
2	`String`	`FUNCTION_SCHEM`	Authorization identifier name
3	`String`	`FUNCTION_NAME`	Function name
4	`String`	`REMARKS`	Description of function
5	`short`	`FUNCTION_TYPE`	Function type
6	`String`	`SPECIFIC_NAME`	Name that uniquely identifies this function within the schema

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.33 getIdentifierQuoteString()

(1) Function

This method acquires the character string used to enclose SQL identifiers.

(2) Format

```
public synchronized String getIdentifierQuoteString() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

The method returns double quotation marks (").

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.34 getImportedKeys(String catalog, String schema, String table)

(1) Function

This method acquires information about a referenced table's primary key.

The information about primary keys that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between privileges and the information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getImportedKeys(String catalog, String schema, String t
able) throws SQLException
```

(3) Arguments

`String catalog`

Specifies the catalog name of the referencing table. This argument is ignored, if specified.

`String schema`

Specifies the schema name pattern# of the referencing table.

`String table`

Specifies the table name pattern# of the referencing table.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The following table shows the format of the `ResultSet` object that is returned.

Table 8-53: Format of the ResultSet object that is returned (getImportedKeys method)

Column No.	Type	Column name	Description
1	String	PKTABLE_CAT	Catalog name of the referenced table. The method always returns a null character string.
2	String	PKTABLE_SCHEM	Schema name of the referenced table
3	String	PKTABLE_NAME	Table name of the referenced table
4	String	PKCOLUMN_NAME	Column name of primary key
5	String	FKTABLE_CAT	Catalog name of the referencing table. The method always returns a null character string.
6	String	FKTABLE_SCHEM	Schema name of the referencing table
7	String	FKTABLE_NAME	Table name of the referencing table
8	String	FKCOLUMN_NAME	Column name of foreign key
9	short	KEY_SEQ	Sequence number of foreign key
10	short	UPDATE_RULE	Operation that is applied to foreign keys when the primary key is updated • importedKeyNoAction: The primary key cannot be updated. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be updated.
11	short	DELETE_RULE	Operation that is applied to foreign keys when the primary key is deleted • importedKeyNoAction: The primary key cannot be deleted. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key can be deleted.
12	String	FK_NAME	Constraint name of referential constraints
13	String	PK_NAME	Index name of the primary key
14	short	DEFERRABILITY	Whether evaluation of constraints on the foreign key can be postponed until the transaction is committed • importedKeyNotDeferrable: Cannot be postponed.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.35 getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)

(1) Function

This method acquires information about indexes.

The information about indexes that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the index information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getIndexInfo(String catalog, String schema, String tabl
e, boolean unique, boolean approximate) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schema`

Specifies a schema name pattern#. This value is case sensitive.

`String table`

Specifies a table name pattern#. This value is case sensitive.

`boolean unique`

Specifies the unique attribute. Specify one of the following values:

`true`: Acquires information for unique indexes only.

`false`: Acquires all index information.

`boolean approximate`

You do not need to specify this value. If specified, the specification is ignored.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The method returns a `ResultSet` object.

The following table shows the format of the `ResultSet` object that is returned.

Table 8-54: Format of the ResultSet object that is returned (getIndexInfo method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name. The method always returns a null character string.
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Table name
4	`boolean`	`NON_UNIQUE`	If the key values for which the index is defined (total value of one or multiple columns defined as the index) are different in every row, the method returns `false`; otherwise, the method returns `true`.
5	`String`	`INDEX_QUALIFIER`	Index's catalog name. The method always returns a null character string.

Column No.	Type	Column name	Description
6	String	INDEX_NAME	Index identifier
7	short	TYPE	Index type. The method always returns DatabaseMetaData.tableIndexOther.
8	short	ORDINAL_POSITION	The null value is always returned.
9	String	COLUMN_NAME	The method always returns a null character string.
10	String	ASC_OR_DESC	The method always returns a null character string.
11	int	CARDINALITY	Number of unique values in the index. The null value is always returned.
12	int	PAGES	Number of pages used for the index. The null value is always returned.
13	String	FILTER_CONDITION	Filter condition. The method always returns a null character string.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.36 getJDBCMajorVersion()

(1) Function

This method acquires the driver's JDBC major version.

(2) Format

```
public synchronized int getJDBCMajorVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the JDBC major version. A value of 4 is returned.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.37 getJDBCMinorVersion()

(1) Function

This method acquires the driver's JDBC minor version.

(2) Format

```
public synchronized int getJDBCMinorVersion() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the JDBC minor version.

If the JDBC driver is for JRE 8, `2` is returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.38 getMaxBinaryLiteralLength()

(1) Function

This method acquires the maximum number of hexadecimal characters that can be used in a binary literal.

(2) Format

```
public synchronized int getMaxBinaryLiteralLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `64000`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.39 getMaxCatalogNameLength()

(1) Function

This method acquires the maximum length of a catalog name (number of characters).

(2) Format

```
public synchronized int getMaxCatalogNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `0`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.40 getMaxCharLiteralLength()

(1) Function

This method acquires the maximum length of an item of character string data (number of characters).

(2) Format

```
public synchronized int getMaxCharLiteralLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for character string data. A value of `32000` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.41 getMaxColumnNameLength()

(1) Function

This method acquires the maximum length of a column name (number of characters).

(2) Format

```
public synchronized int getMaxColumnNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for a column name. A value of `100` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.42 getMaxColumnsInGroupBy()

(1) Function

This method acquires the maximum number of grouping columns that can be specified in the `GROUP BY` clause.

(2) Format

```
public synchronized int getMaxColumnsInGroupBy() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of grouping columns that can be specified in the `GROUP BY` clause. A value of `64` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.43 getMaxColumnsInIndex()

(1) Function

This method acquires the maximum number of columns that can comprise an index.

(2) Format

```
public synchronized int getMaxColumnsInIndex() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of columns that can comprise an index. A value of `16` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.44 getMaxColumnsInOrderBy()

(1) Function

This method acquires the maximum number of columns that can be specified in an `ORDER BY` clause.

(2) Format

```
public synchronized int getMaxColumnsInOrderBy() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of columns permitted in an `ORDER BY` clause. A value of `16` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.45 getMaxColumnsInSelect()

(1) Function

This method acquires the maximum number of selection expressions that can be specified in a selection list.

(2) Format

```
public synchronized int getMaxColumnsInSelect() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of selection expressions permitted in a selection list. A value of `1000` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.46 getMaxColumnsInTable()

(1) Function

This method acquires the maximum number of columns that can be defined in a table.

(2) Format

```
public synchronized int getMaxColumnsInTable() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of columns that can be defined for a table. A value of `1000` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.47 getMaxConnections()

(1) Function

This method acquires the maximum number of HADB clients that can connect concurrently to the HADB server.

(2) Format

```
public synchronized int getMaxConnections() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of HADB users that can connect concurrently to the HADB server.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.48 getMaxCursorNameLength()

(1) Function

This method acquires the maximum length of a cursor name (number of characters).

(2) Format

```
public synchronized int getMaxCursorNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for a cursor name. The method always returns `0`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.49 getMaxIndexLength()

(1) Function

This method acquires the maximum length of an index key.

(2) Format

```
public synchronized int getMaxIndexLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum length of an index key. A value of `4036` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.50 getMaxLogicalLobSize()

(1) Function

This method acquires the maximum number of bytes this database allows as the logical size for a LOB.

(2) Format

```
public synchronized long getMaxLogicalLobSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `0`.

(5) Exceptions

If the `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.51 getMaxProcedureNameLength()

(1) Function

This method acquires the maximum length of a procedure name (number of characters).

(2) Format

```
public synchronized int getMaxProcedureNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for a procedure name. The method always returns `0`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.52 getMaxRowSize()

(1) Function

This method acquires the maximum length of a row (in bytes).

(2) Format

```
public synchronized int getMaxRowSize() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `0`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.53 getMaxSchemaNameLength()

(1) Function

This method acquires the maximum length of a schema name (number of characters).

(2) Format

```
public synchronized int getMaxSchemaNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for a schema name. A value of `100` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.54 getMaxStatementLength()

(1) Function

This method acquires the maximum length of a character string that can be specified as an SQL statement.

(2) Format

```
public synchronized int getMaxStatementLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum length of a character string that can be specified as an SQL statement. A value of `16000000` (in characters) is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.55 getMaxStatements()

(1) Function

This method acquires the maximum number of SQL statements that can be executed concurrently.

The method acquires the maximum number of `Statement` objects that can be created by a single `Connection` object.

(2) Format

```
public synchronized int getMaxStatements() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of SQL statements that can be executed concurrently. A value of `4095` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.56 getMaxTableNameLength()

(1) Function

This method acquires the maximum length of a table name (number of characters).

(2) Format

```
public synchronized int getMaxTableNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for a table name. A value of `100` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.57 getMaxTablesInSelect()

(1) Function

This method acquires the maximum number of tables that can be specified in a `SELECT` statement.

(2) Format

```
public synchronized int getMaxTablesInSelect() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of tables that can be specified in a `SELECT` statement. A value of `64` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.58 getMaxUserNameLength()

(1) Function

This method acquires the maximum length of an authorization identifier (number of characters).

(2) Format

```
public synchronized int getMaxUserNameLength() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the maximum number of characters permitted for an authorization identifier. A value of `100` is always returned.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.59 getNumericFunctions()

(1) Function

This method acquires a list of the available mathematical functions (delimited by the comma).

(2) Format

```
public synchronized String getNumericFunctions() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.60 getPrimaryKeys(String catalog, String schema, String table)

(1) Function

This method acquires information about a specified table's primary key columns.

The information about primary keys that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the primary key information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getPrimaryKeys(String catalog, String schema, String ta
ble) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schema`

Specifies a schema name pattern#.

`String table`

Specifies a table name pattern#.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

This method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-55: Format of the ResultSet object that is returned (getPrimaryKeys method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name The method always returns a null character string.
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Table name
4	`String`	`COLUMN_NAME`	Column name
5	`short`	`KEY_SEQ`	Column's order number within the primary key
6	`String`	`PK_NAME`	Primary key name

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.61 getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)

(1) Function

This method acquires information about stored procedure parameters.

(2) Format

```
public synchronized ResultSet getProcedureColumns(String catalog, String schemaPatter
n, String procedureNamePattern, String columnNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is ignored, if specified.

`String procedureNamePattern`

Specifies a procedure name pattern. This argument is ignored, if specified.

`String columnNamePattern`

Specifies a parameter name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-56: Format of the ResultSet object that is returned (getProcedureColumns method)

Column No.	Type	Column name	Description
1	String	PROCEDURE_CAT	Catalog name
2	String	PROCEDURE_SCHEM	Schema name
3	String	PROCEDURE_NAME	Procedure name
4	String	COLUMN_NAME	Parameter name
5	short	COLUMN_TYPE	Parameter type
6	int	DATA_TYPE	Parameter's SQL type
7	String	TYPE_NAME	Parameter's SQL type name
8	int	PRECISION	Parameter's precision
9	int	LENGTH	Parameter size
10	short	SCALE	Parameter scaling
11	short	RADIX	Radix of parameter
12	short	NULLABLE	Whether the null value can be used
13	String	REMARKS	Comment related to the parameter
14	String	COLUMN_DEF	Column's default value
15	int	SQL_DATA_TYPE	Reserved for future use
16	int	SQL_DATETIME_SUB	Reserved for future use
17	int	CHAR_OCTET_LENGTH	Maximum length of a binary or character-based parameter or column
18	int	ORDINAL_POSITION	Order of input and output parameters (beginning with 1)
19	String	IS_NULLABLE	Whether the null value is permitted in the parameter or column
20	String	SPECIFIC_NAME	Name that uniquely identifies this procedure

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.62 getProcedures(String catalog, String schemaPattern, String procedureNamePattern)

(1) Function

This method acquires information about stored procedures.

(2) Format

```
public synchronized ResultSet getProcedures(String catalog, String schemaPattern, Str
ing procedureNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is ignored, if specified.

`String procedureNamePattern`

Specifies a procedure name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-57: Format of the ResultSet object that is returned (getProcedures method)

Column No.	Type	Column name	Description
1	String	PROCEDURE_CAT	Catalog name
2	String	PROCEDURE_SCHEM	Schema name
3	String	PROCEDURE_NAME	Procedure name
4	String	RESERVE1	Reserved for future use
5	String	RESERVE2	Reserved for future use
6	String	RESERVE3	Reserved for future use
7	String	REMARKS	Description of procedure
8	short	PROCEDURE_TYPE	Procedure type
9	String	SPECIFIC_NAME	Name that uniquely identifies this procedure

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.63 getProcedureTerm()

(1) Function

This method acquires a word recommended for `procedure`.

(2) Format

```
public synchronized String getProcedureTerm() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns a null character string.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.64 getPseudoColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern)

(1) Function

This method acquires a description of the pseudo or hidden columns in a particular table within the specified catalog and schema. This method ignores all arguments and always returns an empty result set.

(2) Format

```
public synchronized ResultSet getPseudoColumns(String catalog,String schemaPattern,St
ring tableNamePattern,String columnNamePattern) throws SQLException
```

(3) Arguments

`String catalog`:

Specifies the catalog name. However, any value you specify is ignored.

`String schemaPattern`:

Specifies a schema name pattern. However, any value you specify is ignored.

`String tableNamePattern`:

Specifies a table name pattern. However, any value you specify is ignored.

`String columnNamePattern`:

Specifies a column name pattern. However, any value you specify is ignored.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table lists the format of the returned `ResultSet` object.

Table 8-58: Format of returned ResultSet object (getPseudoColumns method)

Column No.	Data type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Table name
4	`String`	`COLUMN_NAME`	Column name
5	`int`	`DATA_TYPE`	SQL type from `java.sql.Types`
6	`int`	`COLUMN_SIZE`	Column size
7	`int`	`DECIMAL_DIGITS`	Number of decimal places
8	`int`	`NUM_PREC_RADIX`	Radix value
9	`String`	`COLUMN_USAGE`	Usage permitted for the column
10	`String`	`REMARKS`	Comment column
11	`int`	`CHAR_OCTET_LENGTH`	Maximum length (in bytes) of `char` type column
12	`String`	`IS_NULLABLE`	Whether a null value can be used

(5) Exceptions

If the `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.65 getResultSetHoldability()

(1) Function

This method acquires the holdability of the `ResultSet` object.

(2) Format

```
public synchronized int getResultSetHoldability() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method always returns `ResultSet.HOLD_CURSORS_OVER_COMMIT`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.66 getRowIdLifetime()

(1) Function

This method indicates whether the `RowId` type is supported. If the `RowId` type is supported, the method also indicates the period during which the `RowId` object is valid.

(2) Format

```
public synchronized RowIdLifetime getRowIdLifetime() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `RowIdLifetime. ROWID_UNSUPPORTED`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.67 getSchemas()

(1) Function

This method acquires schema names.

The information about schemas that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the schema information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getSchemas() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-59: Format of the ResultSet object that is returned (getSchemas method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_SCHEM`	Schema name
2	`String`	`TABLE_CATALOG`	The method always returns a null character string.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.68 getSchemas(String catalog, String schemaPattern)

(1) Function

This method acquires schema names.

The information about schemas that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the schema information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getSchemas(String catalog, String schemaPattern) throws
 SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is case sensitive.

For details about the special characters that can be specified in the pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

This method returns a `ResultSet` object.

The following table shows the format of the `ResultSet` object that is returned.

Table 8-60: Format of the ResultSet object that is returned

Column No.	Type	Column name	Description
1	String	TABLE_SCHEM	Schema name
2	String	TABLE_CATALOG	The method always returns a null character string.

8.6.69 getSchemaTerm()

(1) Function

This method acquires a word recommended for schema.

(2) Format

```
public synchronized String getSchemaTerm() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a String object.

"schema" is returned.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.70 getSearchStringEscape()

(1) Function

This method acquires the character string used as the escape sequence for wildcard characters.

Execute this method to acquire the character string used to escape wildcard characters specified in arguments that specify pattern character strings in methods of the DatabaseMetaData interface that acquire list information such as table and column lists.

(2) Format

```
public synchronized String getSearchStringEscape() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns \.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.71 getSQLKeywords()

(1) Function

This method acquires a list (delimited by the comma) of all database-specific SQL keywords that are not SQL:2003 keywords.

(2) Format

```
public synchronized String getSQLKeywords() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a String object.

For details about the reserved words, see the topic *Reserved words* in the manual *HADB SQL Reference*.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

(6) Notes

The reserved words returned by this method include reserved words that have been unregistered.

8.6.72 getSQLStateType()

(1) Function

This method acquires a value indicating whether SQLSTATE returned by the getSQLState method of the SQLException class is an X/Open (currently Open Group) SQL CLI or SQL:2003.

(2) Format

```
public synchronized int getSQLStateType() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `sqlStateSQL`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.73 getStringFunctions()

(1) Function

This method acquires a list of string functions (delimited by the comma).

(2) Format

```
public synchronized String getStringFunctions() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.74 getSuperTables(String catalog, String schemaPattern, String tableNamePattern)

(1) Function

This method acquires information about table hierarchies defined in a specific schema.

(2) Format

```
public synchronized ResultSet getSuperTables(String catalog, String schemaPattern, St
ring tableNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is ignored, if specified.

`String tableNamePattern`

Specifies a table name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-61: Format of the ResultSet object that is returned (getSuperTables method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Type name
4	`String`	`SUPERTABLE_NAME`	Super type name

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.75 getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)

(1) Function

This method acquires information about the hierarchy of the user-defined types (UDTs) that are defined in a specific schema.

(2) Format

```
public synchronized ResultSet getSuperTypes(String catalog, String schemaPattern, Str
ing typeNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This argument is ignored, if specified.

`String typeNamePattern`

Specifies a UDT (user-defined-type) name pattern. This argument is ignored, if specified.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-62: Format of the ResultSet object that is returned (getSuperTypes method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	UDT's catalog name
2	`String`	`TABLE_SCHEM`	UDT's schema name
3	`String`	`TABLE_NAME`	UDT's type name
4	`String`	`SUPERTYPE_CAT`	Catalog name of the direct super type
5	`String`	`SUPERTYPE_SCHEM`	Schema name of the direct super type
6	`String`	`SUPERTYPE_NAME`	Super type name of the direct super type

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.76 getSystemFunctions()

(1) Function

This method acquires the available system functions (delimited by the comma).

(2) Format

```
public synchronized String getSystemFunctions() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.77 getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)

(1) Function

This method acquires information about access privileges for a table.

The information about access privileges that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between the privileges and the table information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getTablePrivileges(String catalog, String schemaPattern
, String tableNamePattern) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern#.

`String tableNamePattern`

Specifies a table name pattern#.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The following table shows the format of the `ResultSet` object that is returned.

Table 8-63: Format of the ResultSet object that is returned (getTablePrivileges method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_CAT`	Catalog name. The method always returns a null character string.
2	`String`	`TABLE_SCHEM`	Schema name
3	`String`	`TABLE_NAME`	Table name
4	`String`	`GRANTOR`	User who grants access privilege

Column No.	Type	Column name	Description
5	String	GRANTEE	User who receives access privilege
6	String	PRIVILEGE	Permitted access privilege name: • "SELECT": SELECT privilege • "INSERT": INSERT privilege • "UPDATE": UPDATE privilege • "DELETE": DELETE privilege • "TRUNCATE": TRUNCATE privilege • "REFERENCES": REFERENCES privilege • "IMPORT TABLE": IMPORT TABLE privilege • "REBUILD INDEX": REBUILD INDEX privilege • "GET COSTINFO": GET COSTINFO privilege • "EXPORT TABLE": EXPORT TABLE privilege • "MERGE CHUNK": MERGE CHUNK privilege • "CHANGE CHUNK COMMENT": CHANGE CHUNK COMMENT privilege • "CHANGE CHUNK STATUS": CHANGE CHUNK STATUS privilege • "ARCHIVE CHUNK": ARCHIVE CHUNK privilege • "UNARCHIVE CHUNK": UNARCHIVE CHUNK privilege
7	String	IS_GRANTABLE	Whether a user who has received an access privilege can grant access privileges to other HADB users • YES: Access privileges can be granted to other HADB users. • NO: Access privileges cannot be granted to other HADB users.

(5) Exceptions

If this Connection object is closed before this method is executed, the JDBC driver throws an SQLException.

8.6.78 getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)

(1) Function

This method acquires information about tables.

The information about tables that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the table information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

(2) Format

```
public synchronized ResultSet getTables(String catalog, String schemaPattern, String
tableNamePattern, String[] types) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern#. This value is case sensitive.

`String tableNamePattern`

Specifies a table name pattern#. This value is case sensitive.

`String[] types`

Specifies a list of table types. Specify table types returned by the `getTableTypes` method. This value is case sensitive.

If `null` is specified, the method assumes that the types of all tables were specified.

#

For details about the special characters that can be specified in each pattern, see (4) Special characters that can be specified in pattern character strings in 8.6.1 List of the methods in the DatabaseMetaData interface.

(4) Return value

The method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-64: Format of the ResultSet object that is returned (getTables method)

Column No.	Type	Column name	Description
1	String	TABLE_CAT	The method always returns a null character string.
2	String	TABLE_SCHEM	Schema name
3	String	TABLE_NAME	Table name
4	String	TABLE_TYPE	Table type: • TABLE: Base table • VIEW: Viewed table • SYSTEM TABLE: Dictionary table or system table
5	String	REMARKS	The method always returns a null character string.
6	String	TYPE_CAT	The method always returns a null character string..
7	String	TYPE_SCHEM	The method always returns a null character string..
8	String	TYPE_NAME	The method always returns a null character string..
9	String	SELF_REFERENCING_COL_NAME	The method always returns a null character string..
10	String	REF_GENERATION	The method always returns a null character string..

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- The `Connection` object is closed before this method is executed.
- At least one element in the `String[] types` argument is `null`.

- At least one element in the `String[] types` argument is not one of the following character strings: `"TABLE","VIEW", "SYSTEM TABLE"`

8.6.79 getTableTypes()

(1) Function

This method acquires the table types.

(2) Format

```
public synchronized ResultSet getTableTypes() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-65: Format of the ResultSet object that is returned (getTableTypes method)

Column No.	Type	Column name	Description
1	`String`	`TABLE_TYPE`	Table type: • `TABLE`: Base table • `VIEW`: Viewed table • `SYSTEM TABLE`: Dictionary table or system table

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.80 getTimeDateFunctions()

(1) Function

This method acquires a list of the available time and date functions (delimited by the comma).

(2) Format

```
public synchronized String getTimeDateFunctions() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.81 getTypeInfo()

(1) Function

This method acquires information about the default SQL types.

(2) Format

```
public synchronized ResultSet getTypeInfo() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `ResultSet` object. The following table shows the format of the `ResultSet` object that is returned.

Table 8-66: Format of the ResultSet object that is returned (getTypeInfo method)

Column No.	Type	Column name	Description
1	`String`	`TYPE_NAME`	Type name
2	`short`	`DATA_TYPE`	SQL data type of `java.sql.Types`
3	`int`	`PRECISION`	Maximum precision
4	`String`	`LITERAL_PREFIX`	Prefix used to quote literals
5	`String`	`LITERAL_SUFFIX`	Suffix used to quote literals
6	`String`	`CREATE_PARAMS`	Parameter used to create types
7	`short`	`NULLABLE`	Whether the null value is permitted for this type. The method always returns `typeNullableUnknown`.
8	`boolean`	`CASE_SENSITIVE`	Whether the value is case sensitive: • `true`: Character string data types • `false`: Other data types
9	`short`	`SEARCHABLE`	Whether `WHERE` can be used for this type.

Column No.	Type	Column name	Description
			The method always returns `typeSearchable`.
10	`boolean`	`UNSIGNED_ATTRIBUTE`	Whether the attribute is unsigned. The method returns `false` for numeric data (because it is signed) and `true` for other data types (because they are unsigned).
11	`boolean`	`FIXED_PREC_SCALE`	Whether this can be a currency value. The method always returns `false`.
12	`boolean`	`AUTO_INCREMENT`	Whether this can be used as an automatic increment value. The method always returns `false`.
13	`String`	`LOCAL_TYPE_NAME`	The type name's localized version. The method returns the same value as the type name.
14	`short`	`MINIMUM_SCALE`	Supported minimum scale
15	`short`	`MAXIMUM_SCALE`	Supported maximum scale
16	`int`	`SQL_DATA_TYPE`	The null value is always returned.
17	`int`	`SQL_DATETIME_SUB`	The null value is always returned.
18	`int`	`NUM_PREC_RADIX`	`10` for numeric data, `0` for all other data.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.82 getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)

(1) Function

This method acquires information about the user-defined types (UDTs).

(2) Format

```
public synchronized ResultSet getUDTs(String catalog, String schemaPattern, String ty
peNamePattern, int[] types) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schemaPattern`

Specifies a schema name pattern. This value is case sensitive.

`String tableNamePattern`

Specifies a table name pattern. This value is case sensitive.

`int[] types`

Specifies a list of the user-defined types. This value is case sensitive.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-67: Format of the ResultSet object that is returned (getUDTs method)

Column No.	Type	Column name	Description
1	`String`	`TYPE_CAT`	Catalog name
2	`String`	`TYPE_SCHEM`	Schema name
3	`String`	`TYPE_NAME`	Type name
4	`String`	`CLASS_NAME`	Java class name
5	`int`	`DATA_TYPE`	Type value defined by `java.sql.Types`
6	`String`	`REMARKS`	Description of the type
7	`short`	`BASE_TYPE`	Type code of `DISTINCT`-type source or type code of an implementation of the `SELF_REFERENCING_COLUMN` user-defined reference type of a structure type defined by `java.sql.Types`

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.83 getURL()

(1) Function

This method acquires the URL that specifies information about the HADB server at the connection destination.

(2) Format

```
public synchronized String getURL() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

If there is no URL that specifies information about the connection destination, the method returns `null`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.84 getUserName()

(1) Function

This method acquires the authorization identifier used to connect to the HADB server.

(2) Format

```
public synchronized String getUserName() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns a `String` object.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.85 getVersionColumns(String catalog, String schema, String table)

(1) Function

This method acquires information about the table columns that are updated automatically when rows in the table are modified.

(2) Format

```
public synchronized ResultSet getVersionColumns(String catalog, String schema, String
 table) throws SQLException
```

(3) Arguments

`String catalog`

Specifies a catalog name. This argument is ignored, if specified.

`String schema`

Specifies a schema name. This value is case sensitive.

`String table`

Specifies a table name. This value is case sensitive.

(4) Return value

This method always returns a `ResultSet` object that contains no retrieval result rows. The following table shows the format of the `ResultSet` object that is returned.

Table 8-68: Format of the ResultSet object that is returned (getVersionColumns method)

Column No.	Type	Column name	Description
1	short	SCOPE	Not used
2	String	COLUMN_NAME	Column name
3	int	DATA_TYPE	SQL type
4	String	TYPE_NAME	Type name
5	int	COLUMN_SIZE	Precision
6	int	BUFFER_LENGTH	Length of column value (in bytes)
7	short	DECIMAL_DIGITS	Decimal places
8	short	PSEUDO_COLUMN	Whether this is a pseudo column

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.86 insertsAreDetected(int type)

(1) Function

This method acquires a value indicating whether insertion of a visible row can be detected by calling the `rowInserted` method of the `ResultSet` class.

(2) Format

```
public synchronized boolean insertsAreDetected(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.87 isCatalogAtStart()

(1) Function

This method acquires a value indicating whether a catalog appears as the leading (or trailing) part of a fully qualified table name.

(2) Format

```
public synchronized boolean isCatalogAtStart() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.88 isReadOnly()

(1) Function

This method acquires a value indicating whether the database is in the read-only mode.

(2) Format

```
public synchronized boolean isReadOnly() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.89 locatorsUpdateCopy()

(1) Function

This method acquires a value indicating whether a change was made to a copy of a LOB or directly to the LOB.

(2) Format

```
public synchronized boolean locatorsUpdateCopy() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.90 nullPlusNonNullIsNull()

(1) Function

This method acquires a value indicating whether a join of a null value and a non-null value is treated as being a null value.

(2) Format

```
public synchronized boolean nullPlusNonNullIsNull() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.91 nullsAreSortedAtEnd()

(1) Function

This method acquires a value indicating whether null values are sorted at the end regardless of the sort order.

(2) Format

```
public synchronized boolean nullsAreSortedAtEnd() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.92 nullsAreSortedAtStart()

(1) Function

This method acquires a value indicating whether null values are sorted at the start regardless of the sort order.

(2) Format

```
public synchronized boolean nullsAreSortedAtStart() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.93 nullsAreSortedHigh()

(1) Function

This method acquires a value indicating whether null values are sorted high.

(2) Format

```
public synchronized boolean nullsAreSortedHigh() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.94 nullsAreSortedLow()

(1) Function

This method acquires a value indicating whether null values are sorted in descending order.

(2) Format

```
public synchronized boolean nullsAreSortedLow() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.95 othersDeletesAreVisible(int type)

(1) Function

This method acquires a value indicating whether a deletion performed externally is visible.

(2) Format

```
public synchronized boolean othersDeletesAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.96 othersInsertsAreVisible(int type)

(1) Function

This method acquires a value indicating whether an insertion performed externally is visible.

(2) Format

```
public synchronized boolean othersInsertsAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.97 othersUpdatesAreVisible(int type)

(1) Function

This method acquires a value indicating whether an updating performed externally is visible.

(2) Format

```
public synchronized boolean othersUpdatesAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.98 ownDeletesAreVisible(int type)

(1) Function

This method acquires a value indicating whether a deletion of a result set itself is visible.

(2) Format

```
public synchronized boolean ownDeletesAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.99 ownInsertsAreVisible(int type)

(1) Function

This method acquires a value indicating whether an insertion of a result set itself is visible.

(2) Format

```
public synchronized boolean ownInsertsAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.100 ownUpdatesAreVisible(int type)

(1) Function

This method acquires a value indicating whether an updating of a result set itself is visible.

(2) Format

```
public synchronized boolean ownUpdatesAreVisible(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.101 storesLowerCaseIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all lowercase letters.

(2) Format

```
public synchronized boolean storesLowerCaseIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.102 storesLowerCaseQuotedIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all lowercase letters.

(2) Format

```
public synchronized boolean storesLowerCaseQuotedIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.103 storesMixedCaseIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in uppercase and lowercase letters.

(2) Format

```
public synchronized boolean storesMixedCaseIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.104 storesMixedCaseQuotedIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in uppercase and lowercase letters.

(2) Format

```
public synchronized boolean storesMixedCaseQuotedIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.105 storesUpperCaseIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all uppercase letters.

(2) Format

```
public synchronized boolean storesUpperCaseIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.106 storesUpperCaseQuotedIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being not case sensitive, and then the results are stored in all uppercase letters.

(2) Format

```
public synchronized boolean storesUpperCaseQuotedIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.107 supportsAlterTableWithAddColumn()

(1) Function

This method acquires a value indicating whether `ALTER TABLE` with added columns is supported.

(2) Format

```
public synchronized boolean supportsAlterTableWithAddColumn() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.108 supportsAlterTableWithDropColumn()

(1) Function

This method acquires a value indicating whether `ALTER TABLE` with dropped columns is supported.

(2) Format

```
public synchronized boolean supportsAlterTableWithDropColumn() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.109 supportsANSI92EntryLevelSQL()

(1) Function

This method acquires a value indicating whether the ANSI92 entry-level SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsANSI92EntryLevelSQL() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.110 supportsANSI92FullSQL()

(1) Function

This method acquires a value indicating whether the ANSI92 full-level SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsANSI92FullSQL() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.111 supportsANSI92IntermediateSQL()

(1) Function

This method acquires a value indicating whether the ANSI92 intermediate-level SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsANSI92IntermediateSQL() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.112 supportsBatchUpdates()

(1) Function

This method acquires a value indicating whether batch updating is supported.

(2) Format

```
public synchronized boolean supportsBatchUpdates() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.113 supportsCatalogsInDataManipulation()

(1) Function

This method acquires a value indicating whether catalog names can be used in data manipulation statements.

(2) Format

```
public synchronized boolean supportsCatalogsInDataManipulation() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.114 supportsCatalogsInIndexDefinitions()

(1) Function

This method acquires a value indicating whether catalog names can be used in index definition statements.

(2) Format

```
public synchronized boolean supportsCatalogsInIndexDefinitions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.115 supportsCatalogsInPrivilegeDefinitions()

(1) Function

This method acquires a value indicating whether catalog names can be used in definition statements for granting privileges (`GRANT` statement) or revoking privileges (`REVOKE` statement).

(2) Format

```
public synchronized boolean supportsCatalogsInPrivilegeDefinitions() throws SQLExcept
ion
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.116 supportsCatalogsInProcedureCalls()

(1) Function

This method acquires a value indicating whether catalog names can be used in procedure call statements.

(2) Format

```
public synchronized boolean supportsCatalogsInProcedureCalls() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.117 supportsCatalogsInTableDefinitions()

(1) Function

This method acquires a value indicating whether catalog names can be used in table definition statements.

(2) Format

```
public synchronized boolean supportsCatalogsInTableDefinitions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.118 supportsColumnAliasing()

(1) Function

This method acquires a value indicating whether aliases are supported for columns.

(2) Format

```
public synchronized boolean supportsColumnAliasing() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.119 supportsConvert()

(1) Function

This method acquires a value indicating whether the `CONVERT` function is supported for SQL types.

(2) Format

```
public synchronized boolean supportsConvert() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.120 supportsConvert(int fromType, int toType)

(1) Function

This method acquires a value indicating whether the `CONVERT` function is supported for specified SQL types.

(2) Format

```
public synchronized boolean supportsConvert(int fromType, int toType) throws SQLExcep
tion
```

(3) Arguments

`int fromType`

SQL type of the conversion source (an SQL type of `java.sql.Types`)

`int toType`

SQL type of the conversion target (an SQL type of `java.sql.Types`)

(4) Return value

The method returns one of the following values:

- `true`: Supported
- `false`: Not supported

The following table lists the return values depending on the combination of conversion source and target types.

Table 8-69: Return values depending on the combination of conversion source and target types

Conversion source type: java.sql.Types	Conversion target type: java.sql.Types																										
	BIT #1	TINYINT #1	SMALLINT #1	INTEGER #2	BIGINT #2	FLOAT #1	REAL #1	DOUBLE #2	NUMERIC #1	DECIMAL #2	CHAR #2	VARCHAR #2	LONGVARCHAR #1	DATE #2	TIME #2	TIMESTAMP #2	BINARY #2	VARBINARY #2	LONGVARBINARY #1	JAVA_OBJECT #1	STRUCT #1	ARRAY #1	BLOB #1	CLOB #1	REF #1
BIT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
TINYINT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
SMALLINT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
INTEGER #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	J	--	J	--	--	--	--	--	--	--	--	--
BIGINT #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	J	--	J	--	--	--	--	--	--	--	--	--
FLOAT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
REAL #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
DOUBLE #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--
NUMERIC #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
DECIMAL #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--
CHAR #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	Y	Y	Y	Y	Y	--	--	--	--	--	--	--
VARCHAR #2	--	--	--	Y	Y	--	--	Y	--	Y	Y	Y	--	Y	Y	Y	Y	Y	--	--	--	--	--	--	--
LONGVARCHAR #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
DATE #2	--	--	--	J	J	--	--	--	--	--	Y	Y	--	Y	--	Y	--	--	--	--	--	--	--	--	--
TIME #2	--	--	--	--	--	--	--	--	--	--	Y	Y	--	--	Y	--	--	--	--	--	--	--	--	--	--
TIMESTAMP #2	--	--	--	J	J	--	--	--	--	--	Y	Y	--	Y	--	Y	--	--	--	--	--	--	--	--	--
BINARY #2	--	--	--	--	--	--	--	--	--	--	Y	Y	--	--	--	--	Y	Y	--	--	--	--	--	--	--
VARBINARY #2	--	--	--	--	--	--	--	--	--	--	Y	Y	--	--	--	--	Y	Y	--	--	--	--	--	--	--
LONGVARBINARY #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
JAVA_OBJECT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
STRUCT #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
ARRAY #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
BLOB #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
CLOB #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
REF #1	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Legend:

Y: Return value is `true` (supported).

J: Return value is `true` (supported as Julian date conversion).

--: Return value is `false` (not supported).

#1

There is no corresponding data type in HADB servers.

#2

For details about the corresponding data type on HADB servers, see (1) Correspondence between HADB's data types and JDBC's SQL data types in 7.6.1 Mapping data types.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.121 supportsCoreSQLGrammar()

(1) Function

This method acquires a value indicating whether the ODBC Core SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsCoreSQLGrammar() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.122 supportsCorrelatedSubqueries()

(1) Function

This method acquires a value indicating whether subqueries that contain external reference columns are supported.

(2) Format

```
public synchronized boolean supportsCorrelatedSubqueries() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.123 supportsDataDefinitionAndDataManipulationTransactions()

(1) Function

This method acquires a value indicating whether data definition statements and data manipulation statements are both supported in transactions.

(2) Format

```
public synchronized boolean supportsDataDefinitionAndDataManipulationTransactions() t
hrows SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.124 supportsDataManipulationTransactionsOnly()

(1) Function

This method acquires a value indicating whether only data manipulation statements are supported in transactions.

(2) Format

```
public synchronized boolean supportsDataManipulationTransactionsOnly() throws SQLExce
ption
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.125 supportsDifferentTableCorrelationNames()

(1) Function

This method acquires a value indicating whether the table names must be different from the correlation names when table correlation names are supported.

(2) Format

```
public synchronized boolean supportsDifferentTableCorrelationNames() throws SQLExcept
ion
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.126 supportsExpressionsInOrderBy()

(1) Function

This method acquires a value indicating whether value expressions are supported in an `ORDER BY` list.

(2) Format

```
public synchronized boolean supportsExpressionsInOrderBy() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.127 supportsExtendedSQLGrammar()

(1) Function

This method acquires a value indicating whether the ODBC Extended SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsExtendedSQLGrammar() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.128 supportsFullOuterJoins()

(1) Function

This method acquires a value indicating whether nested full outer joins are supported.

(2) Format

```
public synchronized boolean supportsFullOuterJoins() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.129 supportsGetGeneratedKeys()

(1) Function

This method acquires a value indicating whether automatic generation keys can be acquired after statements have executed.

(2) Format

```
public synchronized boolean supportsGetGeneratedKeys() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.130 supportsGroupBy()

(1) Function

This method acquires a value indicating whether the `GROUP BY` clause form is supported.

(2) Format

```
public synchronized boolean supportsGroupBy() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.131 supportsGroupByBeyondSelect()

(1) Function

This method acquires a value indicating whether a column that is not in the `SELECT` statement can be used in the `GROUP BY` clause, provided that all columns in the `SELECT` statement are included in the `GROUP BY` clause.

(2) Format

```
public synchronized boolean supportsGroupByBeyondSelect() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.132 supportsGroupByUnrelated()

(1) Function

This method acquires a value indicating whether a column that is not in the `SELECT` statement can be used in the `GROUP BY` clause.

(2) Format

```
public synchronized boolean supportsGroupByUnrelated() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.133 supportsIntegrityEnhancementFacility()

(1) Function

This method acquires a value indicating whether the SQL Integrity Enhancement Facility is supported.

(2) Format

```
public synchronized boolean supportsIntegrityEnhancementFacility() throws SQLExceptio
n
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.134 supportsLikeEscapeClause()

(1) Function

This method acquires a value indicating whether the escape clause is supported in the `LIKE` predicate.

(2) Format

```
public synchronized boolean supportsLikeEscapeClause() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.135 supportsLimitedOuterJoins()

(1) Function

This method acquires a value indicating whether limited support is provided for outer joins.

(2) Format

```
public synchronized boolean supportsLimitedOuterJoins() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.136 supportsMinimumSQLGrammar()

(1) Function

This method acquires a value indicating whether the ODBC Minimum SQL grammar is supported.

(2) Format

```
public synchronized boolean supportsMinimumSQLGrammar() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.137 supportsMixedCaseIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is not enclosed in double quotation marks is processed as being case sensitive, and then the results are stored in uppercase and lowercase letters.

(2) Format

```
public synchronized boolean supportsMixedCaseIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.138 supportsMixedCaseQuotedIdentifiers()

(1) Function

This method acquires a value indicating whether an SQL identifier containing uppercase and lowercase letters that is enclosed in double quotation marks is processed as being case sensitive, and then the results are stored in uppercase and lowercase letters.

(2) Format

```
public synchronized boolean supportsMixedCaseQuotedIdentifiers() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.139 supportsMultipleOpenResults()

(1) Function

This method acquires a value indicating whether it is possible to have multiple `ResultSet` objects that have been returned by a `CallableStatement` object.

(2) Format

```
public synchronized boolean supportsMultipleOpenResults() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.140 supportsMultipleResultSets()

(1) Function

This method acquires a value indicating whether multiple `ResultSet` objects can be acquired from a single call of the `execute` method.

(2) Format

```
public synchronized boolean supportsMultipleResultSets() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.141 supportsMultipleTransactions()

(1) Function

This method acquires a value indicating whether multiple transactions can be open at the same time (for different connections).

(2) Format

```
public synchronized boolean supportsMultipleTransactions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.142 supportsNamedParameters()

(1) Function

This method acquires a value indicating whether named parameters are supported for the `CALL` statement.

(2) Format

```
public synchronized boolean supportsNamedParameters() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.143 supportsNonNullableColumns()

(1) Function

This method acquires a value indicating whether columns can be defined as non-null columns.

(2) Format

```
public synchronized boolean supportsNonNullableColumns() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.144 supportsOpenCursorsAcrossCommit()

(1) Function

This method acquires a value indicating whether the cursor can remain open between commit operations.

(2) Format

```
public synchronized boolean supportsOpenCursorsAcrossCommit() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.145 supportsOpenCursorsAcrossRollback()

(1) Function

This method acquires a value indicating whether the cursor can remain open between rollback operations.

(2) Format

```
public synchronized boolean supportsOpenCursorsAcrossRollback() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.146 supportsOpenStatementsAcrossCommit()

(1) Function

This method acquires a value indicating whether the statement handle can remain open between commit operations.

(2) Format

```
public synchronized boolean supportsOpenStatementsAcrossCommit() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.147 supportsOpenStatementsAcrossRollback()

(1) Function

This method acquires a value indicating whether the statement handle can remain open between rollback operations.

(2) Format

```
public synchronized boolean supportsOpenStatementsAcrossRollback() throws SQLExceptio
n
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.148 supportsOrderByUnrelated()

(1) Function

This method acquires a value indicating whether a column that is not in the `SELECT` statement can be used in the `ORDER BY` clause.

(2) Format

```
public synchronized boolean supportsOrderByUnrelated() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.149 supportsOuterJoins()

(1) Function

This method acquires a value indicating whether some form of outer join is supported.

(2) Format

```
public synchronized boolean supportsOuterJoins() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.150 supportsPositionedDelete()

(1) Function

This method acquires a value indicating whether positioned `DELETE` statements are supported.

(2) Format

```
public synchronized boolean supportsPositionedDelete() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.151 supportsPositionedUpdate()

(1) Function

This method acquires a value indicating whether positioned `UPDATE` statements are supported.

(2) Format

```
public synchronized boolean supportsPositionedUpdate() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.152 supportsRefCursors()

(1) Function

This method acquires a value that indicates whether the database supports `REF CURSOR`.

(2) Format

```
public synchronized boolean supportsRefCursors() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If the `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.153 supportsResultSetConcurrency(int type, int concurrency)

(1) Function

This method acquires a value indicating whether the combination of a specified result set type and a specified concurrent processing type is supported.

(2) Format

```
public synchronized boolean supportsResultSetConcurrency(int type, int concurrency) t
hrows SQLException
```

(3) Arguments

`int type`

Specifies a result set type.

`int concurrency`

Specifies a concurrent processing type.

(4) Return value

`boolean` type:

The method returns one of the following values:

`true`: Supported

`false`: Not supported

If `type` is `ResultSet.TYPE_FORWARD_ONLY` or `ResultSet.TYPE_SCROLL_INSENSITIVE` and `concurrency` is `ResultSet.CONCUR_READ_ONLY`, the method returns `true`; otherwise, the method returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.154 supportsResultSetHoldability(int holdability)

(1) Function

This method acquires a value indicating whether holdability is supported for the specified `ResultSet` object.

(2) Format

```
public synchronized boolean supportsResultSetHoldability(int holdability) throws SQLE
xception
```

(3) Arguments

`int holdability`

Specifies one of the following values:

- `ResultSet.HOLD_CURSORS_OVER_COMMIT`
  The `ResultSet` object is not closed when the `Connection.commit` method is called.
- `ResultSet.CLOSE_CURSORS_AT_COMMIT`
  The `ResultSet` object is closed when the `Connection.commit` method is called.

(4) Return value

`boolean` type:

`true`: Supported

`false`: Not supported

If `ResultSet.HOLD_CURSORS_OVER_COMMIT` is specified for `holdability`, the method returns `true`; otherwise, the method returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.155 supportsResultSetType(int type)

(1) Function

This method acquires a value indicating whether a specified result set type is supported.

(2) Format

```
public synchronized boolean supportsResultSetType(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies a result set type.

(4) Return value

`boolean` type:

The method returns one of the following values:

`true`: Supported

`false`: Not supported

If `ResultSet.TYPE_FORWARD_ONLY` or `ResultSet.TYPE_SCROLL_INSENSITIVE` is specified for `type`, the method returns `true`; otherwise, the method returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.156 supportsSavepoints()

(1) Function

This method acquires a value indicating whether save points are supported.

(2) Format

```
public synchronized boolean supportsSavepoints() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.157 supportsSchemasInDataManipulation()

(1) Function

This method acquires a value indicating whether schema names can be used in data manipulation statements.

(2) Format

```
public synchronized boolean supportsSchemasInDataManipulation() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.158 supportsSchemasInIndexDefinitions()

(1) Function

This method acquires a value indicating whether schema names can be used in index definition statements.

(2) Format

```
public synchronized boolean supportsSchemasInIndexDefinitions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.159 supportsSchemasInPrivilegeDefinitions()

(1) Function

This method acquires a value indicating whether schema names can be used in definition statements for granting privileges (`GRANT` statement) or revoking privileges (`REVOKE` statement).

(2) Format

```
public synchronized boolean supportsSchemasInPrivilegeDefinitions() throws SQLExcepti
on
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.160 supportsSchemasInProcedureCalls()

(1) Function

This method acquires a value indicating whether schema names can be used in procedure call statements.

(2) Format

```
public synchronized boolean supportsSchemasInProcedureCalls() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.161 supportsSchemasInTableDefinitions()

(1) Function

This method acquires a value indicating whether schema names can be used in table definition statements.

(2) Format

```
public synchronized boolean supportsSchemasInTableDefinitions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.162 supportsSelectForUpdate()

(1) Function

This method acquires a value indicating whether `SELECT FOR UPDATE` statements are supported.

(2) Format

```
public synchronized boolean supportsSelectForUpdate() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.163 supportsStatementPooling()

(1) Function

This method acquires a value indicating whether pooling of statement handles is supported.

(2) Format

```
public synchronized boolean supportsStatementPooling() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.164 supportsStoredFunctionsUsingCallSyntax()

(1) Function

This method acquires a value indicating whether user-defined functions or vendor functions that use a stored procedure escape syntax are supported.

(2) Format

```
public synchronized boolean supportsStoredFunctionsUsingCallSyntax() throws SQLExcept
ion
```

(3) Arguments

None.

(4) Return value

The method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.165 supportsStoredProcedures()

(1) Function

This method acquires a value indicating whether stored procedure calls that use a stored procedure escape syntax are supported.

(2) Format

```
public synchronized boolean supportsStoredProcedures() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.166 supportsSubqueriesInComparisons()

(1) Function

This method acquires a value indicating whether subqueries are supported in comparison predicates.

(2) Format

```
public synchronized boolean supportsSubqueriesInComparisons() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.167 supportsSubqueriesInExists()

(1) Function

This method acquires a value indicating whether subqueries are supported in `EXISTS` predicates.

(2) Format

```
public synchronized boolean supportsSubqueriesInExists() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.168 supportsSubqueriesInIns()

(1) Function

This method acquires a value indicating whether subqueries are supported in `IN` predicates.

(2) Format

```
public synchronized boolean supportsSubqueriesInIns() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.169 supportsSubqueriesInQuantifieds()

(1) Function

This method acquires a value indicating whether subqueries are supported in quantified predicates.

(2) Format

```
public synchronized boolean supportsSubqueriesInQuantifieds() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.170 supportsTableCorrelationNames()

(1) Function

This method acquires a value indicating whether table correlation names are supported.

(2) Format

```
public synchronized boolean supportsTableCorrelationNames() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.171 supportsTransactionIsolationLevel(int level)

(1) Function

This method returns a value indicating whether a specified transaction isolation level is supported.

(2) Format

```
public synchronized boolean supportsTransactionIsolationLevel(int level) throws SQLEx
ception
```

(3) Arguments

`int level`

Specifies a transaction isolation level.

(4) Return value

`boolean` type:

`true`: Supported

`false`: Not supported

If `Connection.TRANSACTION_READ_COMMITTED` or `Connection.TRANSACTION_REPEATABLE_READ` is specified in `level`, the method returns `true`; otherwise, the method returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.172 supportsTransactions()

(1) Function

This method acquires a value indicating whether transactions are supported.

(2) Format

```
public synchronized boolean supportsTransactions() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.173 supportsUnion()

(1) Function

This method acquires a value indicating whether `SQL UNION` is supported.

(2) Format

```
public synchronized boolean supportsUnion() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.174 supportsUnionAll()

(1) Function

This method acquires a value indicating whether SQL UNION ALL is supported.

(2) Format

```
public synchronized boolean supportsUnionAll() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `true`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.175 updatesAreDetected(int type)

(1) Function

This method acquires a value indicating whether updating performed on visible rows can be detected by calling the `rowUpdated` method of the `ResultSet` class.

(2) Format

```
public synchronized boolean updatesAreDetected(int type) throws SQLException
```

(3) Arguments

`int type`

Specifies one of the following result set types:

- `ResultSet.TYPE_FORWARD_ONLY`
- `ResultSet.TYPE_SCROLL_INSENSITIVE`
- `ResultSet.TYPE_SCROLL_SENSITIVE`

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.176 usesLocalFilePerTable()

(1) Function

This method acquires a value indicating whether a file is to be used for each table.

(2) Format

```
public synchronized boolean usesLocalFilePerTable() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.6.177 usesLocalFiles()

(1) Function

This method acquires a value indicating whether tables are to be stored in local files.

(2) Format

```
public synchronized boolean usesLocalFiles() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method always returns `false`.

(5) Exceptions

If this `Connection` object is closed before this method is executed, the JDBC driver throws an `SQLException`.

8.7 ResultSetMetaData interface

This section explains the methods provided by the `ResultSetMetaData` interface.

8.7.1 List of the methods in the ResultSetMetaData interface

(1) Main functions of the ResultSetMetaData interface

The `ResultSetMetaData` interface provides the following main function:

- Return of meta information, such as the data type and the data length, for each column in a `ResultSet`

(2) Methods in the ResultSetMetaData interface that are supported by HADB

The following table lists and describes the methods in the `ResultSetMetaData` interface that are supported by HADB.

Table 8-70: Methods in the ResultSetMetaData interface

No.	Method in the ResultSetMetaData interface	Function
1	`getCatalogName(int column)`	Acquires the catalog name for a specified column in a table.
2	`getColumnClassName(int column)`	Acquires the fully specified Java class name for the data type of a column.
3	`getColumnCount()`	Acquires the number of columns in the `ResultSet` object.
4	`getColumnDisplaySize(int column)`	Acquires a specified column's maximum width (in characters).
5	`getColumnLabel(int column)`	Acquires a recommended print or display title for a specified column.
6	`getColumnName(int column)`	Acquires a specified column's column name.
7	`getColumnType(int column)`	Acquires a specified column's SQL data type.
8	`getColumnTypeName(int column)`	Acquires a specified column's data type.
9	`getPrecision(int column)`	Acquires a specified column's length (in digits).
10	`getScale(int column)`	Acquires the number of decimal places in a specified column.
11	`getSchemaName(int column)`	Acquires a specified column's schema name.
12	`getTableName(int column)`	Acquires the name of a specified column's table.
13	`isAutoIncrement(int column)`	Returns a value indicating whether a specified column is both numbered automatically and treated as being read-only.
14	`isCaseSensitive(int column)`	Returns a value indicating whether a specified column is case sensitive.
15	`isCurrency(int column)`	Returns a value indicating whether a specified column is for currency values.
16	`isDefinitelyWritable(int column)`	Returns a value indicating whether write operations on a specified column will be successful.

No.	Method in the ResultSetMetaData interface	Function
17	`isNullable(int column)`	Returns a value indicating whether the null value can be set in a specified column.
18	`isReadOnly(int column)`	Returns a value indicating whether a specified column's values are read-only.
19	`isSearchable(int column)`	Returns a value indicating whether a specified column can be specified in a `where` clause.
20	`isSigned(int column)`	Returns a value indicating whether a specified column is for signed numeric values.
21	`isWritable(int column)`	Returns a value indicating whether write operations on a specified column can be successful.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` might be thrown.

(3) Required package name and class name

The package and class names required in order to use the `ResultSetMetaData` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbResultSetMetaData`

8.7.2 getCatalogName(int column)

(1) Function

This method acquires the catalog name for a specified column in a table.

(2) Format

```
public synchronized String getCatalogName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method always returns a null character string.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.3 getColumnClassName(int column)

(1) Function

This method acquires the fully specified Java class name for the data type of a column.

(2) Format

```
public synchronized String getColumnClassName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method returns the result of executing the `getObject` method of the `ResultSet` object on a column as the `String` Java class type. The following table shows the column data types and corresponding return values.

Table 8-71: Character strings returned when the getColumnClassName method is executed

Column's data type (HADB data type)	Character string returned
INTEGER	"java.lang.Long"
SMALLINT	"java.lang.Integer"
	"java.lang.Short"#
DOUBLE PRECISION	"java.lang.Double"
DECIMAL	"java.math.BigDecimal"
CHAR	"java.lang.String"
VARCHAR	"java.lang.String"
DATE	"java.sql.Date"
TIME	"java.sql.Time"
TIMESTAMP	"java.sql.Timestamp"
BINARY	"java.lang.Object"
VARBINARY	"java.lang.Object"
ROW	"java.lang.Object"

Column's data type (HADB data type)	Character string returned
BOOLEAN (column found only in a ResultSet created from DatabaseMetaData)	"java.lang.Boolean"

#

This column is found only in a ResultSet created from DatabaseMetaData. If the column's data type is defined as short, this value is returned.

(5) Exceptions

If the specified column value is 0 or less, or is greater than the number of columns in the table, the JDBC driver throws an SQLException.

8.7.4 getColumnCount()

(1) Function

This method acquires the number of columns in the ResultSet object.

(2) Format

```
public synchronized int getColumnCount() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the number of columns in the ResultSet object.

(5) Exceptions

None.

8.7.5 getColumnDisplaySize(int column)

(1) Function

This method acquires a specified column's maximum width (in characters).

(2) Format

```
public synchronized int getColumnDisplaySize(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns the maximum width in characters. The following table shows the return values of the `getColumnDisplaySize` method.

Table 8-72: Return values of the getColumnDisplaySize method

Column's data type (HADB data type)	Return value (maximum number of characters)
`INTEGER`	`20`
`SMALLINT`	`11`
	`6`#1
`DOUBLE PRECISION`	`23`
`DECIMAL(`*m*`,`*n*`)`	*m* + 2
`CHAR(`*n*`)` `VARCHAR(`*n*`)`	*n*
`DATE`	`10`
`TIME(`*p*`)`	*p* = 0: `8` *p* > 0: 8 + (*n* + 1)
`TIMESTAMP(`*p*`)`	*p* = 0: `19` *p* > 0: 19 + (*n* + 1)
`BINARY(`*n*`)` `VARBINARY(`*n*`)`	*n* × 2
`ROW`	*row-length*#2
`BOOLEAN` (column found only in a `ResultSet` created from `DatabaseMetaData`)	`5`

#1

This column is found only in a `ResultSet` created from `DatabaseMetaData`. If the column's data type is defined as `short`, this value is returned.

#2

Sum of the data lengths of all columns. For details about how to obtain a column's data length, see the topic *List of data types* in the manual *HADB SQL Reference*.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.6 getColumnLabel(int column)

(1) Function

This method acquires a recommended print or display title for a specified column.

(2) Format

```
public synchronized String getColumnLabel(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method returns the column name, which is the same value returned by `getColumnName` of the `ResultSetMetaData` object.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.7 getColumnName(int column)

(1) Function

This method acquires a specified column's column name.

(2) Format

```
public synchronized String getColumnName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method returns the name of a retrieval result column from the column name information sent from the HADB server. For details about the names of retrieval result columns, see *Rules* in *Specification format and rules for the SELECT statement* in the manual *HADB SQL Reference*.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.8 getColumnType(int column)

(1) Function

This method acquires a specified column's SQL data type.

(2) Format

```
public synchronized int getColumnType(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns the SQL type from `java.sql.Types`.

For details about the correspondence between data types and return values of columns, see (1) Correspondence between HADB's data types and JDBC's SQL data types in 7.6.1 Mapping data types.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.9 getColumnTypeName(int column)

(1) Function

This method acquires a specified column's data type.

(2) Format

```
public synchronized String getColumnTypeName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object. The following table shows the return values of the `getColumnTypeName` method.

Table 8-73: Return values of the getColumnTypeName method

Column's data type (HADB data type)	Return value (character string returned)
`INTEGER`	`"INTEGER"`
`SMALLINT`	`"SMALLINT"`#
`DECIMAL`	`"DECIMAL"`
`CHAR`	`"CHAR"`
`DOUBLE PRECISION`	`"DOUBLE PRECISION"`
`VARCHAR`	`"VARCHAR"`
`DATE`	`"DATE"`
`TIME`	`"TIME"`
`TIMESTAMP`	`"TIMESTAMP"`
`BINARY`	`"BINARY"`
`VARBINARY`	`"VARBINARY"`
`ROW`	`"ROW"`
`BOOLEAN` (column found only in a `ResultSet` created from `DatabaseMetaData`)	`"BOOLEAN"`

#

This column is found only in a `ResultSet` created from `DatabaseMetaData`. If the column's data type is defined as `short`, this value is returned.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.10 getPrecision(int column)

(1) Function

This method acquires a specified column's length (in digits).

(2) Format

```
public synchronized int getPrecision(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns the specified column's length in decimal digits. If the specified column has a numeric data type, the method returns the number of digits. If it does not have a numeric data type, the method returns the column length in bytes. The following table shows the return values of the `getPrecision` method.

Table 8-74: Return values of the getPrecision method

Column's data type (HADB data type)	Return value (column length in digits)
`INTEGER`	`19`
`SMALLINT`	`10`
	`5`#1
`DOUBLE PRECISION`	`17`
`DECIMAL(`*m*`,`*n*`)`	*m*
`CHAR(`*n*`)` `VARCHAR(`*n*`)`	*n*
`DATE`	`10`
`TIME(`*p*`)`	*p* = 0: `8` *p* > 0: 8 + (*n* + 1)
`TIMESTAMP(`*p*`)`	*p* = 0: `19` *p* > 0: 19 + (*n* + 1)
`BINARY(`*n*`)` `VARBINARY(`*n*`)`	*n*
`ROW`	*row-length*#2
`BOOLEAN` (column found only in a `ResultSet` created from `DatabaseMetaData`)	`1`

#1

This column is found only in a `ResultSet` created from `DatabaseMetaData`. If the column's data type is defined as `short`, this value is returned.

#2

Sum of the data lengths of all columns. For details about how to obtain the data length of a column, see the topic *List of data types* in the manual *HADB SQL Reference*.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.11 getScale(int column)

(1) Function

This method acquires the number of decimal places in a specified column.

(2) Format

```
public synchronized int getScale(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a decimal number indicating the number of decimal places. The following table shows the return values of the `getScale` method.

Table 8-75: Return values of the getScale method

Column's data type (HADB data type)	Return value (number of decimal places)
`DECIMAL(`*m*`,`*n*`)`	*n*
`TIME(`*p*`)` `TIMESTAMP(`*p*`)`	*p*
Other than the above	`0`

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.12 getSchemaName(int column)

(1) Function

This method acquires a specified column's schema name.

(2) Format

```
public synchronized String getSchemaName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method always returns a null character string.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.13 getTableName(int column)

(1) Function

This method acquires the name of a specified column's table.

(2) Format

```
public synchronized String getTableName(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns a `String` object.

This method always returns a null character string.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.14 isAutoIncrement(int column)

(1) Function

This method returns a value indicating whether a specified column is both numbered automatically and treated as being read-only.

(2) Format

```
public synchronized boolean isAutoIncrement(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.15 isCaseSensitive(int column)

(1) Function

This method returns a value indicating whether a specified column is case sensitive.

(2) Format

```
public synchronized boolean isCaseSensitive(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.16 isCurrency(int column)

(1) Function

This method returns a value indicating whether a specified column is for currency values.

(2) Format

```
public synchronized boolean isCurrency(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.17 isDefinitelyWritable(int column)

(1) Function

This method returns a value indicating whether write operations on a specified column will be successful.

(2) Format

```
public synchronized boolean isDefinitelyWritable(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.18 isNullable(int column)

(1) Function

This method returns a value indicating whether the null value can be set in a specified column.

(2) Format

```
public synchronized int isNullable(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns one of the following values:

- `ResultSetMetaData.columnNoNulls`: The null value cannot be set.
- `ResultSetMetaData.columnNullable`: The null value can be set.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.19 isReadOnly(int column)

(1) Function

This method returns a value indicating whether a specified column's values are read-only.

(2) Format

```
public synchronized boolean isReadOnly(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.20 isSearchable(int column)

(1) Function

This method returns a value indicating whether a specified column can be specified in a `where` clause.

(2) Format

```
public synchronized boolean isSearchable(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns one of the following values:

`true`

The column can be specified in a `where` clause.

`false`

The column cannot be specified in a `where` clause.

If the `ResultSet` object was created by the `DatabaseMetaData` interface, the method returns `false`; otherwise, the method returns `true`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.21 isSigned(int column)

(1) Function

This method returns a value indicating whether a specified column is for signed numeric values.

(2) Format

```
public synchronized boolean isSigned(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

The method returns one of the following values:

`true`

The column is for signed numeric values.

`false`

The column is not for signed numeric values.

The following table shows the return value depending on the parameter's data type.

Table 8-76: Return value depending on the parameter's data type

Parameter's data type	Return value
`INTEGER`, `SMALLINT`, `DOUBLE PRECISION`, `DECIMAL`	`true`
Other	`false`

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.7.22 isWritable(int column)

(1) Function

This method returns a value indicating whether write operations on a specified column can be successful.

(2) Format

```
public synchronized boolean isWritable(int column) throws SQLException
```

(3) Arguments

`int column`

Specifies a column number (where the first column is `1`).

(4) Return value

This method always returns `false`.

(5) Exceptions

If the specified `column` value is `0` or less, or is greater than the number of columns in the table, the JDBC driver throws an `SQLException`.

8.8 SQLException interface

The `SQLException` interface uses the `SQLException` class of the `java.sql` package directly. For details and usage information about the methods provided by the `SQLException` interface, refer to the JDBC documentation provided by JavaSoft.

8.9 SQLWarning interface

The `SQLWarning` interface provides information related to database access warnings.

`SQLWarning` objects are accumulated in a method's object that triggers warning reports without an exception notification.

8.9.1 Creating an SQLWarning object

If settings are specified in such a manner that warnings that occur during execution of SQL statements are to be retained in the JDBC driver, the JDBC driver generates `SQLWarning` objects and retains warning information.

You can specify warning retention with `sqlwarningkeep` in the URL or user property or with the `setSQLWarningKeep` method.

8.9.2 Releasing SQLWarning objects

`SQLWarning` objects are accumulated as a chain linked to the method's object (`Connection`, `Statement`, `PreparedStatement`, and `ResultSet`) that triggers the warning reports.

To release accumulated `SQLWarning` objects explicitly, execute the `clearWarnings` method for the method's object that triggered the warnings.

8.10 Unsupported interfaces

HADB does not support the following interfaces:

- `Array`
- `Blob`
- `CallableStatement`
- `Clob`
- `Savepoint`
- `SQLData`
- `SQLInput`
- `SQLOutput`

9 The JDBC 2.1 Core API

This chapter explains HADB's scope of support for the functions added in the JDBC 2.1 Core API.

9.1 Scope of support for the result set extended functions

The following table shows HADB's scope of support for the result set extended functions (`ResultSet` class) of the JDBC 2.1 Core API.

Table 9-1: HADB's scope of support for the result set extended functions (`ResultSet` class) of the JDBC 2.1 Core API

Result set extended function (ResultSet class)		HADB's scope of support
Scrolling type	Forward-only scrolling	Y
	Scroll-insensitive scrolling	Y
	Scroll-sensitive scrolling	N
Concurrent processing type	Read-only processing	Y
	Updatable	N

Legend:

Y: Supported by HADB

N: Not supported by HADB

Important

- No error results if an unsupported scrolling type or concurrent processing type is specified. The JDBC 2.1 Core API assumes the result set that is closest to the specified scrolling type or concurrent processing type, and creates an instance of the `Statement` class or that subclass. In this case, the API generates a warning (`SQLWarning` object) and associates it with an instance of the `Connection` class.
- In the case of a scrolling-type result set, all retrieved data is cached in the JDBC driver. This means that a large amount of data increases the possibility of a memory shortage or deterioration in performance. When you use a scrolling-type result set, take steps in advance to minimize the volume of retrieved data by, for example, adding appropriate conditions to the SQL statements.

9.2 Scope of support for batch update functionality

This section explains HADB's scope of support for the batch update functionality with the `Statement` and `PreparedStatement` classes.

9.2.1 SQL statements that can use the batch update functionality

The following SQL statements can use the batch update functionality:

- Definition SQL statements
- `DELETE`
- `INSERT`
- `PURGE CHUNK`
- `TRUNCATE TABLE`
- `UPDATE`

Note that if you execute the following SQL statements, `BatchUpdateException` is thrown when a `executeBatch` or `executeLargeBatch` method is executed.

- `SELECT`
- `PURGE CHUNK` statement that specifies a dynamic parameter

9.2.2 Batch update functionality with the Statement class

The following notes apply to the batch update functionality with the `Statement` class.

- Use the `addBatch` method to register multiple SQL statements.
- Use the `executeBatch` method or the `executeLargeBatch` method to execute the registered SQL statements as a batch.
- An array of the numbers of rows updated by the individual SQL statements is returned as the batch execution results.
- If an error occurs during batch execution, the batch update functionality throws a `BatchUpdateException`.

9.2.3 Batch update functionality with the PreparedStatement class

The following notes apply to the batch update functionality with the `PreparedStatement` class.

- Use the normal procedure (`set`*XXX* method) to specify dynamic parameters for SQL statements specified when a `PreparedStatement` instance was created.
- Use the `addBatch` method to register the dynamic parameter sets.
- Use the `executeBatch` method or the `executeLargeBatch` method to execute the multiple registered dynamic parameter sets as a batch.

- An array of the numbers of rows updated by the individual dynamic parameter sets is returned as the batch execution results.
- If an error occurs during batch execution, the batch update functionality throws a `BatchUpdateException`.

9.2.4 Notes

(1) Executing implicit commit processing

If the SQL statements registered with `addBatch` contain any of the SQL statements shown below, you must exercise caution in using the batch update functionality for the SQL statements because the HADB server performs commit processing implicitly when such an SQL statement is executed:

- Definition SQL statements
- `PURGE CHUNK` statement
- `TRUNCATE TABLE` statement

(2) Processing by the batch update functionality when addBatch specifications for parameters and SQL statements are combined

When `addBatch` lines specifying parameters are mixed in with `addBatch` lines for SQL statements, the batch update functionality executes the `addBatch` statements sequentially, instead of in a single 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 application program is executed, an SQL statement is executed at each `addBatch` line, because an `addBatch` line specifying parameters is mixed in with `addBatch` lines for the SQL statements. Therefore, executing this application program produces the same results as executing the following application program:

```
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();
```

If you use the batch update functionality in code that contains a mix of `addBatch` lines specifying parameters and `addBatch` lines for SQL statements, we recommend that you disable the automatic commit mode for the `Connection` class.

> **Note**
>
> For code that contains a mix of `addBatch` parameters and SQL statements, individual lines are executed sequentially. However, if the automatic commit mode is enabled, commit processing is performed implicitly for each execution unit. If an error occurs during batch update processing, only the portion of the processing up to the error is committed, making it impossible to identify the point at which commit processing occurred. Therefore, we recommend that you disable the automatic commit mode.

(3) Registering many parameters with the addBatch method

The JDBC driver retains all the parameters registered with the `addBatch` method until the `executeBatch` method or the `executeLargeBatch` method is executed. Be aware of the amount of memory that will be used when you register many parameters.

(4) Update count reported by BatchUpdateException

The following shows the update count that is reported in the return value of the `getUpdateCounts` method of the `BatchUpdateException` that occurs during batch update processing:

- The array that is returned contains as many elements as the number of SQL statements that executed.
- The number of updated rows is set in each array element.

If the processing is rolled back internally because of an exception, an array containing no elements is returned.

The following shows an example of an update count.

■ Example program of consecutive execution using a JDBC driver

```
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO T1 VALUES(1,'aaaa')");
stmt.addBatch("INSERT INTO T1 VALUES(2,'bbbbbbbb')");...[A]
stmt.addBatch("INSERT INTO T1 VALUES(3,'cccc')");
stmt.executeBatch();
```

The following shows the update count that is returned by the `getUpdateCounts` method when this example program is executed and the parameter or SQL statement registered in `[A]` results in an error:

- Array containing one element
- Value of element 0: Number of updated rows

If the number of updated rows might exceed `Integer.MAX_VALUE`, use the `executeLargeBatch` method instead of the `executeBatch` method. Similarly, use the `getLargeUpdateCounts` method instead of the `getUpdateCounts` method.

9.3 Added data types

Several new JDBC SQL types have been added to the JDBC 2.1 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

9.4 Unsupported interfaces

HADB does not support the following interfaces:

- `Array`
- `Blob`
- `Clob`
- `Ref`
- `SQLData`
- `SQLInput`
- `SQLOutput`
- `Struct`

10 The JDBC 2.0 Optional Package

This chapter explains the interfaces and methods in the JDBC 2.0 Optional Package.

10.1 HADB's scope of support for the functions added in the JDBC 2.0 Optional Package

The following table shows HADB's scope of support for the functions added in the JDBC 2.0 Optional Package.

Table 10-1: HADB's scope of support for the functions added in the JDBC 2.0 Optional Package

Function added in the JDBC 2.0 Optional Package	Corresponding interface	HADB's scope of support
JNDI support	`DataSource`	Y
Connection pool	`ConnectionPoolDataSource`	Y
	`PooledConnection`	Y
`RowSets`	`RowSet`	N
	`RowSetInternal`	N
	`RowSetListner`	N
	`RowSetMetaData`	N
	`RowSetReader`	N

Legend:

Y: Supported by HADB

N: Not supported by HADB

In addition to the interfaces listed above, there also are methods related to specification and acquisition of HADB-specific connection information. For details, see 10.5 Connection information setup and acquisition interface.

10.2 DataSource interface

This section explains the methods provided by the `DataSource` interface.

10.2.1 List of the methods in the DataSource interface

The following table lists and describes the methods in the `DataSource` interface that are supported by HADB.

Table 10-2: Methods in the DataSource interface

No.	Method of the DataSource interface	Function
1	`getConnection()`	Connects to the HADB server.
2	`getConnection(String username, String password)`	
3	`getLoginTimeout()`	Acquires the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server).
4	`getLogWriter()`	Acquires the `DataSource` object's log writer.
5	`setLoginTimeout(int seconds)`	Specifies a timeout value (in seconds) for HADB server connection processing.
6	`setLogWriter(PrintWriter out)`	Sets a log writer for the `DataSource` object.

The package and class names required in order to use the `DataSource` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbDataSource`

10.2.2 getConnection()

(1) Function

This method connects to the HADB server.

The connection is performed according to the HADB server connection information specified in the `DataSource` object, and then a `Connection` object is returned.

Specification priorities apply to the HADB server connection information. For details about the priorities, see 7.3.3 Connection information priorities.

Note that you must have the `CONNECT` privilege to execute the `getConnection` method.

(2) Format

```
public synchronized Connection getConnection() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns a `Connection` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurs.
- Specified HADB server connection information is invalid.

10.2.3 getConnection(String username, String password)

(1) Function

This method connects to the HADB server.

The connection is performed according to the HADB server connection information specified in the `DataSource` object and the connection information specified in the arguments, and then a `Connection` object is returned.

Specification priorities apply to the HADB server connection information. For details about the priorities, see 7.3.3 Connection information priorities.

Note that you must have the `CONNECT` privilege to execute the `getConnection` method.

(2) Format

```
public synchronized Connection getConnection(String username, String password) throws
 SQLException
```

(3) Arguments

`String username`

Specifies the authorization identifier that is to be used to connect to the HADB server.

`String password`

Specifies a password for the authorization identifier that is to be used to connect to the HADB server.

If `null` is specified for `username` or `password`, the JDBC driver assumes that no authorization identifier or password, respectively, is specified. If a character string with a length of zero is specified for `password`, the JDBC driver also assumes that no password is specified.

(4) Return value

This method returns a `Connection` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following cases:

- A database access error occurs.
- Specified HADB server connection information is invalid.
- The authorization identifier specified for `username` is a character string with a length of zero.

10.2.4 getLoginTimeout()

(1) Function

This method acquires the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server).

(2) Format

```
public synchronized int getLoginTimeout()
```

(3) Arguments

None.

(4) Return value

This method returns the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server). If no timeout value has been specified by the `setLoginTimeout` method, the method returns `0`.

(5) Exceptions

None.

10.2.5 getLogWriter()

(1) Function

This method acquires the `DataSource` object's log writer.

(2) Format

```
public synchronized PrintWriter getLogWriter() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the `DataSource` object's log writer. If no log writer has been set, the method returns `null`.

(5) Exceptions

None.

10.2.6 setLoginTimeout(int seconds)

(1) Function

This method specifies a timeout value (in seconds) for HADB server connection processing.

This timeout value is applied when the `getConnection` method is used to connect to the HADB server.

(2) Format

```
public synchronized void setLoginTimeout(int seconds) throws SQLException
```

(3) Arguments

`int seconds`

Specifies a timeout value in the range from 0 to 300 (seconds) for HADB server connection processing.

If `0` is specified, the value specified for `adb_clt_rpc_con_wait_time` in the system properties, user properties, or connection URL properties is assumed. If `adb_clt_rpc_con_wait_time` is not specified, the default value of `adb_clt_rpc_con_wait_time` is assumed.

(4) Return value

None.

(5) Exceptions

If an invalid value (a value less than 1 or greater than 300) is specified for `seconds`, the JDBC driver throws an `SQLException`.

(6) Notes

You can set timeout times for connections with the HADB server in a number of locations, such as in various properties and methods. For information about the relative priorities that determine which timeout time applies to connections with the HADB server, see (1) Connection information needed when a connection to the HADB server is established in 7.3.3 Connection information priorities.

10.2.7 setLogWriter(PrintWriter out)

(1) Function

This method sets a log writer for the `DataSource` object.

(2) Format

```
public synchronized void setLogWriter(PrintWriter out) throws SQLException
```

(3) Arguments

`PrintWriter out`

Specifies a log writer.

(4) Return value

None.

(5) Exceptions

None.

10.3 ConnectionPoolDataSource interface

This section explains the methods provided by the `ConnectionPoolDataSource` interface.

10.3.1 List of the methods in the ConnectionPoolDataSource interface

The following table lists and describes the methods in the `ConnectionPoolDataSource` interface that are supported by HADB.

Table 10-3: Methods in the ConnectionPoolDataSource interface

No.	Method of the ConnectionPoolDataSource interface	Function
1	`getLoginTimeout()`	Acquires the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server).
2	`getLogWriter()`	Acquires the `ConnectionPoolDataSource` object's log writer.
3	`getPooledConnection()`	Creates a `PooledConnection` object from the connection information specified in the `DataSource` object.
4	`getPooledConnection(String user, String password)`	Creates a `PooledConnection` object from the connection information specified in the arguments and the connection information contained in the `DataSource` object.
5	`setLoginTimeout(int seconds)`	Specifies a timeout value (in seconds) for HADB server connection processing.
6	`setLogWriter(PrintWriter out)`	Sets a log writer for the `ConnectionPoolDataSource` object.

The package and class names required in order to use the `ConnectionPoolDataSource` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbConnectionPoolDataSource`

10.3.2 getLoginTimeout()

(1) Function

This method acquires the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server).

(2) Format

```
public synchronized int getLoginTimeout() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the value set by the `setLoginTimeout` method (the timeout time for connections to the HADB server). If no timeout value has been specified by the `setLoginTimeout` method, the method returns `0`.

(5) Exceptions

None.

10.3.3 getLogWriter()

(1) Function

This method acquires the `ConnectionPoolDataSource` object's log writer.

(2) Format

```
public synchronized PrintWriter getLogWriter() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the `ConnectionPoolDataSource` object's log writer. If no log writer has been set, the method returns `null`.

(5) Exceptions

None.

10.3.4 getPooledConnection()

(1) Function

This method creates a `PooledConnection` object from the connection information specified in the `DataSource` object.

Specification priorities apply to the authorization identifier and password. For details about the priorities, see 7.3.3 Connection information priorities.

Note that you must have the `CONNECT` privilege to execute the `getPooledConnection` method.

(2) Format

```
public synchronized PooledConnection getPooledConnection() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns a `PooledConnection` object.

(5) Exceptions

None.

10.3.5 getPooledConnection(String user, String password)

(1) Function

This method creates a `PooledConnection` object from the connection information specified in the arguments and the connection information contained in the `DataSource` object.

Specification priorities apply to the authorization identifier and password. For details about the priorities, see 7.3.3 Connection information priorities.

Note that you must have the `CONNECT` privilege to execute the `getPooledConnection` method.

(2) Format

```
public synchronized PooledConnection getPooledConnection(String user, String password
) throws SQLException
```

(3) Arguments

`String user`

Specifies the authorization identifier that is to be used to connect to the HADB server.

`String password`

Specifies a password for the authorization identifier that is to be used to connect to the HADB server.

If `null` is specified for `user` or `password`, the JDBC driver assumes that no authorization identifier or password, respectively, is specified. If a character string with a length of zero is specified for `password`, the JDBC driver also assumes that no password is specified.

The authorization identifier specified for `user` takes precedence over the authorization identifier specified by the `setUser` method. Similarly, the password specified for `password` takes precedence over the password specified by the `setPassword` method.

(4) Return value

This method returns a `PooledConnection` object.

(5) Exceptions

The JDBC driver throws an `SQLException` in the following case:

- The authorization identifier specified for `user` is a character string with a length of zero.

10.3.6 setLoginTimeout(int seconds)

(1) Function

This method specifies a timeout value (in seconds) for HADB server connection processing.

This timeout value is applied when the `getConnection` method is used to connect to the HADB server.

(2) Format

```
public synchronized void setLoginTimeout(int seconds) throws SQLException
```

(3) Arguments

`int seconds`

Specifies a timeout value in the range from 0 to 300 (seconds) for HADB server connection processing.

If `0` is specified, the value of `adb_clt_rpc_con_wait_time` in the system properties, user properties, or connection URL properties is assumed. If `adb_clt_rpc_con_wait_time` is not specified, the default value of `adb_clt_rpc_con_wait_time` is assumed.

(4) Return value

None.

(5) Exceptions

If an invalid value (a value less than 1 or greater than 300) is specified for `seconds`, the JDBC driver throws an `SQLException`.

(6) Notes

You can set timeout times for connections with the HADB server in a number of locations, such as in various properties and methods. For information about the relative priorities that determine which timeout time applies to connections with the HADB server, see (1) Connection information needed when a connection to the HADB server is established in 7.3.3 Connection information priorities.

10.3.7 setLogWriter(PrintWriter out)

(1) Function

This method sets a log writer for the `ConnectionPoolDataSource` object.

(2) Format

```
public synchronized void setLogWriter(PrintWriter out)
```

(3) Arguments

`PrintWriter out`

Specifies a log writer.

(4) Return value

None.

(5) Exceptions

None.

10.4 PooledConnection interface

This section explains the methods provided by the `PooledConnection` interface.

10.4.1 List of the methods in the PooledConnection interface

The following table lists and describes the methods in the `PooledConnection` interface that are supported by HADB.

Table 10-4: Methods in the PooledConnection interface

No.	Method of the PooledConnection interface	Function
1	`addConnectionEventListener(ConnectionEventListener listener)`	Registers an event listener so that events that occur in this `PooledConnection` object will be reported.
2	`close()`	Closes the physical connection with the HADB server. This method closes all physical connections pooled in the connection pool.
3	`getConnection()`	Connects to the HADB server by using a connection pooled in the connection pool.
4	`removeConnectionEventListener(ConnectionEventListener listener)`	Deletes a specified event listener from the component list that is reported when events occur in this `PooledConnection` object.

The package and class names required in order to use the `PooledConnection` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbPooledConnection`

10.4.2 addConnectionEventListener(ConnectionEventListener listener)

(1) Function

This method registers an event listener so that events that occur in this `PooledConnection` object will be reported.

No other methods can be called from the event listener that is registered by this method. Deadlock might result if an attempt is made to call another method, resulting in a loss of response.

(2) Format

```
public synchronized void addConnectionEventListener(ConnectionEventListener listener)
```

(3) Arguments

`ConnectionEventListener listener`

Specifies a component that implements the `ConnectionEventListener` interface, so that if the connection is closed or an error occurs, that event will be reported. Normally, this is a connection pool management program.

If `null` is specified, nothing is registered.

(4) Return value

None.

(5) Exceptions

None.

10.4.3 close()

(1) Function

This method closes the physical connection with the HADB server. The method closes all physical connections pooled in the connection pool. This method closes the connection even if the database is being accessed.

(2) Format

```
public synchronized void close()
```

(3) Arguments

None.

(4) Return value

None.

(5) Exceptions

None.

10.4.4 getConnection()

(1) Function

This method connects to the HADB server by using a connection pooled in the connection pool. If all connections pooled in the connection pool are in use, this method establishes a new physical connection with the HADB server to connect to the HADB server.

Note that you must have the `CONNECT` privilege to execute the `getConnection` method.

Note

- A physical connection with the HADB server is not closed until this class object is closed. The physical connection with the HADB server is maintained even if the `close` method is executed on the `Connection` object (this class object maintains the physical connection). This connection is used again the next time the `getConnection` method is executed.

- When a connection pooled in the connection pool is used to connect to the HADB server, the timeout value for HADB server connection processing that was specified by the `setLoginTimeout` method is not applied. Normally, the timeout value specified by the `setLoginTimeout` method is used to monitor the time required for communication processing when a physical connection is established with the HADB server. This timeout value is not applied when a connection pooled in the connection pool is used to connect to the HADB server because no physical connection occurs (no time is required for communication processing).

(2) Format

```
public synchronized Connection getConnection() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns a `Connection` object.

(5) Exceptions

The JDBC driver throws an `SQLException` if a database access error occurs.

10.4.5 removeConnectionEventListener(ConnectionEventListener listener)

(1) Function

This method deletes a specified event listener from the component list that is reported when events occur in this `PooledConnection` object.

(2) Format

```
public synchronized void removeConnectionEventListener(ConnectionEventListener listen
er)
```

(3) Arguments

`ConnectionEventListener listener`

Specifies a component that implements the `ConnectionEventListener` interface and has been registered as an event listener. Normally, this is a connection pool management program.

(4) Return value

None.

(5) Exceptions

None.

10.5 Connection information setup and acquisition interface

This section explains the methods provided by the connection information setup and acquisition interface.

10.5.1 List of the methods in the connection information setup and acquisition interface

The `DataSource` and `ConnectionPoolDataSource` classes provide the HADB-specific methods described in the table below, in addition to the methods that are standardized for the JDBC 2.0 Optional Package.

Table 10-5: Methods in the connection information setup and acquisition interface

No.	Method	Function
1	`getApName()`	Acquires the application identifier that was specified by the `setApName` method.
2	`getEncodeLang()`	Acquires the name of the conversion character set that was specified in the `setEncodeLang` method.
3	`getInterfaceMethodTrace()`	Acquires the status of acquisition of JDBC interface method traces that was specified by the `setInterfaceMethodTrace` method.
4	`getNotErrorOccurred()`	Acquires the setting as to whether generation of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.
5	`getPassword()`	Acquires the password that was specified by the `setPassword` method.
6	`getSQLWarningKeep()`	Acquires the setting as to whether warning information generated during execution of SQL statements is to be retained.
7	`getTraceNumber()`	Acquires the number of entries for a JDBC interface method trace that was specified by the `setTraceNumber` method.
8	`getUser()`	Acquires the authorization identifier that was specified by the `setUser` method.
9	`getHostName()`	Acquires the host name of the HADB server that was specified by the `setHostName` method.
10	`getPort()`	Acquires the port number of the HADB server that was specified by the `setPort` method.
11	`setApName(String name)`	Specifies the application identifier for connecting to the HADB server.
12	`setEncodeLang(String lang)`	Specifies the name of the conversion character set to be used when conversion of character encoding is performed.
13	`setInterfaceMethodTrace(boolean flag)`	Specifies whether JDBC interface method traces are to be acquired.
14	`setNotErrorOccurred(boolean mode)`	Specifies the setting as to whether generation of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.
15	`setPassword(String password)`	Specifies a password for the authorization identifier that is used to connect to the HADB server.
16	`setSQLWarningKeep(boolean mode)`	Specifies the setting as to whether warning information generated during execution of SQL statements is to be retained.
17	`setTraceNumber(int num)`	Specifies the number of entries for a JDBC interface method trace.

No.	Method	Function
18	`setUser(String user)`	Specifies the authorization identifier to be used to connect to the HADB server.
19	`setHostName(String name)`	Specifies the host name of the HADB server at the connection destination.
20	`setPort(int port)`	Specifies the port number of the HADB server at the connection destination.

10.5.2 getApName()

(1) Function

This method acquires the application identifier that was specified by the `setApName` method.

(2) Format

```
public synchronized String getApName() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the application identifier. If the `setApName` method has not been executed, the method returns `"********"`.

(5) Exceptions

None.

10.5.3 getEncodeLang()

(1) Function

This method acquires the name of the conversion character set that was specified in the `setEncodeLang` method.

(2) Format

```
public synchronized String getEncodeLang() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the name of the conversion character set. If the `setEncodeLang` method has not been executed, the method returns `null`.

(5) Exceptions

None.

10.5.4 getInterfaceMethodTrace()

(1) Function

This method acquires the status of acquisition of JDBC interface method traces that was specified by the `setInterfaceMethodTrace` method. For details about the JDBC interface method traces, see 7.7.1 JDBC interface method traces.

(2) Format

```
public boolean getInterfaceMethodTrace() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the status of acquisition of JDBC interface method traces:

- `true`: JDBC interface method traces are acquired.
- `false`: JDBC interface method traces are acquired.

(5) Exceptions

None.

10.5.5 getNotErrorOccurred()

(1) Function

This method acquires the setting as to whether generation of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed. The method acquires the information that was specified by the `setNotErrorOccurred` method.

(2) Format

```
public boolean getNotErrorOccurred() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the setting as to whether `ConnectionEventListener.connectionErrorOccurred` is to be generated:

- `true`: `connectionErrorOccurred` is not generated (generation is suppressed).
- `false`: `connectionErrorOccurred` is generated (generation is not suppressed).

If the `setNotErrorOccurred` method has not been executed, the method returns `false` (default value).

(5) Exceptions

None.

10.5.6 getPassword()

(1) Function

This method acquires the password that was specified by the `setPassword` method.

(2) Format

```
public synchronized String getPassword() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the password that was specified by the `setPassword` method.

(5) Exceptions

None.

10.5.7 getSQLWarningKeep()

(1) Function

This method acquires the setting as to whether warning information generated during execution of SQL statements is to be retained. The method acquires the information that was specified by the `setSQLWarningKeep` method.

(2) Format

```
public synchronized boolean getSQLWarningKeep() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the setting as to whether warning information generated is to be retained by the `Connection` class:

- `true`: Warning information is to be retained.
- `false`: Warning information is to be not retained.

If the `setSQLWarningKeep` method has not been executed, the method returns `true` (default value).

(5) Exceptions

None.

10.5.8 getTraceNumber()

(1) Function

This method acquires the number of entries for a JDBC interface method trace that was specified by the `setTraceNumber` method.

(2) Format

```
public synchronized int getTraceNumber() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the number of entries for a JDBC interface method trace that was specified by the `setTraceNumber` method. If the `setTraceNumber` method has not been executed, the method returns `500` (default value).

(5) Exceptions

None.

10.5.9 getUser()

(1) Function

This method acquires the authorization identifier that was specified by the `setUser` method.

(2) Format

```
public synchronized String getUser() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the authorization identifier that was specified by the `setUser` method. If the `setUser` method has not been executed, the method returns `null`.

(5) Exceptions

None.

10.5.10 getHostName()

(1) Function

This method acquires the host name of the HADB server that was specified by the `setHostName` method.

(2) Format

```
public synchronized String getHostName() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the host name of the HADB server that was specified by the `setHostName` method. If the `setHostName` method has not been executed, the method returns `null`.

(5) Exceptions

None.

10.5.11 getPort()

(1) Function

This method acquires the port number of the HADB server that was specified by the `setPort` method.

(2) Format

```
public synchronized int getPort() throws SQLException
```

(3) Arguments

None.

(4) Return value

This method returns the port number of the HADB server that was specified by the `setPort` method. If the `setPort` method has not been executed, the method returns `-1`.

(5) Exceptions

None.

10.5.12 setApName(String name)

(1) Function

This method specifies the application identifier for connecting to the HADB server.

(2) Format

```
public synchronized void setApName(String name) throws SQLException
```

(3) Arguments

`String name`

Specifies an application identifier as a character string with a length of 1 to 30 bytes. If a space, a character string with a length of zero, or `null` is specified, the JDBC driver assumes that no application identifier is specified by this method.

If an application program is connected to the HADB server without its application identifier being specified anywhere, `"********"` is set for the application identifier.

(4) Return value

None.

(5) Exceptions

If the value specified for `name` is invalid (such as a character string of more than 30 bytes), the JDBC driver throws an `SQLException`.

(6) Notes

The application identifier specified by this method is converted in accordance with the Java Virtual Machine's (JVM) default conversion character set. Therefore, we recommend that you specify an application identifier consisting exclusively of alphanumeric characters that do not depend on the conversion character set.

10.5.13 setEncodeLang(String lang)

(1) Function

This method specifies the name of the conversion character set to be used when conversion of character encoding is performed.

(2) Format

```
public synchronized void setEncodeLang(String lang) throws SQLException
```

(3) Arguments

`String lang`

Specifies a conversion character set. Select a supported conversion character set from the list of *Supported encodings* in *Internationalization support* in the *JavaTM Platform, Standard Edition* JDK document.

(4) Return value

None.

(5) Exceptions

If the specified conversion character set is not supported by the Java Virtual Machine (JVM), the JDBC driver throws an `SQLException`.

(6) Notes

Specify this method only when you want to use a character set other than the supported conversion character set shown in Table 7-15: Names of the character sets supported for the HADB server's character encoding. If you use the supported conversion character set indicated in Table 7-15: Names of the character sets supported for the HADB server's character encoding for conversion, you do not need to specify this method.

10.5.14 setInterfaceMethodTrace(boolean flag)

(1) Function

This method specifies whether JDBC interface method traces are to be acquired. For details about the JDBC interface method traces, see 7.7.1 JDBC interface method traces.

If you acquire JDBC interface method traces, you must specify an output destination with the `setLogWriter` method.

(2) Format

```
public synchronized void setInterfaceMethodTrace(boolean flag) throws SQLException
```

(3) Arguments

`boolean flag`

Specifies whether JDBC interface method traces are to be acquired:

- `true`: Acquire JDBC interface method traces.
- `false`: Do not acquire JDBC interface method traces.

If this method is not executed, the JDBC driver assumes `false`.

(4) Return value

None.

(5) Exceptions

None.

(6) Notes

Whether JDBC interface method traces are to be acquired cannot be specified for each instance individually. The setting specified by this method applies to all `DataSource` and `ConnectionPoolDataSource` instances that exist at the time this setting is specified and thereafter.

10.5.15 setNotErrorOccurred(boolean mode)

(1) Function

This method specifies the setting as to whether generation of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.

(2) Format

```
public synchronized void setNotErrorOccurred(boolean mode) throws SQLException
```

(3) Arguments

`boolean mode`

Specifies whether generation of `ConnectionEventListener.connectionErrorOccurred` is to be suppressed.

The permitted value is as follows:

- `true`: Suppress generation of `connectionErrorOccurred`.
- `false`: Do not suppress generation of `connectionErrorOccurred`.

If this method is not executed, the JDBC driver assumes `false`.

If `ConnectionPoolDataSource` is used, this argument specifies the setting that suppresses calling of `ConnectionEventListener.connectionErrorOccurred`, which is called if a fatal connection error occurs. Normally, this argument is not specified or is set to `false`.

(4) Return value

None.

(5) Exceptions

None.

10.5.16 setPassword(String password)

(1) Function

This method specifies a password for the authorization identifier that is used to connect to the HADB server.

When the methods listed below are executed, the authorization identifier and password specified by `setUser` and `setPassword`, respectively, are used to connect to the HADB server.

- `getConnection` method of the `DataSource` interface (with no argument specified)
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface

Specification priorities apply to the password. For details, see 7.3.3 Connection information priorities.

(2) Format

```
public synchronized void setPassword(String password) throws SQLException
```

(3) Arguments

`String password`

Specifies a password for the authorization identifier that is to be used to connect to the HADB server. If `null` is specified, the JDBC driver assumes that no password is specified by this method.

(4) Return value

None.

(5) Exceptions

None.

10.5.17 setSQLWarningKeep(boolean mode)

(1) Function

This method specifies the setting as to whether warning information generated during execution of SQL statements is to be retained.

(2) Format

```
public synchronized void setSQLWarningKeep(boolean mode) throws SQLException
```

(3) Arguments

`boolean mode`

Specifies one of the following values indicating whether warning information is to be retained:

- `true`: Retain warning information.
- `false`: Do not retain warning information.

If this method is not executed, the JDBC driver assumes `true`.

(4) Return value

None.

(5) Exceptions

None.

10.5.18 setTraceNumber(int num)

(1) Function

This method specifies the number of entries for a JDBC interface method trace.

(2) Format

```
public synchronized void setTraceNumber(int num) throws SQLException
```

(3) Arguments

`int num`

Specifies the number of entries for a JDBC interface method trace, in the range from 10 to 1,000. If this method is not executed, the number of entries is set to 500.

(4) Return value

None.

(5) Exceptions

If the value specified for the number of entries is outside the range from 10 to 1,000, the JDBC driver throws an `SQLException`.

10.5.19 setUser(String user)

(1) Function

This method specifies the authorization identifier to be used to connect to the HADB server.

When the methods listed below are executed, the authorization identifier and password specified by `setUser` and `setPassword`, respectively, are used to connect to the HADB server.

- `getConnection` method of the `DataSource` interface (with no argument specified)
- `getPooledConnection` method of the `ConnectionPoolDataSource` interface

Specification priorities apply to the authorization identifier. For details, see 7.3.3 Connection information priorities.

(2) Format

```
public synchronized void setUser(String user) throws SQLException
```

(3) Arguments

`String user`

Specifies the authorization identifier that is to be used to connect to the HADB server. If `null` is specified, the JDBC driver assumes that no authorization identifier is specified by this method.

(4) Return value

None.

(5) Exceptions

If the length of the character string specified for `user` is zero, the JDBC driver throws an `SQLException`.

10.5.20 setHostName(String name)

(1) Function

This method specifies the host name of the HADB server at the connection destination.

Specification priorities apply to the host name of the HADB server. For details, see 7.3.3 Connection information priorities.

(2) Format

```
public synchronized void setHostName(String name) throws SQLException
```

(3) Arguments

`String name`

Specifies the host name of the HADB server at the connection destination. If `null` is specified, the JDBC driver assumes that no host name is specified by this method.

(4) Return value

None.

(5) Exceptions

If the value specified for `name` is invalid (such as a character string with a length of 0 bytes or less or 256 bytes or greater), the JDBC driver throws an `SQLException`.

10.5.21 setPort(int port)

(1) Function

This method specifies the port number of the HADB server at the connection destination.

Specification priorities apply to the port number of the HADB server. For details, see 7.3.3 Connection information priorities.

(2) Format

```
public synchronized void setPort(int port) throws SQLException
```

(3) Arguments

`int port`

Specifies the port number of the HADB server at the connection destination, in the range from `5001` to `65535`.

(4) Return value

None.

(5) Exceptions

If the specified `port` argument value is outside the range from `5001` to `65535`, the JDBC driver throws an `SQLException`.

11 The JDBC 3.0 API

This chapter describes the interfaces and methods in the JDBC 3.0 API.

11.1 HADB's scope of support for the functions added in the JDBC 3.0 API

The following table shows HADB's scope of support for the functions added in the JDBC 3.0 API.

Table 11-1: HADB's scope of support for the functions added in the JDBC 3.0 API

Added function	Corresponding interface	HADB's scope of support
Save point	`Connection`	N
	`Savepoint`	N
Enhancement of the connection pool function	`PreparedStatement`	N
Parameter metadata	`ParameterMetaData`	Y
	`PreparedStatement`	Y
Automatic generation key	`Connection`	N
	`DatabaseMetaData`	N
	`Statement`	N
Holdable cursor	`Connection`	Y
	`DatabaseMetaData`	Y
	`Statement`	Y
	`ResultSet`	Y
API for adding database metadata	`DatabaseMetaData`	Y

Legend:

Y: Supported by HADB.

N: Not supported by HADB.

11.2 ParameterMetaData interface

This section explains the methods provided by the `ParameterMetaData` interface.

11.2.1 List of the methods in the ParameterMetaData interface

(1) Main functions of the ParameterMetaData interface

The `ParameterMetaData` interface provides the following main function:

- Return of meta information, such as the data types and data lengths of parameters in the `PreparedStatement` object

(2) Methods in the ParameterMetaData interface that are supported by HADB

The following table lists and describes the methods in the `ParameterMetaData` interface that are supported by HADB.

Table 11-2: Methods in the ParameterMetaData interface

No.	Method of the ParameterMetaData interface	Function
1	`getParameterClassName(int param)`	Acquires the fully specified Java class name for the data type of a parameter.
2	`getParameterCount()`	Acquires the number of parameters in the `PreparedStatement` object.
3	`getParameterMode(int param)`	Acquires a specified parameter's mode.
4	`getParameterType(int param)`	Acquires a specified parameter's SQL data type.
5	`getParameterTypeName(int param)`	Acquires a specified parameter's data type.
6	`getPrecision(int param)`	Acquires the number of digits in a specified parameter.
7	`getScale(int param)`	Acquires the number of decimal places in a specified parameter.
8	`isNullable(int param)`	Returns a value indicating whether the null value can be set in a specified parameter.
9	`isSigned(int param)`	Returns a value indicating whether a specified parameter is for signed numeric values.

Important

HADB does not support methods that are not listed in this table. If an unsupported method is executed, an `SQLException` might be thrown.

(3) Required parameter name and class name

The package and class names required to use the `ParameterMetaData` interface are as follows:

- Package name: `com.hitachi.hadb.jdbc`
- Class name: `AdbParameterMetaData`

11.2.2 getParameterClassName(int param)

(1) Function

This method acquires the fully specified Java class name for the data type of a parameter.

(2) Format

```
public synchronized String getParameterClassName(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns a `String` object.

This method returns the `String` Java class type that is used by the `setObject` method of the `PreparedStatement` object for the parameter. The following table shows the parameter data types and corresponding return values.

Table 11-3: Character strings returned when the getParameterClassName method is executed

Parameter's data type (HADB data type)	Character string returned
INTEGER	"java.lang.Long"
SMALLINT	"java.lang.Integer"
DOUBLE PRECISION	"java.lang.Double"
DECIMAL	"java.math.BigDecimal"
CHAR	"java.lang.String"
VARCHAR	"java.lang.String"
DATE	"java.sql.Date"
TIME	"java.sql.Time"
TIMESTAMP	"java.sql.Timestamp"
BINARY	"java.lang.Object"
VARBINARY	"java.lang.Object"
ROW	"java.sql.Object"

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.3 getParameterCount()

(1) Function

This method acquires the number of parameters in the `PreparedStatement` object.

(2) Format

```
public synchronized int getParameterCount() throws SQLException
```

(3) Arguments

None.

(4) Return value

The method returns the number of parameters in the `PreparedStatement` object.

(5) Exceptions

None.

11.2.4 getParameterMode(int param)

(1) Function

This method acquires a specified parameter's mode.

(2) Format

```
public synchronized int getParameterMode(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method always returns `ParameterMetaData.parameterModeIn`.

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.5 getParameterType(int param)

(1) Function

This method acquires a specified parameter's SQL data type.

(2) Format

```
public synchronized int getParameterType(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns the SQL type obtained from `java.sql.Types`.

For details about the correspondence between data types and return values of columns, see (1) Correspondence between HADB's data types and JDBC's SQL data types in 7.6.1 Mapping data types.

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.6 getParameterTypeName(int param)

(1) Function

This method acquires a specified parameter's data type.

(2) Format

```
public synchronized String getParameterTypeName(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns a `String` object. The following table shows the return values of the `getParameterTypeName` method.

Table 11-4: Return values of the getParameterTypeName method

Parameter's data type (HADB data type)	Return value (character string returned)
INTEGER	"INTEGER"
SMALLINT	"SMALLINT"
DECIMAL	"DECIMAL"
DOUBLE PRECISION	"DOUBLE PRECISION"
CHAR	"CHAR"
VARCHAR	"VARCHAR"
DATE	"DATE"
TIME	"TIME"
TIMESTAMP	"TIMESTAMP"
BINARY	"BINARY"
VARBINARY	"VARBINARY"
ROW	"ROW"

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.7 getPrecision(int param)

(1) Function

This method acquires the number of digits in a specified parameter.

(2) Format

```
public synchronized int getPrecision(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns a decimal number indicating the number of digits in the specified parameter.

If the specified parameter is the numeric data type, the method returns the number of digits. If it is not the numeric data type, the method returns the parameter length in bytes. The following table shows the return values of the `getPrecision` method.

Table 11-5: Return values of the getPrecision method

Parameter's data type (HADB data type)	Return value (column length in digits)
`INTEGER`	`19`
`SMALLINT`	`10`
`DOUBLE PRECISION`	`17`
`DECIMAL(`*m*`,`*n*`)`	*m*
`CHAR(`*n*`)` `VARCHAR(`*n*`)`	*n*
`DATE`	`10`
`TIME(`*p*`)`	*p* = 0: `8` *p* > 0: 8 + (*p* + 1)
`TIMESTAMP(`*p*`)`	*p* = 0: `19` *p* > 0: 19 + (*p* + 1)
`BINARY(`*n*`)` `VARBINARY(`*n*`)`	*n*
`ROW`	*row-length*#

#

Sum of the data lengths of all columns. For details about how to obtain the data length of a column, see *Length of data storage* in the topic *List of data types* in the manual *HADB SQL Reference*.

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.8 getScale(int param)

(1) Function

This method acquires the number of decimal places in a specified parameter.

(2) Format

```
public synchronized int getScale(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns a decimal number indicating the number of decimal places in the specified column. The following table shows the return values of the `getScale` method.

Table 11-6: Return values of the getScale method

Parameter's data type (HADB data type)	Return value (number of decimal places)
`DECIMAL(`*m*`,`*n*`)`	*n*
`TIME(`*p*`)` `TIMESTAMP(`*p*`)`	*p*
Other than the above	`0`

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.9 isNullable(int param)

(1) Function

This method returns a value indicating whether the null value can be set in a specified parameter.

(2) Format

```
public synchronized int isNullable(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns one of the following values:

- `ParameterMetaData.parameterNoNulls`: The null value cannot be set.
- `ParameterMetaData.parameterNullable`: The null value can be set.

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.2.10 isSigned(int param)

(1) Function

This method returns a value indicating whether a specified parameter is for signed numeric values.

(2) Format

```
public synchronized boolean isSigned(int param) throws SQLException
```

(3) Arguments

`int param`

Specifies a parameter number (beginning with 1).

(4) Return value

The method returns one of the following values:

- `true`: The parameter is for signed numeric values.
- `false`: The parameter is not for signed numeric values.

The following table shows the relationship between the parameter's data type and the return value.

Table 11-7: Relationship between the parameter's data type and the return value

Parameter's data type	Return value
INTEGER, SMALLINT, DOUBLE PRECISION, DECIMAL	true
Other than the above	false

(5) Exceptions

If the specified `param` value is `0` or less, or is greater than the number of parameters, the JDBC driver throws an `SQLException`.

11.3 Unsupported interfaces

HADB does not support the following interface:

- `Savepoint`

12 The JDBC 4.0 API

This chapter describes the interfaces and methods in the JDBC 4.0 API.

12.1 HADB's scope of support for the functions added in the JDBC 4.0 API

The following table shows HADB's scope of support for the functions added in the JDBC 4.0 API.

Table 12-1: HADB's scope of support for the functions added in the JDBC 4.0 API

Added function	Corresponding interface	HADB's scope of support
Automatic loading of `java.sql.Driver`	--	Y
`ROWID` data type	`CallableStatement`	N
	`PreparedStatement`	N
	`RowId`	N
	`DatabaseMetaData`	N
National character data type	`PreparedStatement`	N
	`ResultSet`	N
XML support	`SQLXML`	N
Wrapper pattern	`Connection`	Y
	`DatabaseMetaData`	Y
	`DataSource`	Y
	`ResultSet`	Y
	`ResultSetMetaData`	Y
	`Statement`	Y
	`PreparedStatement`	Y
	`ParameterMetaData`	Y
SQL exception extension	All interfaces	Y
Connection management	`Connection`	Y
	`Statement`	Y
Added scalar functions	`Connection`	Y
	`DatabaseMetaData`	Y

Legend:

--: There is no corresponding interface.

Y: Supported by HADB.

N: Not supported by HADB.

12.1.1 Automatic loading of java.sql.Driver

In JDBC 4.0 API, there is no need to register the `Driver` class. However, processing that explicitly registers the `Driver` class causes no problem.

12.1.2 Wrapper pattern

In the JDBC 4.0 API, the following interfaces inherit the `Wrapper` interface:

- `Connection`
- `DatabaseMetaData`
- `DataSource`
- `ResultSet`
- `ResultSetMetaData`
- `Statement`
- `PreparedStatement`
- `ParameterMetaData`

12.1.3 SQL exception extension

Multiple exception classes have been added as subclasses of `SQLException`. The following table lists the added exception classes and whether each is supported by HADB.

Table 12-2: Added exception classes and whether each is supported by HADB

No.	Exception class name	HADB's scope of support
1	`SQLNonTransientException`	Y
2	`SQLFeatureNotSupportedException`	Y
3	`SQLNonTransientConnectionException`	Y
4	`SQLDataException`	Y
5	`SQLIntegrityConstraintViolationException`	Y
6	`SQLInvalidAuthorizationSpecException`	Y
7	`SQLSyntaxErrorException`	Y
8	`SQLTransientException`	Y
9	`SQLTransientConnectionException`	Y
10	`SQLTransactionRollbackException`	Y
11	`SQLTimeoutException`	Y
12	`SQLRecoverableException`	Y
13	`SQLClientInfoException`	Y

Legend:

Y: Supported by HADB.

For details about SQL exception extension, see 12.3 SQL exception extension function.

12.1.4 Connection management

This driver supports the following interface methods:

- `isValid` method of the `Connection` interface
- `isPoolable` method of the `Statement` interface

12.1.5 Added scalar functions

The following table shows HADB's scope of support for the scalar functions added in the JDBC 4.0 API.

Table 12-3: Scalar functions supported by HADB

No.	Scalar function	HADB's scope of support
1	CHAR_LENGTH	N
2	CHARACTER_LENGTH	N
3	CURRENT_DATE	Y
4	CURRENT_TIME	Y
5	CURRENT_TIMESTAMP	Y
6	EXTRACT	Y
7	OCTET_LENGTH	N
8	POSITION	N

Legend:

Y: Supported by HADB.

N: Not supported by HADB.

12.2 Wrapper interface

This section explains the methods provided by the `Wrapper` interface.

12.2.1 List of the methods in the Wrapper interface

(1) Main functions of the Wrapper interface

The `Wrapper` interface provides a standardized approach for calling methods that are not JDBC-compliant.

(2) Methods in the Wrapper interface that are supported by HADB

The following table lists and describes the methods in the `Wrapper` interface that are supported by HADB.

Table 12-4: Methods in the Wrapper interface

No.	Method of the Wrapper interface	Function
1	`isWrapperFor(Class<?> iface)`	Returns a value indicating whether a specified class's object can be returned by the `unwrap` method.
2	`unwrap(Class<T> iface)`	Returns a specified class's object.

(3) Classes that can be specified

The following table lists the classes that can be specified in `unwrap` for an interface that has inherited `Wrapper`.

Table 12-5: Classes that can be specified in unwrap

No.	Interface	Class that can be specified in unwrap
1	`java.sql.Connection`	`AdbConnection`
2	`java.sql.DatabaseMetaData`	`AdbDatabaseMetaData`
3	`javax.sql.DataSource`	`AdbDataSource`
4	`java.sql.ResultSet`	`AdbResultSet`
5	`java.sql.ResultSetMetaData`	`AdbResultSetMetaData`
6	`java.sql.Statement`	`AdbStatement`
7	`java.sql.PreparedStatement`	`AdbPreparedStatement`
8	`java.sql.ParameterMetaData`	`AdbParameterMetaData`

(4) Example coding

The following shows an example of `Wrapper` interface coding:

```
Connection con = DriverManager.getConnection(url,info);
Class<?> clazz = Class.forName("com.hitachi.hadb.jdbc.AdbConnection");
if(con.isWrapperFor(clazz)){
  AdbConnection acon = (AdbConnection)con.unwrap(clazz);
```

```
  acon.xxxx();
}
```

12.2.2 isWrapperFor(Class<?> iface)

(1) Function

This method returns a value indicating whether a specified class's object can be returned by the `unwrap` method.

(2) Format

```
public synchronized boolean isWrapperFor(Class<?> iface) throws SQLException
```

(3) Arguments

`Class<?> iface`

Specifies a class to be checked.

(4) Return value

If the specified class's object can be returned by the `unwrap` method, the method returns `true`; otherwise, the method returns `false`.

(5) Exceptions

None.

12.2.3 unwrap(Class<T> iface)

(1) Function

This method returns a specified class's object.

(2) Format

```
public synchronized <T> T unwrap(Class<T> iface) throws SQLException
```

(3) Arguments

`Class<T> iface`

Specifies a class to be checked.

(4) Return value

The method returns the specified class's object.

(5) Exceptions

If the specified class's object cannot be returned, the JDBC driver throws an `SQLException`.

12.3 SQL exception extension function

Multiple exception classes have been added as subclasses of `SQLException`. The following table lists the exception classes that are returned by JDBC 4.0 API and describes each class.

Table 12-6: List of exception classes returned by JDBC 4.0 API and description of the classes

No.	Exception class name	Description	Connection status if an error occurs while connected
1	`SQLNonTransientException`	Indicates an error that is not transient. This exception is thrown when the SQL statement resulting in an error cannot be re-executed successfully.	Valid
2	`SQLFeatureNotSupportedException`	Thrown when the class value of `SQLSTATE` is `0A` (unsupported function).	Valid
3	`SQLNonTransientConnectionException`	Thrown when the class value of `SQLSTATE` is `08` (connection violation).	--
4	`SQLDataException`	Thrown when the class value of `SQLSTATE` is `22` (data exception).	Valid
5	`SQLIntegrityConstraintViolationException`	Thrown when the class value of `SQLSTATE` is `23` (integrity constraint violation).	Valid
6	`SQLInvalidAuthorizationSpecException`	Thrown when the class value of `SQLSTATE` is `28` (specified authorization identifier is invalid).	--
7	`SQLSyntaxErrorException`	Thrown when the class value of `SQLSTATE` is `42` (syntax error or access rule violation).	Valid
8	`SQLTransientException`	Indicates a transient error. This exception is thrown when the SQL statement resulting in an error might be re-executed successfully.	Valid
9	`SQLTransientConnectionException`	Thrown when the class value of `SQLSTATE` is `08` (connection violation). This applies, for example, when the HADB server is starting or terminating.	--
10	`SQLTransactionRollbackException`	Thrown when the class value of `SQLSTATE` is `40` (transaction rolled back).	Valid
11	`SQLTimeoutException`	Thrown when a timeout occurs.	Valid
12	`SQLRecoverableException`	Thrown when the transaction resulting in an error might be re-executed successfully after a connection is re-established.	Invalid
13	`SQLClientInfoException`	Thrown by the `Connection.setClientInfo` method when there is one or more client properties that cannot be set.	--

Legend:

--: Not applicable.

Note

For details about the class values of `SQLSTATE`, see the topic *SQLSTATE output format* in the manual *HADB Messages*.

The following shows the inheritance relationships of the exception classes that have been added in JDBC 4.0 API.

```
java.sql.SQLException
  |
  ├ java.sql.SQLNonTransientException
  |     ├ java.sql.SQLFeatureNotSupportedException
  |     ├ java.sql.SQLNonTransientConnectionException
  |     ├ java.sql.SQLDataException
  |     ├ java.sql.SQLIntegrityConstraintViolationException
  |     ├ java.sql.SQLInvalidAuthorizationSpecException
  |     └ java.sql.SQLSyntaxErrorException
  |
  ├ java.sql.SQLTransientException
  |     ├ java.sql.SQLTransientConnectionException
  |     ├ java.sql.SQLTransactionRollbackException
  |     └ java.sql.SQLTimeoutException
  |
  ├ java.sql.SQLRecoverableException
  └ java.sql.SQLClientInfoException
```

Each exception class directly uses the classes of the `java.sql` package. For details about and usage of individual methods provided by the exception classes, see the related JDBC standard documentation.

Important

If a method such as `executeQuery` that executes SQL statements times out by exceeding the time specified in the `setQueryTimeout` method or `adb_clt_rpc_sql_wait_time`, the JDBC standard dictates that `SQLTimeoutException` is thrown. However, because the connection with the HADB server is always lost if a timeout occurs when a HADB client is waiting for the HADB server to respond to a processing request, `SQLRecoverableException` is thrown instead.

12.4 Unsupported interfaces

HADB does not support the following interfaces:

- `NClob`
- `RowId`
- `SQLXML`

13 The JDBC 4.1 API

This chapter describes the scope of support in HADB for the functions added in the JDBC 4.1 API.

13.1 HADB's scope of support for the functions added in the JDBC 4.1 API

The following table lists HADB's scope of support for the functions added in the JDBC 4.1 API.

Table 13-1: HADB's scope of support for the functions added in the JDBC 4.1 API

Added function	Corresponding interface	Supported in HADB
`try-with-resources` statement	`Connection`	Y
	`ResultSet`	
	`Statement`	
Specification of `Java` data type as the conversion destination for a `getObject` method	`CallableStatement`	Y#
	`ResultSet`	
Acquisition of parent Logger	`Driver`	N
	`DataSource`	
	`ConnectionPoolDataSource`	
Schema specification	`Connection`	N
Stop and time out physical connections	`Connection`	N
Closing `Statement` objects when their dependent objects close	`Statement`	Y
API that adds database metadata	`DatabaseMetaData`	Y

Legend:

Y: Supported by HADB.

N: Not supported by HADB.

#

The `getObject` method of the `ResultSet` interface Only some conversions to the `Java` data type are supported. For details, see 8.5.43 getObject(int columnIndex,Class<T> type).

13.1.1 try-with-resources statement

This statement is a try statement that declares resources. By using a `try-with-resources` statement, you can ensure that each resource is closed at the end of the statement.

You can use a `try-with-resources` statement in a `Connection` interface, `ResultSet` interface, or `Statement` interface. For details about how to use this statement, see the related documentation about the JDBC specification.

13.1.2 Closing Statement objects when their dependent objects close

HADB supports the `closeOnCompletion` and `isCloseOnCompletion` methods in the `Statement` interface.

14 The JDBC 4.2 API

This chapter describes the scope of support in HADB for the functions added in the JDBC 4.2 API.

14.1 HADB's scope of support for the functions added in the JDBC 4.2 API

The following table lists HADB's scope of support for the functions added in the JDBC 4.2 API.

Table 14-1: HADB's scope of support for the functions added in the JDBC 4.2 API

Added function	Corresponding interface	Supported in HADB
`REF CURSOR`	`CallableStatement`	N
`SQLType` interface	`SQLType`	N
	`CallableStatement`	
	`PreparedStatement`	
	`ResultSet`	
	`SQLOutput`	
`JDBCType Enum`	`SQLType`	N
Large update counts	`PreparedStatement`	Y
	`Statement`	
API that adds database metadata	`DatabaseMetaData`	Y

Legend:

Y: Supported by HADB.

N: Not supported by HADB.

14.1.1 Large update counts

HADB supports the following methods that handle the updated row count of update SQL statements as `long` values.

PreparedStatement interface

- `executeLargeUpdate`

Statement interface

- `getLargeUpdateCount`
- `setLargeMaxRows`
- `getLargeMaxRows`
- `executeLargeBatch`
- `executeLargeUpdate`

Use these methods when executing the following SQL statements if the number of updated rows might exceed `Integer.MAX_VALUE`:

- `UPDATE`
- `INSERT`
- `DELETE`

15 Creating Application Programs

This chapter explains the environment setup for an HADB ODBC driver and points to be taken into consideration during the creation of application programs that support ODBC.

15.1 ODBC driver provided by HADB

You can use the ODBC driver provided by HADB (referred to hereafter as the *HADB ODBC driver*) to access the HADB database. This chapter explains the scope of the ODBC driver with which the HADB ODBC driver is compliant and the system configuration for the HADB ODBC driver.

15.1.1 ODBC driver version with which the HADB ODBC driver is compliant

The HADB ODBC driver complies with ODBC 3.5. The HADB ODBC driver enables you to use the ODBC interfaces to access the HADB database and link with the BI tools.

15.1.2 System configuration

(1) Prerequisite programs

The HADB ODBC driver requires the following programs.

■ **HADB ODBC driver (64-bit mode)**

Supported operating systems:

- Windows Server 2008 R2
- Windows Server 2012
- Windows Server 2012 R2
- Windows Server 2016
- Windows 7 (x64)
- Windows 8.1 (x64)
- Windows 10 (x64)

Required software:

- Microsoft Data Access Components
- HADB client

■ **HADB ODBC driver (32-bit mode)**

Supported operating systems:

- Windows 7
- Windows 7 (x64)
- Windows 8.1
- Windows 8.1 (x64)
- Windows 10
- Windows 10 (x64)

Required software:

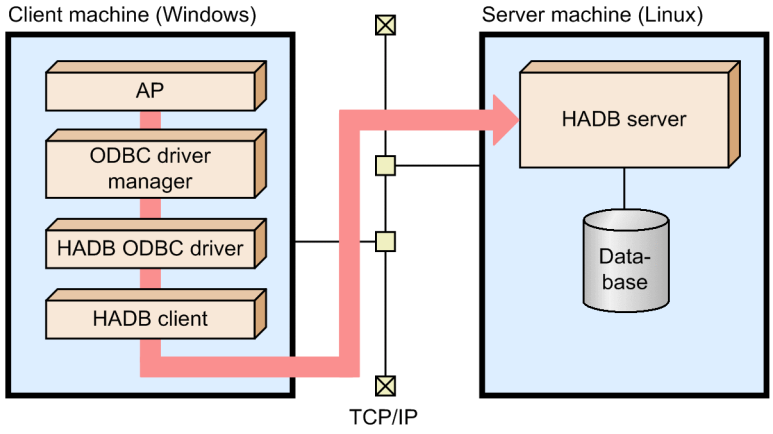
- Microsoft Data Access Components

- HADB client

(2) Device configuration

The following figure shows the system configuration for using the HADB ODBC driver.

Figure 15-1: System configuration for using the HADB ODBC driver



15.1.3 About conversion of character encoding

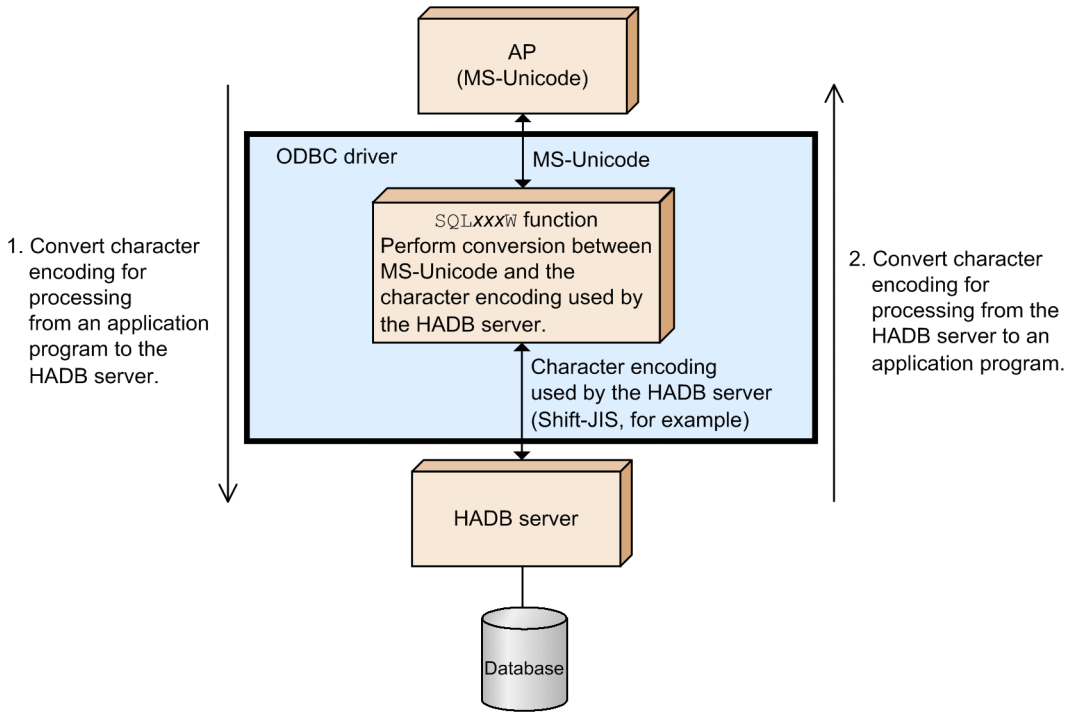
(1) Character encoding conversion processing

The ODBC driver provided by HADB supports ODBC 3.5 that includes the MS-Unicode interfaces.

When an `SQLxxxW` function, which is an MS-Unicode interface, is used, the ODBC driver performs conversion between the MS-Unicode character string data and the character encodings supported by HADB.

The following figure shows how the ODBC driver performs conversion between MS-Unicode character string data and the character encodings supported by HADB.

Figure 15-2: Conversion between MS-Unicode character string data and the character encodings supported by HADB



Explanation:

1. When the character encoding is converted from an application program to the HADB server for processing, the ODBC driver uses an SQL*xxx*W function to convert MS-Unicode character string data to an HADB-supported character encoding, and then passes the processing to the HADB server.
2. When the character encoding is converted from the HADB server to an application program for processing, the ODBC driver uses an SQL*xxx*W function to convert the HADB-supported character encoding to MS-Unicode character string data, and then outputs the results to the application program.

HADB supports the following character encodings:

- Unicode (UTF-8)
- Shift-JIS

(2) Notes

- Characters with a diacritical mark

  If a character with a diacritical mark, such as an umlaut (ä, ö, ü) or a dieresis (ë, ï, ü, ÿ), is compared, assigned to a variable, or retrieved, the character might be treated as not having the diacritical mark.

  This is because diacritical marks are lost when the character encoding is converted by using an SQL*xxx*W function.

- Characters such as surrogate pairs that do not support conversion of character encoding

  When you use a SQL*xxx*W function of the ODBC 3.5 interface, character encoding will be converted. An error occurs if a character such as a surrogate pair whose character encoding cannot be converted is compared or undergoes storage assignment. Characters of this nature might be replaced with # during retrieval, and characters that appear after this # in the retrieval results might become garbled.

15.1.4 About using the ODBC cursor library

Use of the ODBC cursor library provided by Microsoft makes the following functionality available in HADB.

(1) Scrollable cursors

By specifying `SQL_SCROLLABLE` in the `SQL_ATTR_CURSOR_SCROLLABLE` attribute, you can:

- Use `SQLFetchScroll` to fetch rowset data
- Use `SQLSetPos` to position the cursor at a certain row in the rowset

However, you cannot use `SQLSetPos` to update the rowset data to the latest state. You also cannot use `SQLSetPos` to update or delete data in result sets.

(2) Bookmark functionality

You can use the following procedure to acquire a bookmark for the row where the cursor is positioned:

1. Set the `SQL_ATTR_USE_BOOKMARKS` attribute to `SQL_UB_VARIABLE`.
2. Use `SQLSetPos` to position the cursor at the desired row in the rowset.
3. Acquire data from column `0`.

Also, by specifying a bookmark in the `SQL_ATTR_FETCH_BOOKMARK_PTR` attribute, you can specify the bookmark in the first row when executing `SQLFetchScroll`.

15.2 HADB ODBC driver environment setup

This section explains the HADB ODBC driver environment setup. The tasks explained here are performed by an OS user with the administrator permissions.

15.2.1 Specifying data sources

The following explains how to specify a data source.

To specify a data source:

1. Run **ODBC Data Source Administrator (odbcad32.exe)**.

   The ODBC Data Source Administrator to be run depends on the OS and on the BI tool's operating mode on the client machine on which the HADB client is installed. The following table shows the folder in which the ODBC Data Source Administrator is stored.

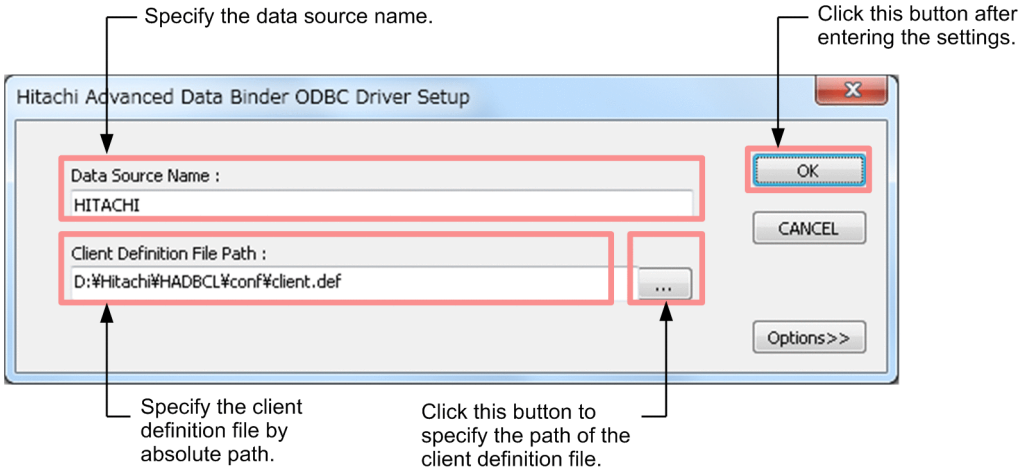
   Table 15-1: Folder in which ODBC Data Source Administrator is stored

No.	OS on the client machine	BI tool's operating mode	HADB ODBC driver type	Folder in which ODBC Data Source Administrator is stored
1	32-bit	32-bit	32-bit	`%windir%\system32\odbcad32.exe`
2	64-bit	64-bit	64-bit	
3		32-bit (WOW64)	32-bit	`%windir%\SysWOW64\odbcad32.exe`
4			64-bit	--

   Legend:

   --: This combination is not supported.

2. From the tab items, select the type of data source to be registered or modified, and then click the **Add** button.
3. The Create New Data Source dialog box is displayed. Select **Hitachi Advanced Data Binder ODBC Driver**.
4. The Hitachi Advanced Data Binder ODBC Driver Setup dialog is displayed. Specify all items, and then click the **OK**.



   Specify a data source name as a string of no more than 32 bytes. If you are entering double-byte characters, note that each displayed character is not equal to one byte.

   Specify a client definition file path as a string of no more than 255 bytes.

Note

- If you intend to output HADB ODBC driver trace information, click **Options>>**. For details about the specification method, see 17.3.1 Configuration in ODBC Data Source Administrator.
- To cancel a data source specification, click the **CANCEL**.

5. The registered data source is displayed. Check the registered information.

15.2.2 Registering the registry key

When you set up the HADB client environment, use the registry registration command to check if the registry key has been registered. For details about how to register a registry key, see (1) Installing an HADB client in 4.2.1 HADB client for Windows.

15.2.3 Deleting data sources

The following explains how to delete a data source.

To delete a data source:

1. Run **ODBC Data Source Administrator (odbcad32.exe)**.
   The ODBC Data Source Administrator to be run depends on the OS and on the BI tool's operating mode on the client machine on which the HADB client is installed. The following table shows the folder in which the ODBC Data Source Administrator is stored.

   Table 15-2: Folder in which ODBC Data Source Administrator is stored

No.	OS on the client machine	BI tool's operating mode	HADB ODBC driver type	Folder in which ODBC Data Source Administrator is stored
1	32-bit	32-bit	32-bit	`%windir%\system32\odbcad32.exe`
2	64-bit	64-bit	64-bit	
3		32-bit (WOW64)	32-bit	`%windir%\SysWOW64\odbcad32.exe`
4			64-bit	--

   Legend:
   --: This combination is not supported.

2. From the tab items, select the type of data source to be deleted, and then select the data source name to be deleted.
3. Click the **Remove** button.

15.3 Correspondence between data types

This section explains the correspondence between data types.

15.3.1 Correspondence between ODBC's SQL data types and HADB's data types

The following table shows the correspondence between ODBC's SQL data types and HADB's data types.

Table 15-3: Correspondence between ODBC's SQL data types and HADB's data types

Classification	ODBC's SQL data type	HADB's corresponding data type	Description	Supported
Character string data	SQL_CHAR	CHARACTER	Fixed-length character string	Y
	SQL_VARCHAR	VARCHAR	Variable-length character string	Y
	SQL_LONGVARCHAR	--	Variable-length character string	N
	SQL_WCHAR	--	Fixed-length character string (for Unicode)	N
	SQL_WVARCHAR	--	Variable-length character string (for Unicode)	N
	SQL_WLONGVARCHAR	--	Variable-length character string (for Unicode)	N
Numeric data	SQL_DECIMAL	DECIMAL	Fixed-point number	Y
	SQL_NUMERIC	--	Fixed-point number	N
	SQL_TINYINT	--	1-byte integer	N
	SQL_SMALLINT	--	2-byte integer	Y[#1]
	SQL_INTEGER	SMALLINT	4-byte integer	Y
	SQL_BIGINT	INTEGER	8-byte integer	Y
	SQL_REAL	--	Single-precision floating point number	N
	SQL_FLOAT	--	Double-precision floating-point number	N
	SQL_DOUBLE	DOUBLE PRECISION	Double-precision floating-point number	Y
	SQL_BIT	--	Bit	N
	SQL_BINARY	BINARY	Fixed-length binary data	Y
	SQL_VARBINARY	VARBINARY	Variable-length binary data	Y
	SQL_LONGVARBINARY	--	Variable-length binary data	N
Date and time data	SQL_TYPE_DATE(, SQL_DATE)[#2]	DATE	Date	Y
	SQL_TYPE_TIME(, SQL_TIME)[#2]	TIME	Time	Y

Classification	ODBC's SQL data type	HADB's corresponding data type	Description	Supported
	SQL_TYPE_TIMESTAMP(, SQL_TIMESTAMP)#2	TIMESTAMP	Time stamp	Y
	SQL_INTERVAL_MONTH	--	Interval in months	N
	SQL_INTERVAL_YEAR	--	Interval in years	N
	SQL_INTERVAL_YEAR_TO_MONTH	--	--	N
	SQL_INTERVAL_DAY	--	Interval in days	N
	SQL_INTERVAL_HOUR	--	Interval in hours	N
	SQL_INTERVAL_MINUTE	--	Interval in minutes	N
	SQL_INTERVAL_SECOND	--	Interval in seconds	N
	SQL_INTERVAL_DAY_TO_HOUR	--	--	N
	SQL_INTERVAL_DAY_TO_MINUTE	--	--	N
	SQL_INTERVAL_DAY_TO_SECOND	--	--	N
	SQL_INTERVAL_HOUR_TO_MINUTE	--	--	N
	SQL_INTERVAL_HOUR_TO_SECOND	--	--	N
	SQL_INTERVAL_MINUTE_TO_SECOND	--	--	N
Other	SQL_GUID	--	--	N

Legend:

--: There is no corresponding data type.

Y: The data type can be used.

N: The data type cannot be used.

#1

Because HADB does not have a data type that corresponds to SQL_SMALLINT, you cannot use SQL_SMALLINT to access the HADB database. SQL_SMALLINT can be used as an interface for some of the catalog functions provided by the HADB ODBC driver. For example, SQL_SMALLINT can be used in the value of SQL_DATA_TYPE in the result set column of SQLColumns.

#2

SQL_TYPE_DATE, SQL_TYPE_TIME, and SQL_TYPE_TIMESTAMP are the datetime data types used in ODBC 3.0. Use these data types unless you have a reason not to.

The data types in parentheses are the datetime data types for ODBC 2.0. They are handled in the same way as those for ODBC 3.0. Depending on the value specified for the ADBODBAPMODE environment variable, the identifier in parentheses might be returned in metadata.

15.3.2 Correspondence between ODBC's SQL data types and C data types

The following tables show the correspondence between ODBC's SQL data types and C data types in terms of the combinations of data types that can be compared and stored or assigned, and the combinations of data types that can be retrieved.

Table 15-4: Combinations of data types that can be compared and stored or assigned (conversion from C data types to ODBC's SQL data types)

ODBC's SQL data type	C data type																				
	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_BIT	SQL_C_NUMERIC	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_TINYINT	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_SSHORT	SQL_C_USHORT	SQL_C_SHORT	SQL_C_SLONG	SQL_C_ULONG	SQL_C_LONG	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_TYPE_DATE (, SQL_C_DATE)	SQL_C_TYPE_TIME (, SQL_C_TIME)	SQL_C_TYPE_TIMESTAMP (, SQL_C_TIMESTAMP)
SQL_CHAR	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SQL_VARCHAR	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SQL_DECIMAL	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	--	--	--
SQL_INTEGER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	Y	Y	Y	Y	--	--	--
SQL_SMALLINT	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
SQL_BIGINT	Y	Y	Y	Y	Y	Y	Y	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	--	--	--
SQL_DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	--	--	--
SQL_BINARY	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	D	--	--	--
SQL_VARBINARY	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	D	--	--	--
SQL_TYPE_DATE (, SQL_DATE)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	D	--	Y
SQL_TYPE_TIME (, SQL_TIME)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	--	D	Y
SQL_TYPE_TIMESTAMP (, SQL_TIMESTAMP)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	Y	Y	D

Legend:

D: Default conversion or recommended combination.

Y: Can be converted. However, data truncation or precision loss might occur during conversion.

--: Cannot be converted.

Table 15-5: Combinations of data types that can be retrieved (conversion from ODBC's SQL data types to C data types)

ODBC's SQL data type	C data type																			
	SQL_C_CHAR	SQL_C_WCHAR	SQL_C_BIT	SQL_C_NUMERIC	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_TINYINT	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_SSHORT	SQL_C_USHORT	SQL_C_SHORT	SQL_C_SLONG	SQL_C_ULONG	SQL_C_LONG	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_TYPE_DATE (, SQL_C_DATE)	SQL_C_TYPE_TIME (, SQL_C_TIME)	SQL_C_TYPE_TIMESTAMP (, SQL_C_TIMESTAMP)
SQL_CHAR	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SQL_VARCHAR	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
SQL_DECIMAL	D	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	--	--	--
SQL_INTEGER	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	Y	Y	Y	Y	--	--	--
SQL_SMALLINT	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	Y	Y	Y	Y	Y	Y	Y	--	--	--
SQL_BIGINT	Y#	Y	Y	Y	Y	Y	Y	D#	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	--	--	--
SQL_DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	D	Y	--	--	--
SQL_BINARY	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	D	--	--	--
SQL_VARBINARY	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	D	--	--	--
SQL_TYPE_DATE (, SQL_DATE)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	D	--	Y
SQL_TYPE_TIME (, SQL_TIME)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	--	D	Y
SQL_TYPE_TIMESTAMP (, SQL_TIMESTAMP)	Y	Y	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	Y	Y	Y	D

Legend:

D: Default conversion or recommended combination.

Y: Can be converted. However, data truncation or precision loss might occur during conversion.

--: Cannot be converted.

#

If ACCESS is specified in the ADBODBAPMODE environment variable, the default conversion will be SQL_C_CHAR.

15.3.3 Notes about data type conversion

(1) Notes about the BINARY and VARBINARY types

■ Rules for comparing and storing or assigning

- The following rules apply to entering data to HADB's BINARY or VARBINARY type:
  - If the input data is SQL_C_WCHAR, after conversion to CHAR data, processing equivalent to conversion from SQL_C_CHAR to binary is performed.
  - If the input data is character string data, every 2 bytes of data is converted to 1 byte (8 bits) of binary data. If the input data is SQL_C_WCHAR, this processing is applied after the data is converted to SQL_C_CHAR. Two bytes of character string data represents one hexadecimal number. For example, 01 is converted to the binary 00000001, and FF is converted to the binary 11111111. Accordingly, an error occurs when half the byte length of the input data exceeds the definition length of the input destination.
- The following rules apply to entering SQL_C_BINARY type data to an HADB data type:

- When the input destination is a data type that expects character string data (such as SQL_CHAR), an error occurs when the actual length of the input data exceeds the definition length of the target data.

■ Rules for retrieval

- The following rule applies when data is output from HADB's BINARY or VARBINARY type:
  - When receiving the data as character string data, each byte (8 bits) of the source data is represented by two ASCII characters. These characters are an ASCII representation of a hexadecimal number. For example, the binary 00000001 is converted to 01, and the binary 11111111 is converted to FF. If BuffeLength is less than twice the byte length of the source data (further doubled when the data is SQL_C_WCHAR), then the data is truncated and a warning output to that effect.

(2) Notes about the DATE, TIME, and TIMESTAMP types

- If date data of the DATE or CHAR type is converted to time stamp data of the TIMESTAMP type, zeros are set in the time part of the time stamp data.
- If time data of the TIME type or date data of the CHAR type is converted to time stamp data of the TIMESTAMP type, the current date is set in the date part of the time stamp data.
- If time data of the TIME or CHAR type is converted to time data of the TIME type or time stamp data of the TIMESTAMP type and the fractional seconds precision of the data before conversion exceeds that of the data after conversion, the excess part is discarded. In such a case, the return value is SQL_SUCCESS.
- If time stamp data of the TIMESTAMP or CHAR type is converted to date data of the DATE type, only the date part of the time stamp data is converted. In such a case, the return value is SQL_SUCCESS.
- If time stamp data of the TIMESTAMP or CHAR type is converted to time data of the TIME type, only the time part of the time stamp data is converted. In such a case, the return value is SQL_SUCCESS.
- If time stamp data of the TIMESTAMP or CHAR type is converted to time data of the TIME type or time stamp data of the TIMESTAMP type and the fractional seconds precision of the data before conversion exceeds that of the data after conversion, the excess part is discarded. In such a case, the return value is SQL_SUCCESS.
- The VARCHAR type is handled in the same manner as the CHAR type.

(3) Notes about the SQL_DECIMAL type

If a fractional value less than 1, whose precision value is equal to its scaling value, is converted to the SQL_DECIMAL type, and a zero is entered to the left of the decimal point, an error will occur. This is because the zero is also counted as one digit. You can convert data successfully by specifying the data that begins at the decimal point.

Example:

Retrieving or comparing and assigning a value whose data type defined in the table is DECIMAL(3,3):

- 0.123: An error occurs.
- .123: The data is converted successfully.

(4) Notes about the SQL_C_NUMERIC type

- The maximum precision supported for HADB's SQL_C_NUMERIC type is 38.
- When data is converted from the SQL_DOUBLE type to the SQL_C_NUMERIC type, or when a numeric character string of the SQL_CHAR type or the SQL_VARCHAR type is converted to data of the SQL_C_NUMERIC type and the source data is in floating-point number format, the data is first converted to the fixed-point number format and then is converted to the SQL_C_NUMERIC type.

- The following rules apply when floating-point numbers are converted to fixed-point numbers or fixed-point numbers are converted to the SQL_C_NUMERIC type:
  - The format for floating-point numbers must comply with the format for floating-point numeric constants. A maximum of 38 digits are supported for the mantissa part and a maximum of three digits are supported for the exponent part. For details about the format for floating-point numeric constants, see the topic *Description format of literals* in the manual *HADB SQL Reference*.
  - The maximum and minimum values supported for the exponent are 38 and -38, respectively.
  - If the exponent is a positive value and the sum of the number of integer digits in the mantissa part and the value of the exponent exceeds 38, SQL_ERROR is returned because the integer is truncated when the data is converted to the fixed-point number format.
  - If the exponent is a negative value and the sum of the number of digits to the right of the decimal point in the mantissa part and the absolute value of the exponent exceeds 38, digit overflow occurs in the fractional part when data is converted to a fixed-point number. If the digits resulting in the overflow include a nonzero number, SQL_SUCCESS_WITH_INFO is returned; otherwise, SQL_SUCCESS is returned.

15.4 Information that is returned in the event of an error

This section explains the information that is returned if an error occurs in the HADB ODBC driver or in HADB (HADB server or HADB client).

How to obtain error information

In the event of an error, obtain error information by specifying the last handle used in `SQLGetDiagField` and `SQLGetDiagRec` or in `SQLGetDiagFieldW` and `SQLGetDiagRecW`.

The error information that can be obtained with `SQLGetDiagField` and `SQLGetDiagRec` or in `SQLGetDiagFieldW` and `SQLGetDiagRecW` depends on the location of the error.

■ HADB ODBC driver errors

- The `SQLSTATE` is set for each error according to the ODBC conventions. For details about `SQLSTATE` values, see the ODBC Programmer's Reference in the MSDN library and the topic *SQLSTATE Values* in the manual *HADB Messages*.
- For `NativeCode`, `0` is always returned.
- For the message text, a message beginning with `KFAA72` is set after the following character string:
  `[Hitachi Advanced Data Binder][ODBC Driver]`

■ HADB errors

- For `SQLSTATE`, the `SQLSTATE` that corresponds to the error that occurred in HADB is set. For details about `SQLSTATE`, see *List of SQLSTATE values* in the manual *HADB Messages*.
- For `NativeCode`, `SQLCODE` is set.
- For the message text, the message returned from HADB is set following the following character string:
  `[Hitachi Advanced Data Binder][ODBC Driver]`
  If detailed information cannot be obtained from an HADB client, only the above character string is set.

15.5 Limitations

This section explains the limitations when HADB is accessed via the ODBC driver.

15.5.1 ROW specification

Queries with the `ROW` specification cannot be executed. `UPDATE` and `INSERT` statements with `ROW` specified can also not be executed.

15.5.2 AUTOCOMMIT specifications

If `AUTOCOMMIT` is `ON`, a commit is executed. Information about preprocessing that has already been performed (information from execution of `SQLPrepare`) is not deleted by a commit that is issued automatically after `SQLExecute`, `SQLExecDirect`, `SQLExecDirectW`, or `SQLCloseCursor` has been executed.

Example:

1. `SQLPrepare` is executed in statement `A`.
2. `SQLPrepare` is executed in statement `B`.
3. `SQLExecute` is executed in statement `A` (commit is issued automatically).
4. `SQLExecute` is executed in statement `B` (commit is executed; preprocessing information in 3 is not deleted).

15.5.3 Notes about the maximum number SQL processing real threads

If you attempt to use multiple statement handles allocated by the same connection handle to execute SQL statements concurrently, but the number of processing real threads available does not match the maximum number of SQL processing real threads, SQL statement execution requests will result in an error (the requests will not be placed in wait status).

The maximum number of SQL processing real threads is specified in the following operands:

- `adb_sql_exe_max_rthd_num` in the server definition
- `adb_sql_exe_max_rthd_num` in the client definition

15.6 Notes

15.6.1 Effects of update operations on a retrieval using a cursor

If an update operation is performed during a retrieval using a cursor, the results of the update operation might be applied to the retrieval results, depending on the timing. To prevent the results of update operations from being applied to retrieval results, do the following:

- Close the cursor before adding or updating rows.
- Specify data and search conditions in such a manner that the rows to be added or updated are not included in the retrieval results.

15.6.2 Notes on using the HADB ODBC driver in ODBC 2.x applications

If `SQL_OV_ODBC2` is specified in the `SQL_ATTR_ODBC_VERSION` attribute of the environment handle, the HADB ODBC driver operates according to the ODBC 2.x specification in relation to the items explained in this subsection.

(1) About SQLSTATE

Pursuant to changes in Open Group and ISO specifications, a number of `SQLSTATE` values have changed in ODBC 3.x. The HADB ODBC driver returns ODBC 2.x `SQLSTATE` values mapped according to the following table. For details about `SQLSTATE`, see *List of SQLSTATE values* in the manual *HADB Messages*.

Table 15-6: Table of SQLSTATE value mapping

No.	ODBC3.x	ODBC2.x
1	07005	24000
2	07009#	S1002
3		S1093
4	22007	22008
5	22018	22005
6	HY001	S1001
7	HY003	S1003
8	HY004	S1004
9	HY007	S1010
10	HY009	S1009
11	HY010	S1010
12	HY011	S1011
13	HY024	S1009
14	HY090	S1090
15	HY091	S1091
16	HY092	S1092

No.	ODBC3.x	ODBC2.x
17	HY096	S1096
18	HY100	S1100
19	HY104	S1104
20	HY105	S1105
21	HYC00	S1C00

#

When SQLSTATE is 07009, mapping does not occur on a one-to-one basis. In this scenario, mapping takes place as shown in the following table.

Table 15-7: Mapping when SQLSTATE is 07009

No.	Function that returns SQLSTATE 07009	SQLSTATE value to which SQLSTATE 07009 is mapped when SQL_OV_ODBC2 is specified
1	SQLBindParameter	S1093
2	SQLColAttribute	S1002
3	SQLDescribeCol	07009
4	SQLDescribeParam	S1093
5	SQLFetch	S1002
6	SQLGetData	S1002
7	SQLGetDescField	07009
8	SQLGetDescRec	07009
9	SQLSetDescRec	07009
10	SQLSetDescField	07009

(2) Return value of SQL_NO_DATA

When an application calls one of the following ODBC functions that executes a DELETE or UPDATE statement that does not affect any rows, the driver returns SQL_SUCCESS. It does not return SQL_NO_DATA.

- SQLExecDirect
- SQLExecute
- SQLParamData

(3) Return value of SQLGetInfo

The value specified for SQL_ATTR_ODBC_VERSION of the environment handle affects the value returned to SQL_ALTER_TABLE in the following ways:

- When SQL_OV_ODBC3 is specified for SQL_ATTR_ODBC_VERSION
  SQL_AT_ADD_COLUMN_SINGLE is returned to SQL_ALTER_TABLE.
- When SQL_OV_ODBC2 is specified for SQL_ATTR_ODBC_VERSION
  SQL_AT_ADD_COLUMN is returned to SQL_ALTER_TABLE.

16 ODBC Functions

This chapter explains the ODBC functions provided by HADB.

16.1 List of ODBC functions

The following table lists the ODBC functions.

Table 16-1: List of ODBC functions

No.	Classification	API name	ODBC 3.0	ODBC 3.5	Remarks
1	Connecting to the data source	SQLAllocHandle	Y	Y	--
2		SQLConnect	Y	Y	
3		SQLConnectW#	N	Y	
4		SQLDriverConnect	Y	Y	
5		SQLDriverConnectW#	N	Y	
6		SQLBrowseConnect	Y	Y	
7		SQLBrowseConnectW#	N	Y	
8	Acquiring driver and data source information	SQLDataSources	DM	DM	
9		SQLDataSourcesW#	N	DM	
10		SQLDrivers	DM	DM	
11		SQLDriversW#	N	DM	
12		SQLGetInfo	Y	Y	
13		SQLGetInfoW#	N	Y	
14		SQLGetFunctions	Y	Y	
15		SQLGetTypeInfo	Y	Y	
16		SQLGetTypeInfoW#	N	Y	
17	Specifying and obtaining driver options	SQLSetConnectAttr	Y	Y	
18		SQLSetConnectAttrW#	N	Y	
19		SQLGetConnectAttr	Y	Y	
20		SQLGetConnectAttrW#	N	Y	
21		SQLSetEnvAttr	Y	Y	
22		SQLGetEnvAttr	Y	Y	
23		SQLSetStmtAttr	Y	Y	
24		SQLSetStmtAttrW#	N	Y	
25		SQLGetStmtAttr	Y	Y	
26		SQLGetStmtAttrW#	N	Y	
27	Specifying descriptor values	SQLGetDescField	Y	Y	
28		SQLGetDescFieldW#	N	Y	
29		SQLGetDescRec	Y	Y	
30		SQLGetDescRecW#	N	Y	
31		SQLSetDescField	Y	Y	

No.	Classification	API name	ODBC 3.0	ODBC 3.5	Remarks
32		SQLSetDescFieldW#	N	Y	
33		SQLSetDescRec	Y	Y	
34		SQLCopyDesc	Y	Y	
35	Creating SQL requests	SQLPrepare	Y	Y	
36		SQLPrepareW#	N	Y	
37		SQLBindParameter	Y	Y	
38		SQLGetCursorName	Y	Y	
39		SQLGetCursorNameW#	N	Y	
40		SQLSetCursorName	Y	Y	
41		SQLSetCursorNameW#	N	Y	
42		SQLDescribeParam	Y	Y	
43		SQLNumParams	Y	Y	
44	Executing SQL statements	SQLExecute	Y	Y	
45		SQLExecDirect	Y	Y	
46		SQLExecDirectW#	N	Y	
47		SQLNativeSql	Y	Y	
48		SQLNativeSqlW#	N	Y	
49		SQLParamData	Y	Y	
50		SQLPutData	Y	Y	
51	Acquiring execution results and execution result information	SQLRowCount	Y	Y	
52		SQLNumResultCols	Y	Y	
53		SQLDescribeCol	Y	Y	
54		SQLDescribeColW#	N	Y	
55		SQLColAttribute	Y	Y	
56		SQLColAttributeW#	N	Y	
57		SQLBindCol	Y	Y	
58		SQLFetch	Y	Y	
59		SQLFetchScroll	N	N	This function always results in an error because HADB does not support the functionality for cursor positioning in the reverse direction. However, this function is supported when the cursor library provided by Microsoft is used. For details, see 15.1.4 About using the ODBC cursor library.
60		SQLGetData	Y	Y	--

No.	Classification	API name	ODBC 3.0	ODBC 3.5	Remarks
61		SQLSetPos	N	N	This function always results in an error. However, this function is supported when the cursor library provided by Microsoft is used. For details, see 15.1.4 About using the ODBC cursor library.
62		SQLBulkOperations	N	N	This function always results in an error because HADB does not support bookmarks.
63		SQLMoreResults	Y	Y	This function always returns SQL_NO_DATA.
64		SQLGetDiagField	Y	Y	--
65		SQLGetDiagFieldW#	N	Y	
66		SQLGetDiagRec	Y	Y	
67		SQLGetDiagRecW#	N	Y	
68	Acquiring system information for the data source	SQLColumnPrivileges	Y	Y	The number of rows in the retrieval result set is always zero.
69		SQLColumnPrivilegesW#	N	Y	
70		SQLColumns	Y	Y	--
71		SQLColumnsW#	N	Y	
72		SQLForeignKeys	Y	Y	
73		SQLForeignKeysW#	N	Y	
74		SQLPrimaryKeys	Y	Y	
75		SQLPrimaryKeysW#	N	Y	
76		SQLProcedureColumns	Y	Y	The number of rows in the retrieval result set is always zero.
77		SQLProcedureColumnsW#	N	Y	
78		SQLProcedures	Y	Y	
79		SQLProceduresW#	N	Y	
80		SQLSpecialColumns	Y	Y	
81		SQLSpecialColumnsW#	N	Y	
82		SQLStatistics	Y	Y	--
83		SQLStatisticsW#	N	Y	
84		SQLTablePrivileges	Y	Y	
85		SQLTablePrivilegesW#	N	Y	
86		SQLTables	Y	Y	
87		SQLTablesW#	N	Y	
88	Terminating execution of SQL statements	SQLFreeStmt	Y	Y	
89		SQLCloseCursor	Y	Y	

No.	Classification	API name	ODBC 3.0	ODBC 3.5	Remarks
90		SQLCancel	Y	Y	
91		SQLEndTran	Y	Y	
92	Disconnecting from the data source	SQLDisconnect	Y	Y	
93		SQLFreeHandle	Y	Y	

Legend:

Y: Supported.

N: Not supported.

DM: Provided by the driver manager.

--: None

#

The SQL*xxx*W functions are used to perform conversion between the MS-Unicode character string data and the character encodings supported by HADB. For details about the processing of conversion between character encodings, see 15.1.3 About conversion of character encoding.

16.2 Notes about SQLxxx and SQLxxxW functions

This section provides notes about the following functions that are described in this chapter:

- SQL*xxx* functions used in ODBC 3.0
- SQL*xxx*W functions used in ODBC 3.5

Arguments that use an SQL*xxx* function to define the SQLCHAR type in ODBC 3.0 use an SQL*xxx*W function to define the SQLWCHAR type in ODBC 3.5.

The units for parameters that specify lengths are as follows:

- SQL*xxx* functions in ODBC 3.0: Lengths are in bytes
- SQL*xxx*W functions in ODBC 3.5
  - For a parameter of the SQLPOINTER type: Lengths are in bytes
  - For a parameter of the SQLWCHAR type: Lengths are in characters

For those functions whose names differ between ODBC 3.0 and ODBC 3.5, the SQL*xxx* function used in ODBC 3.0 is explained first, and then the SQL*xxx*W function used in ODBC 3.5 is explained.

16.3 Connecting to the data source

This section explains the ODBC functions that are used for connecting to the data source.

16.3.1 SQLAllocHandle

(1) Function

This ODBC function allocates environment, connection, statement, and descriptor handles.

(2) Format

```
SQLRETURN SQLAllocHandle
(
  SQLSMALLINT     HandleType,         /* In */
  SQLHANDLE       InputHandle,        /* In */
  SQLHANDLE     * OutputHandlePtr     /* In/Out */
)
```

(3) Arguments

`HandleType`

Specifies one of the following handle types:

- `SQL_HANDLE_ENV`: Environment handle
- `SQL_HANDLE_DBC`: Connection handle
- `SQL_HANDLE_STMT`: Statement handle
- `SQL_HANDLE_DESC`: Descriptor handle

`InputHandle`

Specifies one of the following input handles, depending on the value of the `HandleType` parameter:

HandleType	Value that is set for InputHandle
`SQL_HANDLE_ENV`	`SQL_NULL_HANDLE`
`SQL_HANDLE_DBC`	Environment handle
`SQL_HANDLE_STMT`	Connection handle
`SQL_HANDLE_DESC`	Connection handle

`OutputHandlePtr`

Specifies a pointer to the buffer used to return the new handle that was allocated.

If the return value is `SQL_SUCCESS`, the value of the handle has been set. If it is `SQL_ERROR`, the following value is set:

HandleType	Value specified for OutputHandlePtr
`SQL_HANDLE_ENV`	`NULL`
`SQL_HANDLE_DBC`	`SQL_NULL_HDBC`

HandleType	Value specified for OutputHandlePtr
SQL_HANDLE_STMT	SQL_NULL_HSTMT
SQL_HANDLE_DESC	SQL_NULL_HDESC

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08003	Connection does not exist	Connection establishment has not been completed.	Y
HY000	General error	--	N
HY001	Memory allocation error		Y
HY009	Invalid use of null pointer		Y
HY010	Function sequence error	The ODBC version has not been set prior to the acquisition of a connection handle.	Y
HY013	Memory management error	HandleType is SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC, but the memory object cannot be accessed (memory shortage).	N
HY014	Limit on the number of handles exceeded	--	N
HY092	Invalid attribute or option identifier	An invalid value was specified for HandleType.	Y
HYC00	Optional feature not implemented	--	N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.3.2 SQLConnect, SQLConnectW

(1) Function

This ODBC function establishes a connection between the HADB ODBC driver and the data source (HADB server).

Note that you must have the CONNECT privilege to execute SQLConnect and SQLConnectW.

(2) Format

- For SQLConnect

```
SQLRETURN SQLConnect
(
  SQLHDBC           ConnectionHandle,  /* In */
  SQLCHAR         * ServerName,        /* In */
  SQLSMALLINT       NameLength1,       /* In */
  SQLCHAR         * UserName,          /* In */
  SQLSMALLINT       NameLength2,       /* In */
  SQLCHAR         * Authentication,    /* In */
  SQLSMALLINT       NameLength3        /* In */
)
```

- For SQLConnectW

```
SQLRETURN SQLConnectW
(
  SQLHDBC           ConnectionHandle,  /* In */
  SQLWCHAR        * ServerName,        /* In */
  SQLSMALLINT       NameLength1,       /* In */
  SQLWCHAR        * UserName,          /* In */
  SQLSMALLINT       NameLength2,       /* In */
  SQLWCHAR        * Authentication,    /* In */
  SQLSMALLINT       NameLength3        /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies the connection handle.

ServerName

Specifies the name of the data source.

NameLength1

Specifies the length# of the data source name.

If the data source name ends with the null terminating character, specify SQL_NTS.

If zero or a negative value is specified, an error results.

UserName

Specifies the user ID (authorization identifier) for connecting to the HADB server.

NameLength2

Specifies the length# of the user ID.

If the user ID ends with the null terminating character, specify SQL_NTS.

If zero or a negative value is specified, an error results.

Authentication

Specifies the password for the authorization identifier being used to connect to the HADB server. If NULL is specified, an error results.

Because of ODBC implementation conventions, we recommend that you do not use any of the following 13 characters in passwords: [,], {, }, (,), ,, ;, ?, *, =, !, and @. For details about the passwords supported by HADB, see *Password specification rules* in the *HADB Setup and Operation Guide*.

NameLength3

Specifies the length# of the password.

If the password ends with the null terminating character, specify SQL_NTS.

If a negative value or a value 256 or greater is specified, an error results.

Note also that if a character string whose length is 256 characters or more is specified in Authentication and SQL_NTS is specified in this argument, an error results.

#

The length must be in bytes for SQLConnect and in characters for SQLConnectW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
08001	Client unable to establish a connection		N
08002	Connection name in use		Y
08004	Server rejected the connection		N
08S01	Communication link failure		N
28000	Invalid authorization specification		Y
5C002	Character encoding conversion error	A character encoding that cannot be converted was detected.	Y
5C052	Version mismatch error	The versions of the ODBC driver and HADB client are different.	Y
5D001	HADB-specific error	An error occurred on HADB, but a specific SQLSTATE or error message cannot be obtained.	Y
HY000	General error	--	N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length		Y
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM002	No data source is found and Default driver is not specified		N

SQLSTATE	Description	Remarks	Returned
IM003	Specified driver could not be loaded		N
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed		N
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed		N
IM006	Driver's SQLSetConnectAttr or Driver's SQLSetConnectAttrW failed		N
IM009	Unable to load translation DLL		N
IM010	Data source name too long		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.3.3 SQLDriverConnect, SQLDriverConnectW

(1) Function

This ODBC function establishes a connection with the data source (HADB server) by using one of the following connection attributes:

- The ODBC function establishes a connection by using a connection character string containing the data source name, at least one authorization identifier, at least one password, and other information required for establishing a connection with the data source.
- The ODBC function establishes a connection without using a partial connection character string or additional information. In this case, the driver manager and the HADB ODBC driver will request connection information from the application program.
- The ODBC function establishes a connection with a data source that is not defined in the system definition. If an application provides a partial connection character string, the HADB ODBC driver will request connection information from the user.
- The ODBC function establishes a connection with the data source by using the connection character string created from the information in a .dsn file.

When a connection is established, SQLDriverConnect or SQLDriverConnectW returns a complete connection character string.

Note that you must have the CONNECT privilege to execute SQLDriverConnect and SQLDriverConnectW.

(2) Format

- For SQLDriverConnect

```
SQLRETURN SQLDriverConnect
(
  SQLHDBC         ConnectionHandle,        /* In */
```

```
  SQLHWND         WindowHandle,            /* In */
  SQLCHAR       * InConnectionString,      /* In */
  SQLSMALLINT     StringLength1,           /* In */
  SQLCHAR       * OutConnectionString,     /* Out */
  SQLSMALLINT     BufferLength,            /* In */
  SQLSMALLINT   * StringLength2Ptr,        /* Out */
  SQLUSMALLINT    DriverCompletion         /* In */
)
```

- For SQLDriverConnectW

```
SQLRETURN SQLDriverConnectW
(
  SQLHDBC         ConnectionHandle,        /* In */
  SQLHWND         WindowHandle,            /* In */
  SQLWCHAR      * InConnectionString,      /* In */
  SQLSMALLINT     StringLength1,           /* In */
  SQLWCHAR      * OutConnectionString,     /* Out */
  SQLSMALLINT     BufferLength,            /* In */
  SQLSMALLINT   * StringLength2Ptr,        /* Out */
  SQLUSMALLINT    DriverCompletion         /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

WindowHandle

Specifies the handle of the parent window.

If windows cannot be applied or dialog boxes will not be displayed, specify a null pointer.

InConnectionString

Specifies a connection character string.

The connection attributes permitted in the connection character string are as follows:

Connection attribute	Description
DSN	Data source name
DRIVER	ODBC driver name: Hitachi Advanced Data Binder ODBC Driver
UID	Authorization identifier
PWD	Password
CLTPATH	Absolute path of a client definition file

StringLength1

Specifies the length# of the connection character string specified for InConnectionString.

If the connection character string specified for InConnectionString ends with the null terminating character, specify SQL_NTS.

If zero or a negative value is specified, an error results.

OutConnectionString

Specifies a pointer to the buffer that stores the complete connection character string.

If the connection to the HADB server is successful, the function returns the complete connection character string.

BufferLength

Specifies the length# of the buffer that stores OutConnectionString.

This length includes the null terminating character. SQL_NTS cannot be specified.

StringLength2Ptr

Specifies a pointer to the buffer that stores the valid length# of the complete connection character string. This length does not include the null terminating character.

Important

If the length# of the connection character string stored here is greater than the value of BufferLength without the length# of the null terminating character, the character string stored in OutConnectionString is truncated to the length# equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end.

DriverCompletion

Specifies a flag indicating whether the driver manager or the HADB ODBC driver requires more connection information. You can specify the following flags:

- SQL_DRIVER_PROMPT
- SQL_DRIVER_COMPLETE
- SQL_DRIVER_COMPLETE_REQUIRED
- SQL_DRIVER_NOPROMPT

#

The length must be in bytes for SQLDriverConnect and in characters for SQLDriverConnectW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	The complete connection character string could not be stored because the *OutConnectionString buffer was too small (the information was truncated). The length of the untruncated complete connection character string is stored in the *StringLength2Ptr buffer. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S00	Invalid connection string attribute	The connection character string (InConnectionString) contains an invalid attribute keyword. The function returns SQL_SUCCESS_WITH_INFO.	Y

SQLSTATE	Description	Remarks	Returned
01S02	Option value changed	The option value was replaced with a similar value because the HADB ODBC driver does not support the value specified for ValuePtr in SQLSetConnectAttr or SQLSetConnectAttrW. The function returns SQL_SUCCESS_WITH_INFO.	N
01S08	Error saving file DSN	The connection character string for *InConnectionString contains the FILEDSN keyword, but the .dsn file was not saved. The function returns SQL_SUCCESS_WITH_INFO.	N
01S09	Invalid keyword	--	N
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
08001	Client unable to establish a connection	• The HADB ODBC driver cannot connect to the data source. • SQL_DRIVER_NOPROMPT is specified in DriverCompletion, but the attribute character string required for the connection was not specified in InConnectionString.	Y
08002	Connection name in use	--	N
08004	Server rejected the connection	The data source rejected the connection for the reason defined during implementation.	N
08S01	Communication link failure	--	N
28000	Invalid authorization specification	The authorization identifier or password specified in the connection character string violates the limitations on data source definitions.	Y
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C052	Version mismatch error	The versions of the ODBC driver and HADB client are different.	Y
5D001	HADB-specific error	An error occurred on HADB, but a specific SQLSTATE or error message cannot be obtained.	Y
HY000	General error	--	N
HY001	Memory allocation error		N
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	• The length of a connection attribute value specified in InConnectionString is invalid. • An invalid value (SQL_NTS) was specified in BufferLength.	Y
HY092	Invalid attribute or option identifier	--	N

SQLSTATE	Description	Remarks	Returned
HY110	Option identifier whose DriverCompletion is invalid		N
HYC00	Optional feature not implemented	The HADB ODBC driver does not support the ODBC processing requested by the application.	N
HYT00	Timeout expired	A login timeout occurred before a connection with the data source was completed. The login timeout value can be specified by using SQL_ATTR_LOGIN_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
IM001	Driver does not support this function	--	N
IM002	No data source is found and Default driver is not specified		N
IM003	Specified driver could not be loaded		N
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed		N
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed		N
IM006	Driver's SQLSetConnectAttr or Driver's SQLSetConnectAttrW failed		N
IM007	Data source or driver is not specified and Dialog prohibited	Neither a data source name nor a driver name is specified in the connection character string, but SQL_DRIVER_NOPROMPT is specified in DriverCompletion.	N
IM008	Dialog failed	The driver's attempt to display the login dialog box failed. A null pointer is specified in WindowHandle, but SQL_DRIVER_NOPROMPT is not specified in DriverCompletion.	N
IM009	Unable to load translation DLL	--	N
IM010	Data source name too long		N
IM011	Driver name too long		N
IM012	DRIVER keyword syntax error		N
IM014	Invalid name of file DSN		N
IM015	Corrupt file data source		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- The following table lists and describes the connection attributes that are specified in the connection character string.

Table 16-2: Connection attributes that are specified in the connection character string

No.	Connection attribute	Connection attribute value	Whether specification is required	
			DSN connection	DRIVER connection
1	DSN	Specify a data source name.	Y	--
2	DRIVER	Specify an ODBC driver name. The ODBC driver name is Hitachi Advanced Data Binder ODBC Driver.	--	Y
3	UID	Specify an authorization identifier.	Y	Y
4	PWD	Specify a password. Specify a character string consisting of 255 bytes or less.	Y	Y
5	CLTPATH	Specify the absolute path of the client definition file. Express the absolute path of the client definition file as a maximum of 255 bytes of character string. If this attribute is omitted, the file under the folder specified in the ADBCLTDIR environment definition is used.	--	O

Legend:

Y: Mandatory connection attribute

O: Optional connection attribute

--: Connection attribute whose specification is not needed

- The following shows examples of connection character string specifications:
  - "DSN=*XXXXX*;UID=*YYYYY*;PWD=*ZZZZZ*"
  - "DRIVER=Hitachi Advanced Data BinderODBC Driver;CLTPATH=*XXXXX*;UID=*YYYYY*;PWD=*ZZZZZ*"
- If the DSN and DRIVER connection attributes are both specified in the connection character string, the first connection attribute specified takes effect.
- If the same connection attribute is specified more than once in the connection character string, the value of the last connection attribute specified takes effect.
- The connection attributes can be specified in any order in the connection character string. However, if neither DSN=*XXXXX* nor DRIVER=Hitachi Advanced Data Binder ODBC Driver is specified in the first request, the ODBC driver manager detects an error. This does not apply to the FILEDSN specification.
- The connection attributes are not case-sensitive.
- The connection attribute values are case sensitive.
- A semicolon (;) is treated as a delimiter. For this reason, the semicolon must not be included in a password character string in the connection attributes. Because of ODBC implementation conventions, we recommend that you do not

use any of the following 12 characters in passwords: [,], {, }, (,), ,, ?, *, =, !, and @. For details about the passwords supported by HADB, see *Password specification rules* in the *HADB Setup and Operation Guide*.

16.3.4 SQLBrowseConnect, SQLBrowseConnectW

(1) Function

This ODBC function supports a method of referencing attributes and attribute values one at time that are required to establish a connection with the data source (HADB server). Each call to SQLBrowseConnect or SQLBrowseConnectW returns successive levels of attributes and attribute values. When all levels of attributes are specified, a connection to the data source is completed and a complete connection character string is returned by SQLBrowseConnect or SQLBrowseConnectW.

Note that you must have the CONNECT privilege to execute SQLBrowseConnect and SQLBrowseConnectW.

(2) Format

- For SQLBrowseConnect

```
SQLRETURN SQLBrowseConnect
(
  SQLHDBC         ConnectionHandle,        /* In */
  SQLCHAR       * InConnectionString,      /* In */
  SQLSMALLINT     StringLength1,           /* In */
  SQLCHAR       * OutConnectionString,     /* Out */
  SQLSMALLINT     BufferLength,            /* In */
  SQLSMALLINT   * StringLength2Ptr         /* Out */
)
```

- For SQLBrowseConnectW

```
SQLRETURN SQLBrowseConnectW
(
  SQLHDBC         ConnectionHandle,        /* In */
  SQLWCHAR      * InConnectionString,      /* In */
  SQLSMALLINT     StringLength1,           /* In */
  SQLWCHAR      * OutConnectionString,     /* Out */
  SQLSMALLINT     BufferLength,            /* In */
  SQLSMALLINT   * StringLength2Ptr         /* Out */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

InConnectionString

Specifies a browse request connection character string.

The following table lists and describes the connection attributes that can be specified in the browse request connection character string.

Connection attribute	Description
DSN	Data source name

Connection attribute	Description
DRIVER	ODBC driver name: Hitachi Advanced Data Binder ODBC Driver
UID	Authorization identifier
PWD	Password
CLTPATH	Absolute path of a client definition file

StringLength1

Specifies the length# of the browse request connection character string specified for InConnectionString.

If the browse request connection character string specified for InConnectionString ends with the null terminating character, specify SQL_NTS.

If zero or a negative value is specified, an error results.

OutConnectionString

Specifies a pointer to the buffer that stores the browse result connection character string.

If the connection to the HADB server is successful, the function returns the complete connection character string.

If the function returns SQL_NEED_DATA, it returns the connection attributes that were lacking when connection with the HADB server was attempted.

BufferLength

Specifies the length# of the buffer that stores OutConnectionString.

This length includes the null terminating character. SQL_NTS cannot be specified.

StringLength2Ptr

Specifies a pointer to the buffer that stores the valid length# of the browse result connection character string. This length does not include the null terminating character.

This is the valid length# of the character string that is returned to OutConnectionString.

Important

If the length# of the connection character string stored here is greater than the value of BufferLength without the length# of the null terminating character, the character string stored in OutConnectionString is truncated to the length# equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end.

#

The length must be in bytes for SQLBrowseConnect and in characters for SQLBrowseConnectW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N

SQLSTATE	Description	Remarks	Returned
01004	Character string data was right-truncated	The browse result connection character string could not be stored because the *OutConnectionString buffer was too small (the information was truncated). The length of the untruncated browse result connection character string is stored in the *StringLength2Ptr buffer. The function returns SQL_NEED_DATA.	Y
01S00	Invalid connection string attribute	The browse request connection character string (InConnectionString) contains an invalid attribute keyword. The function returns SQL_NEED_DATA. The attribute keyword specified in the browse request connection character string (InConnectionString) does not apply to the current connection level. The function returns SQL_NEED_DATA.	Y
01S02	Option value changed	The option value was replaced with a similar value because the HADB ODBC driver does not support the value specified for ValuePtr in SQLSetConnectAttr or SQLSetConnectAttrW. The function returns SQL_SUCCESS_WITH_INFO.	N
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
08001	Client unable to establish a connection	The HADB ODBC driver cannot connect to the data source.	N
08002	Connection name in use	--	N
08S01	Communication link failure		N
28000	Invalid authorization specification	The authorization identifier or password specified in the browse request connection character string violates the data source limitations.	Y
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C052	Version mismatch error	The versions of the ODBC driver and HADB client are different.	Y
5D001	HADB-specific error	An error occurred on HADB, but a specific SQLSTATE or error message cannot be obtained.	Y
HY000	General error	--	N
HY001	Memory allocation error		N
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	• The length of a connection attribute value specified in InConnectionString is invalid.	Y

SQLSTATE	Description	Remarks	Returned
		• An invalid value (SQL_NTS) was specified in BufferLength.	
HYT00	Timeout expired	A login timeout occurred before connection with the data source was completed. The login timeout value can be specified by using SQL_ATTR_LOGIN_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
IM001	Driver does not support this function	--	N
IM002	No data source is found and Default driver is not specified		N
IM003	Specified driver could not be loaded		N
IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed		N
IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed		N
IM006	Driver's SQLSetConnectAttr or SQLSetConnectAttrW failed		N
IM009	Unable to load translation DLL		N
IM010	Data source name too long		N
IM011	Driver name too long		N
IM012	DRIVER keyword syntax error		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- The following table lists and describes the connection attributes that are specified in the browse request connection character string.

Table 16-3: Connection attributes that are specified in the browse request connection character string

No.	Connection attribute	Connection attribute value	Whether specification is required	
			DSN connection	DRIVER connection
1	DSN	Specify a data source name.	Y	--

No.	Connection attribute	Connection attribute value	Whether specification is required	
			DSN connection	DRIVER connection
2	DRIVER	Specify an ODBC driver name. The ODBC driver name is Hitachi Advanced Data Binder ODBC Driver.	--	Y
3	UID	Specify an authorization identifier.	Y	Y
4	PWD	Specify a password. Specify a character string consisting of 255 bytes or less.	Y	Y
5	CLTPATH	Specify the absolute path of the client definition file. Express the absolute path of the client property file as a maximum of 255 bytes of character string. If this attribute is omitted, the file under the folder specified in the ADBCLTDIR environment definition is used.	--	O

Legend:

Y: Mandatory connection attribute

O: Optional connection attribute

--: Connection attribute whose specification is not needed

- The following shows an example specification of the browse request connection character string:
  - "DSN=*XXXXX*;UID=*YYYYY*;PWD=*ZZZZZ*"
  - "DRIVER=Hitachi Advanced Data Binder ODBC Driver;CLTPATH=*XXXXX*;UID=*YYYYY*;PWD=*ZZZZZ*"
- If the DSN and DRIVER connection attributes are both specified in the browse request connection character string, the last connection attribute specified takes effect.
- If the same connection attribute is specified more than once in the browse request connection character string, the value of the first connection attribute specified takes effect.
- The connection attributes can be specified in any order in the browse request connection character string. However, if neither DSN=*XXXXX* nor DRIVER=Hitachi Advanced Data Binder ODBC Driver is specified in the first request, the ODBC driver manager detects an error.
- The connection attributes are not case-sensitive.
- The connection attribute values are case sensitive.
- A semicolon (;) is treated as a delimiter. For this reason, the semicolon must not be used in a password character string in the connection attributes. Due to ODBC implementation conventions, we recommend that you do not use any of the following 12 characters in passwords: [,], {, }, (,), ,, ?, *, =, !, and @. For details about the passwords supported by HADB, see *Password specification rules* in the *HADB Setup and Operation Guide*.
- If SQL_NEED_DATA is returned because an invalid parameter value was specified in this function's argument, then re-execution of the function results in SQL_ERROR due to an error such as an invalid password character string, and then SQLFreeHandle is executed as is, the ODBC driver manager might return SQL_ERROR (SQLSTATE: HY010) and the handle might not be freed. In such a case, re-execute the function with the correct parameter specified, and then execute SQLFreeHandle. Alternatively, terminate the application forcibly.

16.4 Acquiring driver and data source information

This section explains the ODBC functions that are used to acquire driver and data source information.

16.4.1 SQLDataSources, SQLDataSourcesW

(1) Function

This ODBC function returns the following information:

- Data source name
- Description of the driver associated with the data source

This function is implemented by the driver manager. The HADB ODBC driver provides `SQLDataSources` or `SQLDataSourcesW` as a function that always returns `SQL_SUCCESS`.

(2) Format

- For `SQLDataSources`

```
SQLRETURN SQLDataSources
(
  SQLHENV           EnvironmentHandle,  /* In */
  SQLUSMALLINT      Direction,          /* In */
  SQLCHAR         * ServerName,         /* Out */
  SQLSMALLINT       BufferLength1,      /* In */
  SQLSMALLINT     * NameLength1Ptr,     /* Out */
  SQLCHAR         * Description,        /* Out */
  SQLSMALLINT       BufferLength2,      /* In */
  SQLSMALLINT     * NameLength2Ptr      /* Out */
)
```

- For `SQLDataSourcesW`

```
SQLRETURN SQLDataSourcesW
(
  SQLHENV           EnvironmentHandle,  /* In */
  SQLUSMALLINT      Direction,          /* In */
  SQLWCHAR        * ServerName,         /* Out */
  SQLSMALLINT       BufferLength1,      /* In */
  SQLSMALLINT     * NameLength1Ptr,     /* Out */
  SQLWCHAR        * Description,        /* Out */
  SQLSMALLINT       BufferLength2,      /* In */
  SQLSMALLINT     * NameLength2Ptr      /* Out */
)
```

(3) Arguments

`EnvironmentHandle`

Specifies an environment handle.

`Direction`

Specifies how the driver manager is to read the data source information. The permitted values are as follows:

- SQL_FETCH_FIRST: Fetch the data source name (hereafter called DSN) at the top of the list (both user and system DSNs).
- SQL_FETCH_FIRST_USER: Fetch the first user DSN.
- SQL_FETCH_FIRST_SYSTEM: Fetch the first system DSN.
- SQL_FETCH_NEXT: Fetch the next DSN in the list. The target is the same type as the DSN that was fetched by FIRST (both user and system DSN, or only user DSN, or only system DSN).

ServerName

Specifies a pointer to the buffer in which the data source name is to be returned.

BufferLength1

Specifies the length# of the *ServerName buffer. The maximum value is SQL_MAX_DSN_LENGTH plus the null terminating character. If a larger value is specified, this maximum value is assumed. This length includes the null terminating character.

NameLength1Ptr

Specifies a pointer to the buffer that stores the total valid length# of *ServerName. This length does not include the null terminating character.

Important

If the length# of the data source name stored here is greater than the value of BufferLength1 without the length# of the null terminating character, the character string stored in *ServerName is truncated to the length# equivalent to BufferLength1 without the null terminating character, and then the null terminating character is added at the end.

Description

Specifies a pointer to the buffer in which the description of the driver associated with the data source (such as an HADB server) is to be returned.

BufferLength2

Specifies the length# of the *Description buffer.

This length includes the null terminating character.

NameLength2Ptr

Specifies a pointer to the buffer that stores the total valid length# to be returned to *Description. This length does not include the null terminating character.

Important

If the length# of the driver description stored here is greater than the value of BufferLength2 without the length# of the null terminating character, the character string stored in *Description is truncated to the length# equivalent to BufferLength2 without the null terminating character, and then the null terminating character is added at the end.

#

The length must be in bytes for SQLDataSources and in characters for SQLDataSourcesW.

(4) Return value

This ODBC function returns SQL_SUCCESS.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
HY000	General error		N
HY001	Memory allocation error		N
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY103	Invalid retrieval code		N

Legend:

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.4.2 SQLDrivers, SQLDriversW

(1) Function

This ODBC function returns a listing of driver descriptions and driver attribute keywords.

This function is implemented by the driver manager. The HADB ODBC driver provides SQLDrivers or SQLDriversW as a function that always returns SQL_SUCCESS.

(2) Format

- For SQLDrivers

```
SQLRETURN SQLDrivers
(
  SQLHENV             EnvironmentHandle,     /* In */
  SQLUSMALLINT        Direction,             /* In */
  SQLCHAR           * DriverDescription,     /* Out */
  SQLSMALLINT         BufferLength1,         /* In */
  SQLSMALLINT       * DescriptionLengthPtr,  /* Out */
  SQLCHAR           * DriverAttributes,      /* Out */
  SQLSMALLINT         BufferLength2,         /* In */
  SQLSMALLINT       * AttributesLengthPtr    /* Out */
)
```

- For SQLDriversW

```
SQLRETURN SQLDriversW
(
  SQLHENV             EnvironmentHandle,     /* In */
  SQLUSMALLINT        Direction,             /* In */
  SQLWCHAR          * DriverDescription,     /* Out */
  SQLSMALLINT         BufferLength1,         /* In */
  SQLSMALLINT       * DescriptionLengthPtr,  /* Out */
```

```
  SQLWCHAR          * DriverAttributes,      /* Out */
  SQLSMALLINT         BufferLength2,         /* In */
  SQLSMALLINT       * AttributesLengthPtr    /* Out */
)
```

(3) Arguments

`EnvironmentHandle`

Specifies an environment handle.

`Direction`

Specifies how the driver manager is to read the data source information. The permitted values are as follows:

- `SQL_FETCH_FIRST`: Fetch the driver description from the top of the list (both user and system DSNs).
- `SQL_FETCH_NEXT`: Fetch the next driver description from the list.

`DriverDescription`

Specifies a pointer to the buffer in which the driver description is to be returned.

`BufferLength1`

Specifies the length# of the buffer in which the driver description is to be returned.

`DescriptionLengthPtr`

Specifies a pointer to the buffer that stores the total valid length# of `*DriverDescription`. This length does not include the null terminating character.

`DriverAttributes`

Specifies a pointer to the buffer that stores the driver attribute values.

`BufferLength2`

Specifies the length# of the buffer in which the driver's attribute values are to be returned.

`AttributesLengthPtr`

Specifies a pointer to the buffer that stores the total valid length# of `*AttributesLengthPtr`. This length does not include the null terminating character.

#

The length must be in bytes for `SQLDrivers` and in characters for `SQLDriversW`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
HY000	General error		N
HY001	Memory allocation error		N

SQLSTATE	Description	Remarks	Returned
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY103	Invalid retrieval code		N

Legend:

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.4.3 SQLGetInfo, SQLGetInfoW

(1) Function

This ODBC function returns general information about the driver and data source associated with a connection.

(2) Format

- For SQLGetInfo

```
SQLRETURN SQLGetInfo
(
  SQLHDBC           ConnectionHandle,   /* In */
  SQLUSMALLINT      InfoType,           /* In */
  SQLPOINTER        InfoValuePtr,       /* In/Out */
  SQLSMALLINT       BufferLength,       /* In */
  SQLSMALLINT     * StringLengthPtr     /* Out */
)
```

- For SQLGetInfoW

```
SQLRETURN SQLGetInfoW
(
  SQLHDBC           ConnectionHandle,   /* In */
  SQLUSMALLINT      InfoType,           /* In */
  SQLPOINTER        InfoValuePtr,       /* In/Out */
  SQLSMALLINT       BufferLength,       /* In */
  SQLSMALLINT     * StringLengthPtr     /* Out */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

InfoType

Specifies a type of information.

For details about the types of information that can be specified, see 16.13 Information types that can be specified for InfoType in SQLGetInfo and SQLGetInfoW.

InfoValuePtr

Specifies a pointer to the buffer in which the information is to be returned. The function returns the information for the requested InfoType. If a specifiable information type is specified for InfoType but that information type is not supported, the function might return the null character string or zero.

BufferLength

Specifies the length (in bytes) of the *InfoValuePtr buffer. This length includes the null terminating character. SQL_NTS cannot be specified.

If the *InfoValuePtr value is not a character string or InfoValuePtr is a null pointer, this argument is ignored.

If the value of InfoValuPtr is not a character string, the HADB ODBC driver assumes that the size of *InfoValuePtr is either SQLUSMALLINT or SQLUINTEGER based on InfoType. If the buffer is too small, operation is not guaranteed.

StringLengthPtr

Specifies a pointer to the buffer that returns the total valid length (in bytes) to be returned to *InfoValuePtr. This length does not include the null terminating character.

Important

For character string data, if this length (in bytes) is greater than the value of BufferLength without the length of the null terminating character, the character string stored in *InfoValuePtr is truncated to the length (in bytes) equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end. For any other data type, the value of BufferLength is ignored, in which case the driver assumes that the size of *InfoValuePtr is either SQLUSMALLINT or SQLUINTEGER based on InfoType.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	Not all of the requested information could be stored because the *InfoValuePtr buffer was too small (the information was truncated). The length of the untruncated requested information is stored in *StringLengthPtr. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
08003	Connection does not exist	--	N

SQLSTATE	Description	Remarks	Returned
08S01	Communication link failure		N
HY000	General error		N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer		N
HY013	Memory management error		N
HY024	Invalid attribute value		N
HY090	Invalid string or buffer length	An invalid value (SQL_NTS) was specified in BufferLength.	Y
HY096	Out-of-range information type	The value specified in InfoType is not supported.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

If the information type specified for InfoType is not SQL_DRIVER_ODBC_VER, the connection must have already been established.

16.4.4 SQLGetFunctions

(1) Function

This ODBC function returns information about whether the driver supports a specific ODBC function or returns a list of support information for the individual ODBC functions.

(2) Format

```
SQLRETURN SQLGetFunctions
(
  SQLHDBC            ConnectionHandle,   /* In */
  SQLUSMALLINT       FunctionId,         /* In */
  SQLUSMALLINT     * SupportedPtr        /* Out */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`FunctionId`

Specifies `SQL_API_ODBC3_ALL_FUNCTIONS` to obtain a list of support information for all functions.

To obtain information on whether a specific function is supported, specify the `#define` value of the target ODBC function.

`SupportedPtr`

Specifies a pointer to the buffer in which the support information for the specified function is to be returned.

If the `#define` value of an ODBC function is specified for `FunctionId`, the function returns `SQL_TRUE` (supported) or `SQL_FALSE` (not supported).

You can also specify `NULL` for `SupportedPtr`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY095	Function type out of range		Y
HYT01	Connection timeout expired		N

Legend:

Y: This `SQLSTATE` might be returned by the HADB ODBC driver.

N: This `SQLSTATE` is not returned by the HADB ODBC driver.

--: None

16.4.5 SQLGetTypeInfo, SQLGetTypeInfoW

(1) Function

This ODBC function returns information about the data types supported by the HADB ODBC driver as a result set for the specified SQL statement.

(2) Format

- For SQLGetTypeInfo

```
SQLRETURN SQLGetTypeInfo
(
  SQLHSTMT       StatementHandle,      /* In */
  SQLSMALLINT    DataType              /* In */
)
```

- For SQLGetTypeInfoW

```
SQLRETURN SQLGetTypeInfoW
(
  SQLHSTMT       StatementHandle,      /* In */
  SQLSMALLINT    DataType              /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

DataType

Specifies an SQL data type identifier. Specify as explained below, depending on the data to be acquired:

- To acquire information about a specific data type

  Specify the applicable ODBC SQL data type identifier provided in 15.3.1 Correspondence between ODBC's SQL data types and HADB's data types.

- To acquire information about all supported data types

  Specify SQL_ALL_TYPES.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

When SQLGetTypeInfo or SQLGetTypeInfoW is executed, a result set is created. The following table shows the format of the result set that is returned.

Table 16-4: Format of the ResultSet object that is returned

Column No.	Type	Column name	Description
1	Varchar	TYPE_NAME	HADB data type name
2	Smallint	DATA_TYPE	ODBC SQL data type
3	Integer	COLUMN_SIZE	Column size
4	Varchar	LITERAL_PREFIX	Prefix used to quote a literal
5	Varchar	LITERAL_SUFFIX	Suffix used to quote a literal
6	Varchar	CREATE_PARAMS	Parameters used to create types
7	Smallint	NULLABLE	Returns a value indicating whether the null value can be used.

Column No.	Type	Column name	Description
			The function always returns SQL_NULLABLE_UNKNOWN.
8	Smallint	CASE_SENSITIVE	Returns a value indicating whether the data type is case-sensitive. • SQL_TRUE: Character string data type • SQL_FALSE: Other data type
9	Smallint	SEARCHABLE	Returns a value indicating how the data type is used in WHERE. The function always returns SQL_SEARCHABLE.
10	Smallint	UNSIGNED_ATTRIBUTE	Returns a value indicating whether the data type has the unsigned attribute. • SQL_FALSE: Numeric data type (because the data type is signed) • Null value: Other data type
11	Smallint	FIXED_PREC_SCALE	Returns a value indicating whether the data type can be a currency value. The function always returns SQL_FALSE.
12	Smallint	AUTO_UNIQUE_VALUE	Returns a value indicating whether the data type can be used for autoincrementing values. The function always returns a null value.
13	Varchar	LOCAL_TYPE_NAME	Localized version of the type name. The function returns the same name as the type name.
14	Smallint	MINIMUM_SCALE	Minimum scale supported. If scale is not applicable, the function returns a null value.
15	Smallint	MAXIMUM_SCALE	Maximum scale supported. If scale is not applicable, the function returns a null value.
16	Smallint	SQL_DATA_TYPE	ODBC SQL data type that is set in the SQL_DESC_TYPE field.
17	Smallint	SQL_DATETIME_SUB	Date-time subcode. If the data type is not datetime, the function returns a null value.
18	Integer	NUM_PREC_RADIX	• 10: Numeric data type • Null value: Other data type
19	Smallint	INTERVAL_PRECISION	The function always returns a null value.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
08S01	Communication link failure		N

SQLSTATE	Description	Remarks	Returned
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY004	Invalid SQL data type		Y
HY008	Operation cancelled		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY117	Connection suspended		N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.5 Specifying and obtaining driver options

This section explains the ODBC functions that are used to specify and obtain driver options.

16.5.1 SQLSetConnectAttr, SQLSetConnectAttrW

(1) Function

This ODBC function sets connection attributes.

(2) Format

- For `SQLSetConnectAttr`

```
SQLRETURN SQLSetConnectAttr
(
  SQLHDBC           ConnectionHandle,   /* In */
  SQLINTEGER        Attribute,          /* In */
  SQLPOINTER        ValuePtr,           /* In */
  SQLINTEGER        StringLength        /* In */
)
```

- For `SQLSetConnectAttrW`

```
SQLRETURN SQLSetConnectAttrW
(
  SQLHDBC           ConnectionHandle,   /* In */
  SQLINTEGER        Attribute,          /* In */
  SQLPOINTER        ValuePtr,           /* In */
  SQLINTEGER        StringLength        /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle for which connection attributes are to be set.

`Attribute`

Specifies the connection attributes to be set. For details about the attributes that can be specified, see 16.14 Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW.

`ValuePtr`

Specifies a pointer to the values to be associated with `Attribute` or a 32-bit integer value. For details about the values that can be specified, see 16.14 Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW.

`StringLength`

Specifies the length of `*ValuePtr` (in bytes). If `ValuePtr` is an integer, this argument is ignored.

If `ValuePtr` is a pointer to a character string, this argument specifies the length of the character string (in bytes) or `SQL_NTS`.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		Y
08002	Connection name in use		N
08003	Connection does not exist		N
08S01	Communication link failure		N
24000	Invalid cursor status		N
25000	Invalid operation in local transaction		N
3D000	Invalid catalog name		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		N
HY011	Attribute cannot be set now		Y
HY013	Memory management error		N
HY024	Invalid attribute value		Y
HY090	Invalid string or buffer length	The following conditions are all satisfied: • *ValuePtr is a character string. • StringLength is less than 0. • The value of StringLength is not SQL_NTS.	Y
HY092	Invalid attribute or option identifier	--	Y
HY114	The driver does not support asynchronous execution at the connection level		N
HY117	Connection suspended		N
HY121	Cursor library and driver-dependent connection pooling cannot be executed concurrently		N
HYC00	Optional feature not implemented	The specified Attribute is a valid argument, but it is not supported by the driver.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N
IM009	Unable to load translation DLL		N

SQLSTATE	Description	Remarks	Returned
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N
S1118	The driver does not support asynchronous notification		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- If you specify a value for Attribute that is specific to HADB; that is, one that does not exist in the ODBC conventions, include adbodb.h.
- The transaction access mode specified in this function (the attribute is SQL_ATTR_ACCESS_MODE) is determined based on the priorities shown in the following table (the smaller the priority number, the higher the priority).

Table 16-5: Priorities of transaction access mode

Priority	Specification location of transaction access mode
1	SQLSetConnectAttr or SQLSetConnectAttrW
2	adb_clt_trn_access_mode operand in the client definition

16.5.2 SQLGetConnectAttr, SQLGetConnectAttrW

(1) Function

This ODBC function returns the current attribute value of a connection handle.

(2) Format

- For SQLGetConnectAttr

```
SQLRETURN SQLGetConnectAttr
(
  SQLHDBC           ConnectionHandle,    /* In */
  SQLINTEGER        Attribute,           /* In */
  SQLPOINTER        ValuePtr,            /* Out */
  SQLINTEGER        BufferLength,        /* In */
  SQLINTEGER      * StringLengthPtr      /* Out */
)
```

- For SQLGetConnectAttrW

```
SQLRETURN SQLGetConnectAttrW
(
  SQLHDBC           ConnectionHandle,    /* In */
  SQLINTEGER        Attribute,           /* In */
  SQLPOINTER        ValuePtr,            /* Out */
  SQLINTEGER        BufferLength,        /* In */
```

```
  SQLINTEGER      * StringLengthPtr      /* Out */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`Attribute`

Specifies the attribute to be obtained. For details about the attributes that can be specified, see 16.14 Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW.

`ValuePtr`

Specifies a pointer to the buffer in which the attribute specified in `Attribute` is to be returned. `NULL` can also be specified in `ValuePtr`.

`BufferLength`

If the type of the attribute value specified in `Attribute` is character string or binary, this argument specifies the length of `*ValuePtr` (in bytes). If it is any other data type, this argument is ignored.

`StringLengthPtr`

Specifies a pointer to the buffer that stores the total valid length (in bytes) of the returned attribute value. This length does not include the null terminating character.

If `ValuePtr` is `NULL`, no value is returned to `*StringLengthPtr`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_NO_DATA`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
08003	Connection does not exist		Y
08S01	Communication link failure		N
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY092	Invalid attribute or option identifier		Y
HYC00	Optional feature not implemented		Y
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

If you specify a value for Attribute that is specific to HADB; that is, one that does not exist in the ODBC conventions, include adbodb.h.

16.5.3 SQLSetEnvAttr

(1) Function

This ODBC function sets environment attributes.

(2) Format

```
SQLRETURN SQLSetEnvAttr
(
  SQLHENV           EnvironmentHandle,    /* In */
  SQLINTEGER        Attribute,            /* In */
  SQLPOINTER        ValuePtr,             /* In */
  SQLINTEGER        StringLength          /* In */
)
```

(3) Arguments

EnvironmentHandle

Specifies an environment handle of a connection whose attributes are to be set.

Attribute

Specifies the environment attributes to be set. For details about the attributes that can be specified, see 16.15 Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr.

ValuePtr

Specifies a pointer to the values associated with Attribute or a 32-bit integer value. For details about the values that can be specified, see 16.15 Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr.

StringLength

If ValuePtr is a pointer to character string or binary, this argument specifies the length of *ValuePtr (in bytes). If ValuePtr is an integer, this argument is ignored.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
HY000	General error		N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY011	Attribute cannot be set now	A connection handle has been assigned to EnvironmentHandle.	N
HY013	Memory management error	--	N
HY024	Invalid attribute value	The value specified in ValuePtr is invalid for the value specified in Attribute.	Y
HY090	Invalid string or buffer length	The value specified in StringLength is less than 0 but is not SQL_NTS.	Y
HY092	Invalid attribute or option identifier	The specified Attribute is invalid.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented	The specified Attribute is a valid argument, but it is not supported by the driver.	Y

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.5.4 SQLGetEnvAttr

(1) Function

This ODBC function returns the current attribute value of an environment handle.

(2) Format

```
SQLRETURN SQLGetEnvAttr
(
  SQLHENV           EnvironmentHandle,   /* In */
  SQLINTEGER        Attribute,           /* In */
  SQLPOINTER        ValuePtr,            /* Out */
  SQLINTEGER        BufferLength,        /* In */
  SQLINTEGER      * StringLengthPtr      /* Out */
)
```

(3) Arguments

EnvironmentHandle

Specifies an environment handle.

Attribute

Specifies the attribute to be obtained. For details about the attributes that can be specified, see 16.15 Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr.

ValuePtr

Specifies a pointer to the buffer in which the attribute specified in `Attribute` is to be returned. `NULL` can also be specified in `ValuePtr`.

BufferLength

If the type of the attribute value specified in `Attribute` is character string or binary, this argument specifies the length of `*ValuePtr` (in bytes). If it is any other data type, this argument is ignored.

StringLengthPtr

Specifies a pointer to the buffer that stores the total number of valid bytes in the returned attribute value. This total number of bytes does not include the number of bytes in the null terminating character. If `ValuePtr` is `NULL`, no value is returned to `*StringLengthPtr`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY092	Invalid attribute or option identifier		Y
HYC00	Optional feature not implemented		Y
IM001	Driver does not support this function		N

Legend:

Y: This `SQLSTATE` might be returned by the HADB ODBC driver.

N: This `SQLSTATE` is not returned by the HADB ODBC driver.

--: None

16.5.5 SQLSetStmtAttr, SQLSetStmtAttrW

(1) Function

This ODBC function sets the attributes related to a statement.

(2) Format

- For `SQLSetStmtAttr`

```
SQLRETURN SQLSetStmtAttr
(
  SQLHSTMT          StatementHandle,      /* In */
  SQLINTEGER        Attribute,            /* In */
  SQLPOINTER        ValuePtr,             /* In */
  SQLINTEGER        StringLength          /* In */
)
```

- For `SQLSetStmtAttrW`

```
SQLRETURN SQLSetStmtAttrW
(
  SQLHSTMT          StatementHandle,      /* In */
  SQLINTEGER        Attribute,            /* In */
  SQLPOINTER        ValuePtr,             /* In */
  SQLINTEGER        StringLength          /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`Attribute`

Specifies the attribute to be set. For details about the attributes that can be specified, see 16.16 Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW.

`ValuePtr`

Specifies the value to be set for the attribute specified in `Attribute`. For details about the attributes that can be specified, see 16.16 Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW.

`StringLength`

If the type of the attribute value specified in `Attribute` is character string or binary, this argument specifies the length of `*ValuePtr` (in bytes). If it is any other data type, this argument is ignored.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		Y
08S01	Communication link failure		N
24000	Invalid cursor status		N
HY000	General error		N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY011	Attribute cannot be set now		Y
HY013	Memory management error		N
HY017	Invalid use of an automatically allocated descriptor handle		Y
HY024	Invalid attribute value		Y
HY090	Invalid string or buffer length		N
HY092	Invalid attribute or option identifier		Y
HYC00	Optional feature not implemented		Y
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.5.6 SQLGetStmtAttr, SQLGetStmtAttrW

(1) Function

This ODBC function returns the current attribute value of a statement handle.

(2) Format

- For SQLGetStmtAttr

```
SQLRETURN SQLGetStmtAttr
(
  SQLHSTMT            StatementHandle,      /* In */
  SQLINTEGER          Attribute,            /* In */
  SQLPOINTER          ValuePtr,             /* Out */
  SQLINTEGER          BufferLength,         /* In */
  SQLINTEGER        * StringLengthPtr       /* Out */
)
```

- For SQLGetStmtAttrW

```
SQLRETURN SQLGetStmtAttrW
(
  SQLHSTMT            StatementHandle,      /* In */
  SQLINTEGER          Attribute,            /* In */
  SQLPOINTER          ValuePtr,             /* Out */
  SQLINTEGER          BufferLength,         /* In */
  SQLINTEGER        * StringLengthPtr       /* Out */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Attribute

Specifies the attribute to be obtained. For details about the attributes that can be specified, see 16.16 Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW.

ValuePtr

Specifies a pointer to the buffer in which the attribute specified in Attribute is to be returned. NULL can also be specified in ValuePtr.

BufferLength

If the type of the attribute value specified in Attribute is character string or binary, this argument specifies the length of *ValuePtr (in bytes). If it is any other data type, this argument is ignored.

StringLengthPtr

Specifies a pointer to the buffer in which the total number of valid bytes in the returned attribute value is to be stored. This value does not include the length in bytes of a null terminating character. If ValuePtr is NULL, no value is returned to *StringLengthPtr.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
24000	Invalid cursor status		N
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY092	Invalid attribute or option identifier		Y

SQLSTATE	Description	Remarks	Returned
HY109	Invalid cursor position		N
HYC00	Optional feature not implemented		Y
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.6 Specifying descriptor values

This section explains the ODBC functions that are used to specify descriptor values.

16.6.1 SQLGetDescField, SQLGetDescFieldW

(1) Function

This ODBC function returns the value of a descriptor field specified in the arguments.

(2) Format

- For `SQLGetDescField`

```
SQLRETURN SQLGetDescField
(
  SQLHDESC         DescriptorHandle,          /* In */
  SQLSMALLINT      RecNumber,                 /* In */
  SQLSMALLINT      FieldIdentifier,           /* In */
  SQLPOINTER       ValuePtr,                  /* Out */
  SQLINTEGER       BufferLength,              /* In */
  SQLINTEGER     * StringLengthPtr            /* Out */
)
```

- For `SQLGetDescFieldW`

```
SQLRETURN SQLGetDescFieldW
(
  SQLHDESC         DescriptorHandle,          /* In */
  SQLSMALLINT      RecNumber,                 /* In */
  SQLSMALLINT      FieldIdentifier,           /* In */
  SQLPOINTER       ValuePtr,                  /* Out */
  SQLINTEGER       BufferLength,              /* In */
  SQLINTEGER     * StringLengthPtr            /* Out */
)
```

(3) Arguments

`DescriptorHandle`

Specifies a descriptor handle.

`RecNumber`

Specifies a column number (`ARD` or `IRD`) or a parameter number (`APD` or `IPD`).

If `FieldIdentifier` indicates a header field, this argument is ignored.

`FieldIdentifier`

Specifies the field (header or record field) of the descriptor whose value is to be returned. For details about the attributes that can be specified, see 16.17 Attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW.

`ValuePtr`

Specifies a pointer to the buffer in which the descriptor information is to be returned. `NULL` can also be specified in this argument.

BufferLength

If the data type of the descriptor field specified in FieldIdentifier is character string or binary, this argument specifies the length of *ValuePtr (in bytes). If it is any other data type, this argument is ignored. SQL_NTS cannot be specified.

StringLengthPtr

Specifies a pointer to the buffer that stores the valid length of the value that is set in *ValuePtr.

A value is set in ValuePtr even when NULL is specified in this argument.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	The column name was truncated because the value specified in BufferLength was smaller than the length of the column name in bytes. The length of the untruncated column name is stored in *StringLengthPtr. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
07009	Invalid descriptor index	The value specified in RecNumber is less than or equal to 0.	Y
08S01	Communication link failure	--	N
HY001	Memory allocation error		N
HY007	Associated statement is not prepared		Y
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle associated with DescriptorHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY021	Inconsistent descriptor information		N
HY090	Invalid string or buffer length	• *ValuePtr is a character string and the value specified in BufferLength is less than 0.	Y

SQLSTATE	Description	Remarks	Returned
		• An invalid value (SQL_NTS) was specified in BufferLength.	
HY091	Invalid descriptor field identifier	FieldIdentifier is not an ODBC-defined field.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.6.2 SQLGetDescRec, SQLGetDescRecW

(1) Function

This ODBC function returns the current settings or values of multiple fields of a descriptor record specified in the arguments. This function returns the fields that describe the name, data type, and storage of column or parameter data.

(2) Format

- For SQLGetDescRec

```
SQLRETURN SQLGetDescRec
(
  SQLHDESC          DescriptorHandle,       /* In */
  SQLSMALLINT       RecNumber,              /* In */
  SQLCHAR         * Name,                   /* Out */
  SQLSMALLINT       BufferLength,           /* In */
  SQLSMALLINT     * StringLengthPtr,        /* Out */
  SQLSMALLINT     * TypePtr,                /* Out */
  SQLSMALLINT     * SubTypePtr,             /* Out */
  SQLLEN          * LengthPtr,              /* Out */
  SQLSMALLINT     * PrecisionPtr,           /* Out */
  SQLSMALLINT     * ScalePtr,               /* Out */
  SQLSMALLINT     * NullablePtr             /* Out */
)
```

- For SQLGetDescRecW

```
SQLRETURN SQLGetDescRecW
(
  SQLHDESC          DescriptorHandle,       /* In */
  SQLSMALLINT       RecNumber,              /* In */
  SQLWCHAR        * Name,                   /* Out */
  SQLSMALLINT       BufferLength,           /* In */
  SQLSMALLINT     * StringLengthPtr,        /* Out */
  SQLSMALLINT     * TypePtr,                /* Out */
  SQLSMALLINT     * SubTypePtr,             /* Out */
  SQLLEN          * LengthPtr,              /* Out */
  SQLSMALLINT     * PrecisionPtr,           /* Out */
  SQLSMALLINT     * ScalePtr,               /* Out */
```

```
  SQLSMALLINT    * NullablePtr            /* Out */
)
```

(3) Arguments

`DescriptorHandle`

Specifies a descriptor handle.

`RecNumber`

Specifies a column number (`ARD` or `IRD`) or a parameter number (`APD` or `IPD`).

`Name`

Specifies a pointer to the buffer in which the `SQL_DESC_NAME` field value in the descriptor record is to be returned.

`BufferLength`

Specifies the length# of the `*Name` buffer.

This length includes the null terminating character. `SQL_NTS` cannot be specified.

`StringLengthPtr`

Specifies a pointer to the buffer that stores the valid length# of the data that is returned to the `*Name` buffer. This length does not include the null terminating character.

Important

If the length# of `*Name` stored here is greater than the value of `BufferLength` without the length# of the null terminating character, the character string stored in `*Name` is truncated to the length# equivalent to `BufferLength` without the null terminating character, and then the null terminating character is added at the end.

#

The length must be in bytes for `SQLGetDescRec` and in characters for `SQLGetDescRecW`.

`TypePtr`

Specifies a pointer to the buffer in which the `SQL_DESC_TYPE` field value in the descriptor record is to be returned.

`SubTypePtr`

Specifies a pointer to the buffer in which the `SQL_DESC_DATETIME_INTERVAL_CODE` field value in the descriptor record is to be returned.

`LengthPtr`

Specifies a pointer to the buffer in which the `SQL_DESC_OCTET_LENGTH` field value in the descriptor record is to be returned.

`PrecisionPtr`

Specifies a pointer to the buffer in which the `SQL_DESC_PRECISION` field value in the descriptor record is to be returned.

`ScalePtr`

Specifies a pointer to the buffer in which the `SQL_DESC_SCALE` field value in the descriptor record is to be returned.

`NullablePtr`

Specifies a pointer to the buffer in which the `SQL_DESC_NULLABLE` field value in the descriptor record is to be returned.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	The column name was truncated because the value specified in BufferLength was smaller than the length of the column name in bytes. The length of the untruncated column name is stored in *StringLengthPtr. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
07009	Invalid descriptor index	The value specified in RecNumber is less than or equal to 0.	Y
08S01	Communication link failure	--	N
HY000	General error		N
HY001	Memory allocation error		N
HY007	Associated statement is not prepared		Y
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle associated with DescriptorHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY090	Invalid string or buffer length	An invalid value (SQL_NTS) was specified in BufferLength.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.6.3 SQLSetDescField, SQLSetDescFieldW

(1) Function

This ODBC function sets the value of a descriptor field specified in the arguments.

(2) Format

- For `SQLSetDescField`

```
SQLRETURN SQLSetDescField
(
  SQLHDESC      DescriptorHandle,       /* In */
  SQLSMALLINT   RecNumber,              /* In */
  SQLSMALLINT   FieldIdentifier,        /* In */
  SQLPOINTER    ValuePtr,               /* In */
  SQLINTEGER    BufferLength            /* In */
)
```

- For `SQLSetDescFieldW`

```
SQLRETURN SQLSetDescFieldW
(
  SQLHDESC      DescriptorHandle,       /* In */
  SQLSMALLINT   RecNumber,              /* In */
  SQLSMALLINT   FieldIdentifier,        /* In */
  SQLPOINTER    ValuePtr,               /* In */
  SQLINTEGER    BufferLength            /* In */
)
```

(3) Arguments

`DescriptorHandle`

Specifies a descriptor handle.

`RecNumber`

Specifies a column number (`ARD` or `IRD`) or a parameter number (`APD` or `IPD`).

If `FieldIdentifier` indicates a header field, this argument is ignored.

If there is no descriptor record with the record number specified in this argument, this function creates a descriptor record with that record number.

`FieldIdentifier`

Specifies the field (header or record field) of the descriptor whose value is to be set. For details about the attributes that can be specified, see 16.17 Attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW.

`ValuePtr`

Specifies the descriptor information to be set (pointer or integer value). For details about the values that can be specified, see 16.17 Attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW.

`BufferLength`

If the data type of the descriptor field specified in `FieldIdentifier` is character string or binary, this argument specifies the length of `*ValuePtr` (in bytes). If it is any other data type, this argument is ignored.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
07009	Invalid descriptor index	• The value specified in RecNumber is less than or equal to 0. • RecNumber is greater than the maximum number of columns or parameters supported by the data source and DescriptorHandle is associated with an ARD or APD. • FieldIdentifier is SQL_DESC_COUNT and the value specified in ValuePtr is less than 0.	Y
08S01	Communication link failure	--	N
22001	Character string data was right-truncated		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
HY001	Memory allocation error	--	Y
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle associated with DescriptorHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY016	Cannot modify an implementation row descriptor	DescriptorHandle is associated with an IRD, but FieldIdentifier is not SQL_DESC_ARRAY_STATUS_PTR or SQL_DESC_ROWS_PROCESSED_PTR	Y
HY021	Inconsistent descriptor information	--	N
HY090	Invalid string or buffer length		N
HY091	Invalid descriptor field identifier	• The value specified in FieldIdentifier is neither an ODBC-defined field nor an implementation-defined value. • FieldIdentifier is invalid for DescriptorHandle.	Y
HY092	Invalid attribute or option identifier	• The value set in ValuePtr is invalid for the value specified in FieldIdentifier.	Y

SQLSTATE	Description	Remarks	Returned
		• FieldIdentifier is SQL_DESC_UNNAMED and ValuePtr is SQL_NAMED.	
HY105	Invalid parameter type	The value specified in the SQL_DESC_PARAMETER_TYPE field is invalid (the value is not SQL_PARAM_INPUT).	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.6.4 SQLSetDescRec

(1) Function

This ODBC function sets values in multiple descriptor record fields in the same descriptor record.

(2) Format

```
SQLRETURN SQLSetDescRec
(
  SQLHDESC         DescriptorHandle,    /* In */
  SQLSMALLINT      RecNumber,           /* In */
  SQLSMALLINT      Type,                /* In */
  SQLSMALLINT      SubType,             /* In */
  SQLLEN           Length,              /* In */
  SQLSMALLINT      Precision,           /* In */
  SQLSMALLINT      Scale,               /* In */
  SQLPOINTER       DataPtr,             /* Deferred  In */
  SQLLEN         * StringLengthPtr,     /* Deferred  In */
  SQLLEN         * IndicatorPtr         /* Deferred  In */
)
```

(3) Arguments

DescriptorHandle

Specifies a descriptor handle. This must not be an IRD handle.

RecNumber

Specifies a column number (ARD or IRD) or a parameter number (APD or IPD).

If there is no descriptor record with the record number specified in this argument, this function creates a descriptor record with that record number.

Type

Specifies the value to be set in the SQL_DESC_TYPE field for the descriptor record.

SubType

Specifies the value to be set in the SQL_DESC_DATETIME_INTERVAL_CODE field for the descriptor record.

Length

Specifies the value to be set in the SQL_DESC_OCTET_LENGTH field for the descriptor record.

Precision

Specifies the value to be set in the SQL_DESC_PRECISION field for the descriptor record.

Scale

Specifies the value to be set in the SQL_DESC_SCALE field for the descriptor record.

DataPtr

Specifies the value to be set in the SQL_DESC_DATA_PTR field for the descriptor record.

StringLengthPtr

Specifies the value to be set in the SQL_DESC_OCTET_LENGTH_PTR field for the descriptor record.

IndicatorPtr

Specifies the value to be set in the SQL_DESC_INDICATOR_PTR field for the descriptor record.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
07009	Invalid descriptor index	• RecNumber is greater than the maximum number of columns or parameters supported by the data source and DescriptorHandle is associated with an ARD, APD, or IPD. • The value specified in RecNumber is less than or equal to 0.	Y
08S01	Communication link failure	--	N
HY000	General error		N
HY001	Memory allocation error		Y
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle associated with DescriptorHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY016	Cannot modify an implementation row descriptor	DescriptorHandle is associated with an IRD.	Y

SQLSTATE	Description	Remarks	Returned
HY021	Inconsistent descriptor information	--	Y
HY090	Invalid string or buffer length		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.6.5 SQLCopyDesc

(1) Function

This ODBC function copies descriptor information from one descriptor handle to another.

(2) Format

```
SQLRETURN SQLCopyDesc
(
  SQLHDESC     SourceDescHandle,              /* In */
  SQLHDESC     TargetDescHandle               /* In */
)
```

(3) Arguments

SourceDescHandle

Specifies a source descriptor handle.

TargetDescHandle

Specifies a target descriptor handle. For this argument, you can specify a handle to an application descriptor or an IPD, but not a handle to an IRD.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
HY000	General error		N

SQLSTATE	Description	Remarks	Returned
HY001	Memory allocation error		Y
HY007	Associated statement is not prepared		Y
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle associated with the descriptor handle of SourceDescHandle or TargetDescHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY016	Cannot modify an implementation row descriptor	TargetDescHandle was associated with an IRD.	Y
HY021	Inconsistent descriptor information	--	N
HY092	Invalid attribute or option identifier		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

If the return value is not SQL_SUCCESS, execute SQLEndTran with SQL_ROLLBACK specified for CompletionType or execute SQLFreeHandle with SQL_HANDLE_STMT specified for HandleType.

16.7 Creating SQL requests

This section explains the ODBC functions that are used to create SQL requests.

16.7.1 SQLPrepare, SQLPrepareW

(1) Function

This ODBC function sends an SQL statement to the data source and preprocesses the SQL statement.

(2) Format

- For `SQLPrepare`

```
SQLRETURN SQLPrepare
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLCHAR        * StatementText,       /* In */
  SQLINTEGER       TextLength           /* In */
)
```

- For `SQLPrepareW`

```
SQLRETURN SQLPrepareW
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLWCHAR       * StatementText,       /* In */
  SQLINTEGER       TextLength           /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`StatementText`

Specifies an SQL character string. Comments (bracketed between `/*` and `*/`) are not permitted within an SQL character string, but specifications bracketed between `/*>>` and `<<*/` (for example, index specifications) are permitted within an SQL character string.

`TextLength`

Specifies the length of `*StatementText` (in bytes for `SQLPrepare` and in characters for `SQLPrepareW`).

If the SQL character string specified in `StatementText` guarantees a null terminating character, `SQL_NTS` can be specified in this argument.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
08S01	Communication link failure		N
21S01	Insert value list does not match column list		N
21S02	Degree of derived table does not match column list		N
22018	Invalid character value for cast specification		N
22019	Invalid escape character		N
22025	Invalid escape sequence		N
24000	Invalid cursor status		N
34000	Invalid cursor name		N
3D000	Invalid catalog name		N
3F000	Invalid schema name		N
42000	Syntax error or access violation		N
42S01	Base table or view already exists		N
42S02	Base table or view not found		N
42S11	Index already exists		N
42S12	Index not found		N
42S21	Column already exists		N
42S22	Column not found		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	Y
5C051	The text character string for an SQL statement exceeded 16,000,000 characters	Even if the text character string for an SQL statement consists of 16,000,000 or fewer characters, the result after character encoding conversion by the driver manager might exceed 16,000,000 characters. SQLSTATE is also returned in such a case.	Y
HY000	General error	--	N
HY001	Memory allocation error	The memory required to execute or complete the function has not been allocated for the driver.	Y
HY008	Operation cancelled	--	N
HY009	Invalid use of null pointer	A null pointer is specified for StatementText.	Y
HY010	Function sequence error	--	Y

SQLSTATE	Description	Remarks	Returned
HY013	Memory management error		N
HY090	Invalid string or buffer length	The value specified in TextLength is less than or equal to 0, or not SQL_NTS.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.7.2 SQLBindParameter

(1) Function

This ODBC function binds a buffer to a dynamic parameter in an SQL statement.

(2) Format

```
SQLRETURN SQLBindParameter
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLUSMALLINT     ParameterNumber,     /* In */
  SQLSMALLINT      InputOutputType,     /* In */
  SQLSMALLINT      ValueType,           /* In */
  SQLSMALLINT      ParameterType,       /* In */
  SQLULEN          ColumnSize,          /* In */
  SQLSMALLINT      DecimalDigits,       /* In */
  SQLPOINTER       ParameterValuePtr,   /* In */
  SQLLEN           BufferLength,        /* In */
  SQLLEN         * StrLen_or_IndPtr     /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

ParameterNumber

Specifies a parameter number. Parameter numbers are assigned in ascending order, beginning with 1.

InputOutputType

Specifies the type of the parameter:

- SQL_PARAM_INPUT

  Input parameter.

- SQL_PARAM_INPUT_OUTPUT

  Input/output parameter.

  Note that input/output parameters are not supported. If SQL_PARAM_INPUT_OUTPUT is specified, the driver assumes that SQL_PARAM_INPUT is specified.

ValueType

Specifies the C data type of the parameter, or specifies SQL_C_DEFAULT.

If SQL_C_DEFAULT is specified, the driver assumes the default C data type.

For details about the supported data types, see 15.3.2 Correspondence between ODBC's SQL data types and C data types. If an unsupported C data type is specified, an error results.

ParameterType

Specifies the ODBC SQL data type of the parameter.

For details about the supported data types, see 15.3.1 Correspondence between ODBC's SQL data types and HADB's data types. If an unsupported ODBC SQL data type is specified, an error results.

ColumnSize

Specifies in bytes the size of the data of the corresponding dynamic parameter.

If ParameterType is SQL_CHAR, SQL_VARCHAR, SQL_DECIMAL, or SQL_DOUBLE, the value of ColumnSize is used.

For any other data type, this argument is ignored.

DecimalDigits

Specifies the number of decimal places of the column or expression of the corresponding dynamic parameter.

If ParameterType is SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_DECIMAL, or SQL_DOUBLE, the value of DecimalDigits is used.

For any other data type, this argument is ignored.

ParameterValuePtr

Specifies a pointer to the buffer for the parameter data. The data type must be in the format specified by ValueType. If *StrLen_or_IndPtr is SQL_NULL_DATA or SQL_DATA_AT_EXEC, a null pointer can be specified.

If *StrLen_or_IndPtr is the result of the SQL_LEN_DATA_AT_EXEC(length) macro or SQL_DATA_AT_EXEC, ParameterValuePtr is an application-defined 32-bit value that is associated with the parameter.

BufferLength

For character-type C data, this argument specifies in bytes the length of the ParameterValuePtr buffer. For any other C data, this argument is ignored.

StrLen_or_IndPtr

Specifies a pointer to the buffer that stores one of the following values:

- Length of the parameter value stored in *ParameterValuePtr

  This value is not used for data other than character-type C data.

- SQL_NTS

  The parameter value is a null terminating character.

- SQL_NULL_DATA
  The parameter value is NULL.
- Result of the SQL_LEN_DATA_AT_EXEC(length) macro
  SQLPutData is used. For length, specify 0 or an integer.
- SQL_DATA_AT_EXEC
  SQLPutData is used.

If StrLen_or_IndPtr is a null pointer, the driver assumes that all input parameter values are not NULL and that character string data is null-terminated.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
07006	Restricted data type attribute violation		N
07009	Invalid descriptor index		Y
HY000	General error		N
HY001	Memory allocation error	The memory required to execute or complete the function has not been allocated for the driver.	Y
HY003	Invalid application buffer data type	--	Y
HY004	Invalid SQL data type		Y
HY009	Invalid use of null pointer		Y
HY010	Function sequence error		Y
HY013	Memory management error		N
HY021	Inconsistent descriptor information		N
HY090	Invalid string or buffer length	• The value specified in BufferLength is less than 0. • A null pointer is specified for SQLBindParameter, but the parameter length does not match 0, or SQL_NULL_DATA, or SQL_DATA_AT_EXEC, or the parameter length is greater than SQL_LEN_DATA_AT_EXEC_OFFSET.	Y
HY104	Invalid precision or scale value	--	N
HY105	Invalid parameter type	The value specified in InputOutputType is invalid.	Y
HY117	Connection suspended	--	N

SQLSTATE	Description	Remarks	Returned
HYC00	Optional feature not implemented	The value of ValueType does not match the value of ParameterType.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- The bound values are in effect until any of the following occurs:
  - SQLBindParameter with the same ParameterNumber specified is called again
  - SQLFreeStmt with the SQL_RESET_PARAMS option specified is called
  - SQLSetDescField is called and 0 is set in the SQL_DESC_COUNT header field of an APD
- Rebinding with offsets is not supported.

16.7.3 SQLGetCursorName, SQLGetCursorNameW

(1) Function

This ODBC function returns the cursor name associated with a specified statement handle.

(2) Format

- For SQLGetCursorName

```
SQLRETURN SQLGetCursorName
(
  SQLHSTMT         StatementHandle,        /* In */
  SQLCHAR        * CursorName,             /* Out */
  SQLSMALLINT      BufferLength,           /* In */
  SQLSMALLINT    * NameLengthPtr           /* Out */
)
```

- For SQLGetCursorNameW

```
SQLRETURN SQLGetCursorNameW
(
  SQLHSTMT         StatementHandle,        /* In */
  SQLWCHAR       * CursorName,             /* Out */
  SQLSMALLINT      BufferLength,           /* In */
  SQLSMALLINT    * NameLengthPtr           /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`CursorName`

Specifies a pointer to the buffer in which the cursor name is to be returned.

`BufferLength`

Specifies the length# of `*CursorName`. `SQL_NTS` cannot be specified.

`NameLengthPtr`

Specifies a pointer to the buffer that stores the total valid length# that is returned to `*CursorName`. This length does not include the null terminating character.

#

The length must be in bytes for `SQLGetCursorName` and in characters for `SQLGetCursorNameW`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
`01000`	General warning	--	N
`01004`	Character string data was right-truncated	The cursor name could not be stored because the `*CursorName` buffer was too small (the cursor name was truncated). The function returns `SQL_SUCCESS_WITH_INFO`.	Y
`01S51`	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns `SQL_SUCCESS_WITH_INFO`.	Y
`HY000`	General error	--	N
`HY001`	Memory allocation error		N
`HY010`	Function sequence error		Y
`HY013`	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
`HY015`	No cursor name available	--	N
`HY090`	Invalid string or buffer length	• The value specified in `BufferLength` is less than `0`. • An invalid value (`SQL_NTS`) was specified in `BufferLength`.	Y
`HYT01`	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using	N

SQLSTATE	Description	Remarks	Returned
		SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	
IM001	Driver does not support this function	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- If the SQLSetCursorName function or SQLSetCursorNameW function has not been used and no explicit cursor name has been set, this function returns the cursor name that is generated automatically by the HADB ODBC driver, which is SQL_CUR*XXXXX* (where *XXXXX* is a sequential number beginning with 00001).
- Use the cursor library provided by Microsoft if you intend to use the cursor name obtained by this function. If the cursor library provided by Microsoft is not used, the HADB ODBC driver ignores the cursor name associated with a statement handle.

16.7.4 SQLSetCursorName, SQLSetCursorNameW

(1) Function

This ODBC function assigns a cursor name to an active statement handle. If an application does not call this function, the HADB ODBC driver generates cursor names.

(2) Format

- For SQLSetCursorName

```
SQLRETURN SQLSetCursorName
(
  SQLHSTMT          StatementHandle,      /* In */
  SQLCHAR         * CursorName,           /* In */
  SQLSMALLINT       NameLength            /* In */
)
```

- For SQLSetCursorNameW

```
SQLRETURN SQLSetCursorName
(
  SQLHSTMT          StatementHandle,      /* In */
  SQLWCHAR        * CursorName,           /* In */
  SQLSMALLINT       NameLength            /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`CursorName`

Specifies a cursor name to be associated.

`NameLength`

Specifies the length of `*CursorName` (in bytes for `SQLSetCursorName` and in characters for `SQLSetCursorNameW`).

If the character string specified in `CursorName` guarantees a null terminating character, `SQL_NTS` can be specified in this argument.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
24000	Invalid cursor status	The statement associated with `StatementHandle` has already executed or is engaged in cursor positioning.	Y
34000	Invalid cursor name	• The size of the cursor name is greater than the maximum size (30 bytes) supported by the HADB ODBC driver. • The cursor name specified in `CursorName` begins with `SQL_CUR`.	Y
3C000	Duplicate cursor name	The cursor name specified in `*CursorName` already exists.	Y
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
HY000	General error	--	N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer	`CursorName` is a null pointer.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	The value specified in `NameLength` is less than or equal to `0`, or not `SQL_NTS`.	Y
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using	N

SQLSTATE	Description	Remarks	Returned
		SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	
IM001	Driver does not support this function	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

Use the cursor library provided by Microsoft if you intend to use the cursor name set by this function. If the cursor library provided by Microsoft is not used, the HADB ODBC driver ignores the set cursor name.

16.7.5 SQLDescribeParam

(1) Function

This ODBC function returns information about a dynamic parameter that has been obtained by executing SQLPrepare. This information is set in the fields of the IPD.

(2) Format

```
SQLRETURN SQLDescribeParam
(
  SQLHSTMT          StatementHandle,    /* In */
  SQLUSMALLINT      ParameterNumber,    /* In */
  SQLSMALLINT     * DataTypePtr,        /* Out */
  SQLULEN         * ParameterSizePtr,   /* Out */
  SQLSMALLINT     * DecimalDigitsPtr,   /* Out */
  SQLSMALLINT     * NullablePtr         /* Out */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

ParameterNumber

Specifies a parameter number. Parameter numbers are assigned in ascending order, beginning with 1.

DataTypePtr

Specifies a pointer to the buffer in which the parameter's SQL data type is to be returned.

ParameterSizePtr

Specifies a pointer to the buffer in which the size of the column or expression of the corresponding parameter is returned. The information defined by the data source is returned.

DecimalDigitsPtr

Specifies a pointer to the buffer in which the number of decimal places in the column or expression of the corresponding parameter is returned. The information defined by the data source is returned.

NullablePtr

Specifies a pointer to the buffer in which a value indicating whether the parameter accepts null values is returned. This value is read from the SQL_DESC_NULLABLE field of the IPD. One of the following values is returned:

- SQL_NO_NULLS

  The parameter does not accept null values.

- SQL_NULLABLE

  The parameter accepts null values.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
07009	Invalid descriptor index		Y
08S01	Communication link failure		N
21S01	Insert value list does not match column list		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.7.6 SQLNumParams

(1) Function

This ODBC function returns the number of parameters in a prepared SQL statement.

(2) Format

```
SQLRETURN SQLNumParams
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLSMALLINT    * ParameterCountPtr    /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`ParameterCountPtr`

Specifies a pointer to the buffer in which the number of parameters in the SQL statement is to be returned.

The number of parameters contained in the SQL statement passed to `SQLPrepare` before this function is executed is set.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	`NULL` is specified in `ParameterCountPtr`.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY117	Connection suspended		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This `SQLSTATE` might be returned by the HADB ODBC driver.

N: This `SQLSTATE` is not returned by the HADB ODBC driver.

--: None

16.8 Executing SQL statements

This section explains the ODBC functions that are used when SQL statements are executed.

16.8.1 SQLExecute

(1) Function

This ODBC function executes a prepared statement using the current parameter values if parameter markers exist in the statement.

(2) Format

```
SQLRETURN SQLExecute
(
  SQLHSTMT         StatementHandle     /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, `SQL_NO_DATA`, `SQL_NEED_DATA`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01001	Cursor operation conflict		N
01003	`NULL` value eliminated in set function		N
01004	Character string data was right-truncated		Y
01006	Privilege not revoked		N
01007	Privilege not granted		N
01S02	Option value changed		N
01S07	Fractional truncation		N
07002	`COUNT` field incorrect	The number of parameters specified in `SQLBindParameter` does not match the number of dynamic parameters.	Y

SQLSTATE	Description	Remarks	Returned
07006	Restricted data type attribute violation	--	N
07007	Restricted parameter value violation		N
07S01	Invalid use of default parameter		N
08S01	Communication link failure		N
08003	Connection does not exist		Y
21S02	Degree of derived table does not match column list		N
22001	Character string data was right-truncated		Y
22002	Required indicator variable not supplied		N
22003	Numeric value out of range		Y
22007	Invalid datetime format		Y
22008	Datetime field overflow		Y
22012	Division by zero		N
22015	Interval field overflow		N
22018	Invalid character value for cast specification		Y
22019	Invalid escape character		N
22025	Invalid escape sequence		N
23000	Integrity constraint violation		N
24000	Invalid cursor status		Y
40001	Serialization failure		N
40003	Statement completion unknown		N
42000	Syntax error or access violation		N
44000	`WITH CHECK OPTION` violation		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C036	Data conversion error	The contents of the requested input data are invalid.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	Y
HY000	General error	--	N
HY001	Memory allocation error		Y
HY003	Invalid C data type		Y
HY004	Invalid SQL data type		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		Y
HY010	Function sequence error	`Prepare` has not been executed yet.	Y
HY013	Memory management error	--	Y

SQLSTATE	Description	Remarks	Returned
HY014	Invalid precision or scale value		Y
HY090	Invalid string or buffer length		Y
HY104	Invalid precision or scale value		Y
HY105	Invalid parameter type		N
HY109	Invalid cursor position		N
HY117	Connection suspended	The Disconnect and r-only functions are permitted.	N
HYC00	Optional feature not implemented	--	Y
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- Processing for handling Microsoft Access(R) Version 2.0 is not supported.
- The bookmark feature is not supported.

16.8.2 SQLExecDirect, SQLExecDirectW

(1) Function

This ODBC function executes a prepared SQL statement. If the SQL statement contains parameter markers, this ODBC function uses the current values of the parameter marker variables to execute the SQL statement. SQLExecDirect is the fastest way to issue an SQL statement for one-time execution.

(2) Format

- For SQLExecDirect

```
SQLRETURN SQLExecDirect
(
  SQLHSTMT        StatementHandle,     /* In */
  SQLCHAR       * StatementText,       /* In */
  SQLINTEGER      TextLength           /* In */
)
```

- For SQLExecDirectW

```
SQLRETURN SQLExecDirectW
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLWCHAR       * StatementText,       /* In */
  SQLINTEGER       TextLength           /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`StatementText`

Specifies the SQL character string to be executed. A null pointer cannot be specified. Specify a character string consisting of at least one character. Comments (bracketed between `/*` and `*/`) are not permitted within an SQL character string, but specifications bracketed between `/*>>` and `<<*/` (for example, index specifications) are permitted within an SQL character string.

`TextLength`

Specifies the length of `*StatementText`. The handling depends on the specified value, as explained in the following table:

Value specified in TextLength	Handling of TextLength
Integer value greater than `0`	The specified data length from the beginning of `*StatementText` takes effect (in bytes for `SQLExecDirect` and in characters for `SQLExecDirectW`).
`SQL_NTS`	The value of `TextLength` is ignored, and from the beginning of `*StatementText` through `NULL` takes effect.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_NO_DATA`, `SQL_ERROR`, `SQL_NEED_DATA`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
`01000`	General warning	--	N
`01001`	Cursor operation conflict		N
`01003`	`NULL` value eliminated in set function		N
`01004`	Character string data was right-truncated		Y
`01006`	Privilege not revoked		N
`01007`	Privilege not granted		N
`01S02`	Option value changed		N
`01S07`	Fractional truncation		N

SQLSTATE	Description	Remarks	Returned
07002	COUNT field incorrect	The number of parameters specified in SQLBindParameter does not match the number of dynamic parameters.	Y
07006	Restricted data type attribute violation	--	N
07S01	Invalid use of default parameter		N
08003	Connection does not exist		Y
08S01	Communication link failure		N
21S01	Insert value list does not match column list		N
21S02	Degree of derived table does not match column list		N
22001	Character string data was right-truncated		Y
22002	Required indicator variable not supplied		N
22003	Numeric value out of range		Y
22007	Invalid datetime format		Y
22008	Datetime field overflow		Y
22012	Division by zero		N
22015	Interval field overflow		N
22018	Invalid character value for cast specification		Y
22019	Invalid escape character		N
22025	Invalid escape sequence		N
23000	Integrity constraint violation		N
24000	Invalid cursor status		Y
34000	Invalid cursor name		N
3D000	Invalid catalog name		N
3F000	Invalid schema name		N
40001	Serialization failure		N
40003	Statement completion unknown		N
42000	Syntax error or access violation		N
42S01	Base table or view already exists		N
42S02	Base table or view not found		N
42S11	Index already exists		N
42S12	Index not found		N
42S21	Column already exists		N
42S22	Column not found		N
44000	WITH CHECK OPTION violation		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y

SQLSTATE	Description	Remarks	Returned
5C036	Data conversion error	The contents of the requested input data are invalid.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	Y
5C051	The text character string for an SQL statement exceeded 16,000,000 characters	Even if the text character string for an SQL statement consists of 16,000,000 or fewer characters, the result after character encoding conversion by the driver manager might exceed 16,000,000 characters. SQLSTATE is also returned in such a case.	Y
HY000	General error	--	N
HY001	Memory allocation error		Y
HY003	Invalid C data type		Y
HY004	Invalid SQL data type		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		Y
HY010	Function sequence error		Y
HY013	Memory management error		Y
HY014	Invalid precision or scale value		Y
HY090	Invalid string or buffer length		Y
HY104	Invalid precision or scale value		Y
HY105	Invalid parameter type		N
HY109	Invalid cursor position		N
HYC00	Optional feature not implemented		Y
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

- Processing for handling Microsoft Access(R) Version 2.0 is not supported.
- The bookmark feature is not supported.

16.8.3 SQLNativeSql, SQLNativeSqlW

(1) Function

This ODBC function returns an SQL character string as modified by the HADB ODBC driver. This function does not execute the SQL statement.

(2) Format

- For `SQLNativeSql`

```
SQLRETURN SQLNativeSql
(
  SQLHDBC          ConnectionHandle,          /* In */
  SQLCHAR        * InStatementText,           /* In */
  SQLINTEGER       TextLength1,               /* In */
  SQLCHAR        * OutStatementText,          /* Out */
  SQLINTEGER       BufferLength,              /* In */
  SQLINTEGER     * TextLength2Ptr             /* Out */
)
```

- For `SQLNativeSqlW`

```
SQLRETURN SQLNativeSqlW
(
  SQLHDBC          ConnectionHandle,          /* In */
  SQLWCHAR       * InStatementText,           /* In */
  SQLINTEGER       TextLength1,               /* In */
  SQLWCHAR       * OutStatementText,          /* Out */
  SQLINTEGER       BufferLength,              /* In */
  SQLINTEGER     * TextLength2Ptr             /* Out */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`InStatementText`

Specifies the source SQL character string. Comments (bracketed between `/*` and `*/`) are not permitted within an SQL character string, but specifications bracketed between `/*>>` and `<<*/` (for example, index specifications) are permitted within an SQL character string.

`TextLength1`

Specifies the length# of `*InStatementText`.

If the SQL character string specified in `InStatementText` ends with a null terminating character, you can specify `SQL_NTS`.

`OutStatementText`

Specifies a pointer to the buffer in which the SQL character string obtained after conversion is to be returned.

`BufferLength`

Specifies the length# of `*OutStatementText`.

This length includes the length of the null terminating character. `SQL_NTS` cannot be specified.

TextLength2Ptr

Specifies a pointer to the buffer that returns the total valid length# to be returned to *OutStatementText. The HADB ODBC driver returns the valid length# of the SQL character string. This length does not include the null terminating character.

> **Important**
>
> If the length# of the SQL character string stored here is greater than the value of BufferLength without the length# of the null terminating character, the character string stored in OutStatementText is truncated to the length# equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end.

#

The length must be in bytes for SQLNativeSql and in characters for SQLNativeSqlW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	The entire SQL character string could not be stored because the *OutStatementText buffer was too small (the information was truncated). The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
08003	Connection does not exist	ConnectionHandle is not a connection status.	Y
08S01	Communication link failure	--	N
22007	Invalid datetime format	An invalid date, time, or time stamp value is specified in the escape clause of *InStatementText.	N
24000	Invalid cursor status	The cursor specified by the statement is located before or after the result set.	N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C051	The text character string for an SQL statement exceeded 16,000,000 characters	Even if the text character string for an SQL statement consists of 16,000,000 or fewer characters, the result after character encoding conversion by the driver manager might exceed 16,000,000 characters. SQLSTATE is also returned in such a case.	Y

SQLSTATE	Description	Remarks	Returned
HY000	General error	--	N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer	*InStatementText is a null pointer.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	• The value specified in TextLength1 is less than 0 and not SQL_NTS. • BufferLength is less than 0, but OutStatementText is not a null pointer. • An invalid value (SQL_NTS) was specified in BufferLength.	Y
HY109	Invalid cursor position	The current row of the cursor has already been deleted or cannot be fetched.	N
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
IM001	Driver does not support this function	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Syntax rules for escape clause

This ODBC function converts any escape clauses in the specified SQL statement to a format that can be executed by HADB, and then returns the SQL statement. The following are the syntax rules for an escape clause:

```
escape-clause ::= escape-sequence-for-date-or-time-or-time-stamp
                | escape-sequence-for-escape-character-in-LIKE-predicate
                | escape-sequence-for-outer-join
                | scalar-function-escape-sequence
escape-sequence-for-date-or-time-or-time-stamp ::= date-escape-sequence
                                                 | time-escape-sequence
                                                 | time-stamp-escape-sequence
date-escape-sequence ::=
          escape-start-code d predefined-character-string-representation-of-date-data
 #1 escape-end-code
time-escape-sequence ::=
          escape-start-code t predefined-character-string-representation-of-time-data
 #2 escape-end-code
time-stamp-escape-sequence ::=
          escape-start-code ts predefined-character-string-representation-of-time-sta
mp-data #3escape-end-code
escape-sequence-for-escape-character-in-LIKE-predicate ::=
          escape-start-code escape escape-character escape-end-code
escape-sequence-for-outer-join ::= escape-start-code oj joined-table escape-end-code
```

```
scalar-function-escape-sequence ::= escape-start-code fn scalar-function escape-end-c
ode
scalar-function ::= scalar-function-in-default-format #4
escape-start-code ::= '{'
escape-end-code ::= '}'
```

#1

Character string representation '*YYYY-MM-DD*'

#2

Character string representation '*hh*:*mm*:*ss*[.*nn...n*]' *nn...n* is the fractional seconds precision with a length of *p* digits. *n* is a number from 0 to 9, and *p* is 0, 3, 6, 9, or 12.

#3

Character string representation '*YYYY-MM-DD hh*:*mm*:*ss*[.*nn...n*]' *nn...n* is the fractional seconds precision with a length of *p* digits. *n* is a number from 0 to 9, and *p* is 0, 3, 6, 9, or 12.

#4

For details about the scalar function in the default format, see Table 16-7: Conversion formats of scalar functions whose default format differs from the HADB format.

Note that an escape clause cannot be specified in an underlined part. Because the ODBC driver does not perform syntax analysis on the underlined parts, these parts will remain the same after conversion and will be subject to syntax analysis by the HADB server.

The following keywords can be used in escape sequences. These keywords are not case sensitive.

1. `d` in a date escape sequence
2. `t` in a time escape sequence
3. `ts` in a time stamp escape sequence
4. `escape` in an escape sequence of an escape character of a LIKE predicate
5. `oj` in an outer join escape sequence
6. `fn` in a scalar function escape sequence

The escape clause entry rules are as follows:

- The space can be used as the delimiter character in an escape clause.
- The delimiter can be inserted following an escape start code, following a keyword, and before an escape end code.
- You can specify multiple escape clauses in a single SQL statement.
- The HADB ODBC driver converts the escape clauses in an SQL statement to a format that can be executed by HADB. Note that only the part of each escape clause that is enclosed in curly brackets is converted. The driver converts nothing outside the escape clauses.

The following table shows the escape clause conversion rules.

Table 16-6: Escape clause conversion rules

Escape clause	Before conversion	After conversion
Date	*escape-start-code* `d` *predefined-character-string-representation-of-date-data escape-end-code*	*predefined-character-string-representation-of-date-data*

Escape clause	Before conversion	After conversion
Time	*escape-start-code* `t` *predefined-character-string-representation-of-time-data escape-end-code*	*predefined-character-string-representation-of-time-data*
Time stamp	*escape-start-code* `ts` *predefined-character-string-representation-of-time-stamp-data escape-end-code*	*predefined-character-string-representation-of-time-stamp-data*
LIKE	*escape-start-code* `escape`*escape-character escape-end-code*	*escape escape-character*
Outer join	*escape-start-code* `oj` *joined-table escape-end-code*	*joined-table*
Scalar function	*escape-start-code* `fn` *scalar-function escape-end-code*	*scalar-function-in-HADB-format*#

#

The ODBC driver converts a scalar function in the default format to the HADB format.

The table below shows the conversion formats of scalar functions whose default format differs from the HADB server format.

In general, the ODBC driver does not check the number of arguments in a scalar function.

Table 16-7: Conversion formats of scalar functions whose default format differs from the HADB format

Scalar functions	Format before conversion	Format after conversion (HADB format)
Mathematical function	`CEILING(number)`	`CEIL(number)`
	`LOG(float)`	`LN(float)`
	`RAND([number, number])`	`RANDOM([number, number])`
	`TRUNCATE(number[, places])`	`TRUNC(number[, places])`
String function	`CHAR(code)`	`CHR(code)`
	`LCASE(string)`	`LOWER(string)`
	`OCTET_LENGTH(string)`	`LENGTHB(string)`
	`SUBSTRING(string, start[, length])`	`SUBSTR(string, start[, length])`
	`UCASE(string)`	`UPPER(string)`
Time and date functions	`CURRENT_DATE()`	`CURRENT_DATE`
	`CURRENT_TIME()`	`CURRENT_TIME`

16.8.4 SQLParamData

(1) Function

Used together with `SQLPutData`, the `SQLParamData` function sends parameter data when the SQL statement is executed or after `SQLExecute`, `SQLExecDirect`, or `SQLExecDirectW` has been executed.

(2) Format

```
SQLRETURN SQLParamData
(
  SQLHSTMT        StatementHandle,    /* In */
```

```
  SQLPOINTER     * ValuePtrPtr          /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`ValuePtrPtr`

Returns a pointer to the location at which the parameter data for the SQL statement is set.

A valid value is returned only when the function's return value is `SQL_NEED_DATA`.

This value is the same as the `ParameterValuePtr` value of `SQLBindParameter` or the `TargetValuePtr` value of `SQLBindCol` as well as the value specified in the `SQL_DESC_DATA_PTR` field of the descriptor record.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_NO_DATA`, `SQL_NEED_DATA`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		Y
07002	`COUNT` field incorrect	The number of parameters specified in `SQLBindParameter` does not match the number of dynamic parameters.	Y
07006	Restricted data type attribute violation	--	N
08003	Connection does not exist		Y
08S01	Communication link failure		N
22001	Character string data was right-truncated		Y
22003	Numeric value out of range		Y
22007	Invalid datetime format		Y
22008	Datetime field overflow		Y
22018	Invalid character value for cast specification		Y
22026	Character string data length mismatch		N
40001	Serialization failure		N
40003	Statement completion unknown		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C036	Data conversion error	The contents of the requested input data are invalid.	Y

SQLSTATE	Description	Remarks	Returned
5C041	Unsupported data type error	The driver does not support the specified data type.	Y
HY000	General error	--	N
HY001	Memory allocation error		Y
HY003	Invalid C data type		Y
HY004	Invalid SQL data type		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	NULL is specified in ParameterCountPtr.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		Y
HY014	Invalid precision or scale value		Y
HY090	Invalid string or buffer length		Y
HY104	Invalid precision or scale value		Y
HY117	Connection suspended		N
HYC00	Optional feature not implemented		Y
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.8.5 SQLPutData

(1) Function

This ODBC function sends parameter data to the HADB ODBC driver when the SQL statement is executed.

(2) Format

```
SQLRETURN SQLPutData
(
  SQLHSTMT          StatementHandle,    /* In */
  SQLPOINTER        DataPtr,            /* In */
  SQLLEN            StrLen_or_Ind       /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`DataPtr`

Specifies a pointer to the buffer containing the actual data for the parameters. The data must be in the C data type specified in `ValueType` of `SQLBindParameter`.

`StrLen_or_Ind`

- When `SQL_C_CHAR` or `SQL_C_BINARY` is specified for the C data type in `SQLBindParameter`
  Specify the length of `*DataPtr`, `SQL_NTS`, or `SQL_NULL_DATA`.
- When any other C data type is specified in `SQLBindParameter`
  Specify `SQL_NULL_DATA`. If any other value is specified, this argument is ignored and the HADB ODBC driver assumes that the size of the `*DataPtr` buffer is the C data type specified in `ValueType` of `SQLBindParameter`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
07006	Restricted data type attribute violation		N
07S01	Invalid use of default parameter		N
08S01	Communication link failure		N
22001	Character string data was right-truncated		N
22003	Numeric value out of range		N
22007	Invalid datetime format		N
22008	Datetime field overflow		N
22012	Division by zero		N
22015	Interval field overflow		N
22018	Invalid character value for cast specification		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	A null pointer is specified for `DataPtr`, but the value specified in `StrLen_or_Ind` is neither `0` nor `SQL_NULL_DATA`.	Y

SQLSTATE	Description	Remarks	Returned
HY010	Function sequence error	A data parameter required for execution has not been obtained by SQLParamData.	Y
HY013	Memory management error	--	N
HY019	Non-character or non-binary data was segmented and sent separately		N
HY020	Attempt to concatenate a null value		N
HY090	Invalid string or buffer length	The following conditions are all satisfied: • The value specified in DataPtr is not a null pointer. • The value specified in StrLen_or_Ind is less than or equal to 0. • The value specified in StrLen_or_Ind is neither SQL_NTS nor SQL_NULL_DATA.	Y
HY117	Connection suspended	--	N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

This ODBC function does not support the following functions:

- Data correspondence for columns
- Segmented transmission

16.9 Acquiring execution results and execution result information

This section explains the ODBC functions that are used to acquire execution results and execution result information.

16.9.1 SQLRowCount

(1) Function

This ODBC function returns the number of rows changed by the following processing before this function was executed:

- Various SQL statements (`UPDATE`, `INSERT`, and `DELETE` statements)

(2) Format

```
SQLRETURN SQLRowCount
(
  SQLHSTMT       StatementHandle,     /* In */
  SQLLEN       * RowCountPtr          /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`RowCountPtr`

Specifies a pointer to the buffer in which the number of changed rows is to be returned.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
HY000	General error		N
HY001	Memory allocation error		N
HY009	Invalid use of null pointer	An invalid value was set in `RowCountPtr`.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY117	Connection suspended		N
HYT01	Connection timeout expired		N

SQLSTATE	Description	Remarks	Returned
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

If the executed SQL statement is UPDATE, INSERT, or DELETE, this function returns the number of rows changed by the request. If the returned value is -1, the possible causes are as follows:

- Overflow occurred on any changed rows.
- The executed SQL statement was not UPDATE, INSERT, or DELETE.

16.9.2 SQLNumResultCols

(1) Function

This ODBC function returns the number of columns in a result set of a prepared SQL statement.

(2) Format

```
SQLRETURN SQLNumResultCols
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLSMALLINT    * ColumnCountPtr       /* Out */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

ColumnCountPtr

Specifies a pointer to the buffer in which the number of columns in a result set is to be returned.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	NULL is specified in ColumnCountPtr.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY117	Connection suspended		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.3 SQLDescribeCol, SQLDescribeColW

(1) Function

This ODBC function returns information about a column in the result set that is obtained by executing SQLPrepare. This information is also set in the fields of the IRD.

(2) Format

- For SQLDescribeCol

```
SQLRETURN SQLDescribeCol
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLUSMALLINT     ColumnNumber,        /* In */
  SQLCHAR        * ColumnName,          /* Out */
  SQLSMALLINT      BufferLength,        /* In */
  SQLSMALLINT    * NameLengthPtr,       /* Out */
  SQLSMALLINT    * DataTypePtr,         /* Out */
  SQLULEN        * ColumnSizePtr,       /* Out */
  SQLSMALLINT    * DecimalDigitsPtr,    /* Out */
  SQLSMALLINT    * NullablePtr          /* Out */
)
```

- For SQLDescribeColW

```
SQLRETURN SQLDescribeColW
(
  SQLHSTMT          StatementHandle,      /* In */
  SQLUSMALLINT      ColumnNumber,         /* In */
  SQLWCHAR        * ColumnName,           /* Out */
  SQLSMALLINT       BufferLength,         /* In */
  SQLSMALLINT     * NameLengthPtr,        /* Out */
  SQLSMALLINT     * DataTypePtr,          /* Out */
  SQLULEN         * ColumnSizePtr,        /* Out */
  SQLSMALLINT     * DecimalDigitsPtr,     /* Out */
  SQLSMALLINT     * NullablePtr           /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`ColumnNumber`

Specifies a column number in result data.

`ColumnName`

Specifies a pointer to the buffer in which the column name is to be returned. For details about the names of retrieval result columns, see *Rules* in *Specification format and rules for the SELECT statement* in the manual *HADB SQL Reference*.

`BufferLength`

Specifies the length# of the `*ColumnName` buffer. This length includes the null terminating character. `SQL_NTS` cannot be specified.

`NameLengthPtr`

Specifies a pointer to the buffer that stores the total valid length# of the value that is set in `*ColumnName`. This length does not include the null terminating character.

Important

If the length# of the `*ColumnName` column name stored here is greater than the value of `BufferLength` without the length# of the null terminating character, the character string stored in `ColumnName` is truncated to the length# equivalent to `BufferLength` without the null terminating character, and then the null terminating character is added at the end.

`DataTypePtr`

Specifies a pointer to the buffer in which the SQL data type of the column is to be returned. If the data type is unknown, the driver returns `SQL_C_DEFAULT`.

`ColumnSizePtr`

Specifies a pointer to the buffer in which the size of the column at the data source is to be returned. If the size of the column is unknown, the driver returns `0`.

`DecimalDigitsPtr`

Specifies a pointer to the buffer in which the number of decimal places of the column at the data source is to be returned. If the number of decimal places is unknown or not applicable, the driver returns `0`.

NullablePtr

Specifies a pointer to the buffer in which a value indicating whether the column allows NULL values is to be returned. One of the following values is returned:

- SQL_NO_NULLS

  The column allows NULL values.

- SQL_NULLABLE

  The column does not allow NULL values.

#

The length must be in bytes for SQLDescribeCol and in characters for SQLDescribeColW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
07005	Prepared statement is not a cursor-specification	--	Y
07009	Invalid descriptor index		Y
08S01	Communication link failure		N
24000	Invalid cursor status		Y
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	An invalid value (SQL_NTS) was specified in BufferLength.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.4 SQLColAttribute, SQLColAttributeW

(1) Function

This ODBC function returns descriptor information for a column in a result set.

(2) Format

- For `SQLColAttribute`

```
SQLRETURN SQLColAttribute
(
  SQLHSTMT          StatementHandle,         /* In */
  SQLUSMALLINT      ColumnNumber,            /* In */
  SQLUSMALLINT      FieldIdentifier,         /* In */
  SQLPOINTER        CharacterAttributePtr,   /* Out */
  SQLSMALLINT       BufferLength,            /* In */
  SQLSMALLINT     * StringLengthPtr,         /* Out */
  SQLLEN          * NumericAttributePtr      /* Out */
)
```

- For `SQLColAttributeW`

```
SQLRETURN SQLColAttributeW
(
  SQLHSTMT          StatementHandle,         /* In */
  SQLUSMALLINT      ColumnNumber,            /* In */
  SQLUSMALLINT      FieldIdentifier,         /* In */
  SQLPOINTER        CharacterAttributePtr,   /* Out */
  SQLSMALLINT       BufferLength,            /* In */
  SQLSMALLINT     * StringLengthPtr,         /* Out */
  SQLLEN          * NumericAttributePtr      /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`ColumnNumber`

Specifies the record number in the IRD from which the field value is to be obtained. This number is assigned sequentially in ascending order of the columns, beginning with 1. The number corresponds to the column number of the result set.

`FieldIdentifier`

Specifies an identifier that corresponds to the descriptor field of the IRD that is to be obtained.

The following table lists the field identifiers that can be specified.

Table 16-8: Field identifiers that can be specified in FieldIdentifier of SQLColAttribute and SQLColAttributeW

No.	Field identifier
1	SQL_COLUMN_LENGTH

No.	Field identifier
2	SQL_COLUMN_PRECISION
3	SQL_COLUMN_SCALE
4	SQL_DESC_AUTO_UNIQUE_VALUE
5	SQL_DESC_BASE_COLUMN_NAME
6	SQL_DESC_BASE_TABLE_NAME
7	SQL_DESC_CASE_SENSITIVE
8	SQL_DESC_CATALOG_NAME
9	SQL_DESC_CONCISE_TYPE
10	SQL_DESC_COUNT
11	SQL_DESC_DISPLAY_SIZE
12	SQL_DESC_FIXED_PREC_SCALE
13	SQL_DESC_LABEL
14	SQL_DESC_LENGTH
15	SQL_DESC_LITERAL_PREFIX
16	SQL_DESC_LITERAL_SUFFIX
17	SQL_DESC_LOCAL_TYPE_NAME
18	SQL_DESC_NAME
19	SQL_DESC_NULLABLE
20	SQL_DESC_NUM_PREC_RADIX
21	SQL_DESC_OCTET_LENGTH
22	SQL_DESC_PRECISION
23	SQL_DESC_SCALE
24	SQL_DESC_SCHEMA_NAME
25	SQL_DESC_SEARCHABLE
26	SQL_DESC_TABLE_NAME
27	SQL_DESC_TYPE
28	SQL_DESC_TYPE_NAME
29	SQL_DESC_UNNAMED
30	SQL_DESC_UNSIGNED
31	SQL_DESC_UPDATABLE

`CharacterAttributePtr`

Specifies a pointer to the buffer in which the descriptor field value of the IRD that corresponds to `FieldIdentifier` is to be returned. If the descriptor field value is not a character string, this argument is not used.

`BufferLength`

Specifies the length of `*CharacterAttributePtr` (in bytes). `SQL_NTS` cannot be specified.

StringLengthPtr

Specifies a pointer to the buffer in which is to be returned the total number of valid bytes (excluding the null terminating character) to be returned to *CharacterAttributePtr.

NumericAttributePtr

Specifies a pointer to the integer buffer in which the descriptor field value of the IRD that corresponds to FieldIdentifier is to be returned. If the descriptor field value is not a numeric value, this argument is not used.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	Not all of the character string value could be stored because the *CharacterAttributePtr buffer was too small (the character string value was truncated). The length of the untruncated character string value is stored in *StringLengthPtr. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
07005	Prepared statement is not a cursor-specification.	The statement associated with StatementHandle did not return a result set.	Y
07009	Invalid descriptor index	• The value specified in ColumnNumber is 0. • The value specified in ColumnNumber is greater than the number of columns in the result set.	Y
24000	Invalid cursor status	--	Y
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		N
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	• A character string pointer is specified in CharacterAttributePtr and the value specified in BufferLength is less than or equal to 0.	Y

SQLSTATE	Description	Remarks	Returned
		• An invalid value is specified in BufferLength (SQL_NTS).	
HY091	Invalid descriptor field identifier	The value specified in FieldIdentifier is neither a defined value nor an implementation-defined value.	Y
HYC00	Optional feature not implemented	The value specified in FieldIdentifier is not supported.	Y
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
IM001	Driver does not support this function	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.5 SQLBindCol

(1) Function

This ODBC function associates an application data area with a column in the result set.

(2) Format

```
SQLRETURN SQLBindCol
(
  SQLHSTMT         StatementHandle,         /* In */
  SQLUSMALLINT     ColumnNumber,            /* In */
  SQLSMALLINT      TargetType,              /* In */
  SQLPOINTER       TargetValuePtr,          /* Out */
  SQLLEN           BufferLength,            /* In */
  SQLLEN         * StrLen_or_IndPtr         /* Out */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

ColumnNumber

Specifies the number of the result set column to be associated.

The column numbering begins with 1 because HADB does not support bookmarks.

TargetType

Specifies the C data type identifier of the area pointed to by TargetValuePtr or specifies SQL_C_DEFAULT. If SQL_C_DEFAULT is specified, the driver assumes the default C data type.

TargetValuePtr

Specifies a pointer to the area that is associated with the result set column.

To release the association, specify a null pointer.

BufferLength

Specifies the length (in bytes) of the area pointed to by TargetValuePtr.

StrLen_or_IndPtr

Specifies a pointer to the area in which the data length or indicator is returned by the HADB ODBC driver.

If the executed SQLFetch function returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the HADB ODBC driver returns the data length or indicator.

If the column data is a null value, the return value is SQL_NULL_DATA.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
07006	Restricted data type attribute violation	The value specified in ColumnNumber is 0, but the value specified in TargetType is neither SQL_C_BOOKMARK nor SQL_C_VARBOOKMARK.	N
07009	Invalid descriptor index	The value specified in ColumnNumber is greater than the maximum number of columns in the result set.	N
HY000	General error	--	N
HY001	Memory allocation error		Y
HY003	Invalid application buffer data type	The value specified in TargetType is neither a valid data type nor SQL_C_DEFAULT.	N
HY010	Function sequence error	The asynchronously executing function that was called for StatementHandle was still executing when this function was called.	N
HY013	Memory management error	The function call could not be processed because the memory object could not be accessed.	N
HY090	Invalid string or buffer length	The following conditions are both satisfied: • The value specified in TargetValuePtr is not a null pointer. • The value specified in BufferLength is less than 0.	Y

SQLSTATE	Description	Remarks	Returned
HYC00	Optional feature not implemented	Zero was specified in ColumnNumber, but the driver does not support bookmarks.	Y
HYT01	Connection timeout expired	A connection timeout occurred before the data source responded to the request. The connection timeout value can be specified by using SQL_ATTR_CONNECTION_TIMEOUT in SQLSetConnectAttr or SQLSetConnectAttrW.	N
IM001	Driver does not support this function	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

This ODBC function does not support the following functions:

- Binding offsets
- Binding arrays
- Row-wise binding

16.9.6 SQLFetch

(1) Function

This ODBC function fetches the next row set of data from the result set and returns data for all columns associated by SQLBindCol.

(2) Format

```
SQLRETURN SQLFetch
(
  SQLHSTMT          StatementHandle    /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated	Character string data other than spaces or binary data other than NULL was truncated from the character string or binary data returned to a column. If it was a character string value, it was right-truncated.	Y
01S01	Error in row	An error occurred while fetching one or more rows.	N
01S07	Fractional truncation	The fractional part of the numeric value was truncated. For time, time stamp, and interval data types that contain a time component, the fractional part of the time was truncated. The function returns SQL_SUCCESS_WITH_INFO.	Y
01S51	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns SQL_SUCCESS_WITH_INFO.	Y
07006	Restricted data type attribute violation	The data value of a column in the result set cannot be converted to the data type specified by TargetType in SQLBindCol.	Y
07009	Invalid descriptor index	--	Y
08S01	Communication link failure		N
22001	Character string data was right-truncated		N
22002	Required indicator variable not supplied	NULL data was fetched into a column that has a null pointer for one of the following: • StrLen_or_IndPtr set by SQLBindCol • SQL_DESC_INDICATOR_PTR set by SQLSetDescField, SQLSetDescFieldW, SQLSetDescRec, or SQLSetDescRecW	Y
22003	Numeric value out of range	The integer part of the numeric value (numeric value or character string) was deleted.	Y
22007	Invalid datetime format	A character column was bound to a date, time, or time stamp C structure, but the value in the column is an invalid date, time, or time stamp.	Y
22012	Division by zero	A value obtained from an arithmetic expression resulting in division by zero was returned.	N
22015	Interval field overflow	--	N
22018	Invalid character value for cast specification		Y
24000	Invalid cursor status	StatementHandle is in executed status, but no result set is associated.	Y

SQLSTATE	Description	Remarks	Returned
40001	Serialization failure	--	N
40003	Statement completion unknown		N
5C002	Character encoding conversion error		Y
5C037	Data format error		Y
5C038	Data conversion error	There is an error in the acquired result data or in the specification of the receiving area.	Y
HY000	General error	--	N
HY001	Memory allocation error	The memory required to execute or complete the function has not been allocated for the HADB ODBC driver.	N
HY003	Invalid application buffer data type	The C data type set by SQLBindCol is an invalid data type.	Y
HY008	Operation cancelled	--	N
HY009	Invalid use of null pointer		Y
HY010	Function sequence error		Y
HY013	Memory management error		Y
HY090	Invalid string or buffer length		Y
HY104	Invalid precision or scale value		Y
HY107	Row value out of range		N
HYC00	Optional feature not implemented	The HADB ODBC driver or the HADB server does not support the conversion specified by the combination of TargetType in SQLBindCol and the SQL data type of the corresponding column.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The driver does not support the functionality to return row statuses to a row status array.

16.9.7 SQLFetchScroll

(1) Function

This ODBC function fetches a specified row set of data from the result set and returns data for all bound columns.

Note that HADB returns SQL_ERROR when SQLSTATE is HYC00.

(2) Format

```
SQLRETURN SQLFetchScroll
(
  SQLHSTMT        StatementHandle,          /* In */
  SQLSMALLINT     FetchOrientation,         /* In */
  SQLLEN          FetchOffset               /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

FetchOrientation

This argument is ignored, if specified.

FetchOffset

This argument is ignored, if specified.

(4) Return value

This ODBC function returns SQL_ERROR or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
01S01	Error in row		N
01S06	An attempt was made to fetch before the result set returned the first row set		N
01S07	Fractional truncation		N
07006	Restricted data type attribute violation		N
07009	Invalid descriptor index		N
08S01	Communication link failure		N
22001	Character string data was right-truncated		N
22002	Required indicator variable not supplied		N
22003	Numeric value out of range		N
22007	Invalid datetime format		N
22012	Division by zero		N
22015	Interval field overflow		N

SQLSTATE	Description	Remarks	Returned
22018	Invalid character value for cast specification		N
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		N
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY106	Fetch type out of range		N
HY107	Row value out of range		N
HY111	Invalid bookmark value		N
HYC00	Optional feature not implemented	The driver returns SQL_ERROR.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.8 SQLGetData

(1) Function

This ODBC function acquires data for a single column in the result set.

(2) Format

```
SQLRETURN SQLGetData
(
  SQLHSTMT         StatementHandle,         /* In */
  SQLUSMALLINT     ColumnNumber,            /* In */
  SQLSMALLINT      TargetType,              /* In */
  SQLPOINTER       TargetValuePtr,          /* Out */
  SQLLEN           BufferLength,            /* In */
  SQLLEN         * StrLen_or_IndPtr         /* Out */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`ColumnNumber`

Specifies the number of the column whose data is to be acquired.

The column numbering begins with 1 because HADB does not support bookmarks.

`TargetType`

Specifies the C data type identifier of the area pointed to by `TargetValuePtr` or specifies `SQL_C_DEFAULT`. If `SQL_C_DEFAULT` is specified, the driver assumes the default C data type.

`TargetValuePtr`

Specifies a pointer to the area that is to receive the column's data.

`BufferLength`

Specifies the length (in bytes) of the area pointed to by `TargetValuePtr`.

`StrLen_or_IndPtr`

Specifies a pointer to the area in which the data length or indicator is returned by the HADB ODBC driver.

The HADB ODBC driver returns the data length of the indicator.

If the column data is a null value, the return value is `SQL_NULL_DATA`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_NO_DATA`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
`01000`	General warning	--	N
`01004`	Character string data was right-truncated	The buffer pointed to by the `TargetValuePtr` buffer is not large enough to store the data for the column specified by `ColumnNumber`. The function returns `SQL_SUCCESS_WITH_INFO`.	Y
`01S07`	Fractional truncation	The fractional part of the numeric value was truncated. For time, time stamp, and interval data types that contain a time component, the fractional part of the time was truncated. The function returns `SQL_SUCCESS_WITH_INFO`.	Y
`01S51`	Code replacement occurred during conversion of character encoding	A character code that cannot be converted was detected and then replaced with the specified character. The function returns `SQL_SUCCESS_WITH_INFO`.	Y
`07006`	Restricted data type attribute violation	A data value in the column in the result set cannot be converted to the C data type specified by `TargetType`.	Y

SQLSTATE	Description	Remarks	Returned
07009	Invalid descriptor index	The column number specified in ColumnNumber does not exist.	Y
08S01	Communication link failure	--	N
22002	Required indicator variable not supplied	StrLen_or_IndPtr is a null pointer and NULL data was acquired.	Y
22003	Numeric value out of range	The integer part of the numeric value (numeric value or character string) was deleted.	N
22007	Invalid datetime format	A character column was bound to a date, time, or time stamp C structure, but the value in the column is an invalid date, time, or time stamp.	Y
22012	Division by zero	A value obtained from an arithmetic expression resulting in division by zero was returned.	N
22015	Interval field overflow	--	N
22018	Invalid character value for cast specification		Y
24000	Invalid cursor status	A cursor was open on StatementHandle and SQLFetch was called, but the cursor was positioned before the beginning of the result set or after the end of the result set.	Y
5C002	Character encoding conversion error	--	Y
5C037	Data format error		Y
5C038	Data conversion error	There is an error in the acquired result data or in the specification of the receiving area.	Y
HY000	General error	--	N
HY001	Memory allocation error	The memory required to execute or complete the function has not been allocated for the HADB ODBC driver.	Y
HY003	Invalid application buffer data type	--	Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	Null pointers were specified for both TargetValuePtr and StrLen_or_IndPtr.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		Y
HY090	Invalid string or buffer length	The following conditions are both satisfied: • The value specified in TargetValuePtr is not a null pointer. • The value specified in BufferLength is less than 0.	Y
HY104	Invalid precision or scale value	--	Y
HY109	Invalid cursor position	--	N
HYC00	Optional feature not implemented	The driver does not support the conversion specified by the combination of TargetType and the SQL data type of the corresponding column.	Y

SQLSTATE	Description	Remarks	Returned
HYT00	Timeout expired	--	N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

This ODBC function does not support the following functions:

- Acquiring variable-length data in segments
- Limitations on the maximum length of data that can be returned

16.9.9 SQLSetPos

(1) Function

This ODBC function sets the cursor position in a row set.

Note that HADB returns SQL_ERROR when SQLSTATE is HYC00.

(2) Format

```
SQLRETURN SQLSetPos
(
  SQLHSTMT         StatementHandle,      /* In */
  SQLSETPOSIROW    RowNumber,            /* In */
  SQLUSMALLINT     Operation,            /* In */
  SQLUSMALLINT     LockType              /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

RowNumber

This argument is ignored, if specified.

Operation

This argument is ignored, if specified.

LockType

This argument is ignored, if specified.

(4) Return value

This ODBC function returns SQL_ERROR or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01001	Cursor operation conflict		N
01004	Character string data was right-truncated		N
01S01	Error in row		N
01S07	Fractional truncation		N
07006	Restricted data type attribute violation		N
07009	Invalid descriptor index		N
21S02	Degree of derived table does not match column list		N
22001	Character string data was right-truncated		N
22003	Numeric value out of range		N
22007	Invalid datetime format		N
22008	Datetime field overflow		N
22015	Interval field overflow		N
22018	Invalid character value for cast specification		N
23000	Integrity constraint violation		N
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
42000	Syntax error or access violation		N
44000	WITH CHECK OPTION violation		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		N
HY011	Attribute cannot be set now		N
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY092	Invalid attribute or option identifier		N
HY107	Row value out of range		N
HY109	Invalid cursor position		N
HYC00	Optional feature not implemented	The driver returns SQL_ERROR.	Y
HYT00	Timeout expired	--	N

SQLSTATE	Description	Remarks	Returned
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.10 SQLBulkOperations

(1) Function

This ODBC function performs batch insertions and batch bookmark operations using bookmarks. The batch bookmark processing includes update, delete, and fetch operations.

Note that HADB returns SQL_ERROR when SQLSTATE is HYC00.

(2) Format

```
SQLRETURN SQLBulkOperations
(
  SQLHSTMT        StatementHandle,         /* In */
  SQLSMALLINT     Operation                /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Operation

This argument is ignored, if specified.

(4) Return value

This ODBC function returns SQL_ERROR or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01004	Character string data was right-truncated		N
01S01	Error in row		N

SQLSTATE	Description	Remarks	Returned
01S07	Fractional truncation		N
07006	Restricted data type attribute violation		N
07009	Invalid descriptor index		N
21S02	Degree of derived table does not match column list		N
22001	Character string data was right-truncated		N
22003	Numeric value out of range		N
22007	Invalid datetime format		N
22008	Datetime field overflow		N
22015	Interval field overflow		N
22018	Invalid character value for cast specification		N
23000	Integrity constraint violation		N
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
42000	Syntax error or access violation		N
44000	WITH CHECK OPTION violation		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		N
HY011	Attribute cannot be set now		N
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY092	Invalid attribute or option identifier		N
HYC00	Optional feature not implemented	The driver returns SQL_ERROR.	Y
HYT00	Timeout expired	--	N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.11 SQLMoreResults

(1) Function

This ODBC function initializes results when the SELECT, UPDATE, INSERT, or DELETE statement is executed.

Note that HADB returns SQL_NO_DATA.

(2) Format

```
SQLRETURN SQLMoreResults
(
  SQLHSTMT          StatementHandle       /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

(4) Return value

This ODBC function returns SQL_NO_DATA, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01S02	Option value changed		N
08S01	Communication link failure		N
40001	Serialization failure		N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.9.12 SQLGetDiagField, SQLGetDiagFieldW

(1) Function

This ODBC function returns the current value of a field in a diagnostic data structure that contains such information as error, warning, and status.

(2) Format

- For `SQLGetDiagField`

```
SQLRETURN SQLGetDiagField
(
  SQLSMALLINT     HandleType,          /* In */
  SQLHANDLE       Handle,              /* In */
  SQLSMALLINT     RecNumber,           /* In */
  SQLSMALLINT     DiagIdentifier,      /* In */
  SQLPOINTER      DiagInfoPtr,         /* Out */
  SQLSMALLINT     BufferLength,        /* In */
  SQLSMALLINT   * StringLengthPtr      /* Out */
)
```

- For `SQLGetDiagFieldW`

```
SQLRETURN SQLGetDiagFieldW
(
  SQLSMALLINT     HandleType,          /* In */
  SQLHANDLE       Handle,              /* In */
  SQLSMALLINT     RecNumber,           /* In */
  SQLSMALLINT     DiagIdentifier,      /* In */
  SQLPOINTER      DiagInfoPtr,         /* Out */
  SQLSMALLINT     BufferLength,        /* In */
  SQLSMALLINT   * StringLengthPtr      /* Out */
)
```

(3) Arguments

`HandleType`

Specifies one of the following handle types:

- `SQL_HANDLE_ENV`: Environment handle
- `SQL_HANDLE_DBC`: Connection handle
- `SQL_HANDLE_STMT`: Statement handle
- `SQL_HANDLE_DESC`: Descriptor handle

`Handle`

Specifies a handle value.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`RecNumber`

Specifies the diagnostic information (status record) number from which the application is to acquire information.

If the application is to acquire the value of the diagnostic header field (the value indicating the diagnostic header field is to be set in DiagIdentifier), this argument is ignored.

If the application is to acquire any other value, specify 1 or a greater value in this argument.

DiagIdentifier

Specifies a required diagnostic field identifier. The two principal types are header fields and record fields. For details about the attributes that can be specified, see 16.18 Attributes that can be specified in DiagIdentifier of SQLGetDiagField and SQLGetDiagFieldW.

DiagInfoPtr

Specifies a pointer to the buffer in which the diagnostic information is to be returned. The data type depends on the value of DiagIdentifier.

BufferLength

Specifies the length of DiagInfoPtr.

This length includes the null terminating character.

Specify the following value depending on the type of DiagInfoPtr:

Type of DiagIdentifier value	Type of DiagInfoPtr value	Value to be specified in BufferLength
Value defined in 16.18 Attributes that can be specified in DiagIdentifier of SQLGetDiagField and SQLGetDiagFieldW	Character string or binary	Length of DiagInfoPtr (in bytes) SQL_NTS cannot be specified.
	Integer	None (ignored)

StringLengthPtr

Specify this argument only when DiagInfoPtr is character string data.

Specifies a pointer to the buffer that stores the total number of valid bytes to be returned to DiagInfoPtr. This total number of bytes does not include the number of bytes in the null terminating character.

Important

If the total length (in bytes) of the character string stored here that is to be set in DiagInfoPtr is greater than the value of BufferLength without the length of the null terminating character, the character string stored in DiagInfoPtr is truncated to the length equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, or SQL_NO_DATA. This function does not set SQLSTATE but provides the following execution results as return values:

Return value	Meaning
SQL_SUCCESS	The processing was successful.
SQL_SUCCESS_WITH_INFO	One of the following errors occurred: • The size of DiagInfoPtr was too small for the acquired value, resulting in truncation. You can determine the truncated size by comparing BufferLength and StringLengthPtr. • A character code that cannot be converted was detected during conversion of character encoding, and was replaced with the specified character.

Return value	Meaning
SQL_ERROR	One of the following errors occurred: • An invalid value was specified in DiagIdentifier. • One of the values listed below was specified in DiagIdentifier, but Handle was not a statement handle: • SQL_DIAG_CURSOR_ROW_COUNT • SQL_DIAG_DYNAMIC_FUNCTION • SQL_DIAG_DYNAMIC_FUNCTION_CODE • SQL_DIAG_ROW_COUNT • A value indicating a diagnostic record field was specified in DiagIdentifier, but the value specified in RecNumber was less than or equal to 0. • The data type of the requested field was character string, but BufferLength satisfied all the following conditions: • Less than 0 • Not SQL_NTS • Not the results of SQL_LEN_BINARY_ATTR(length) macro • An invalid value (SQL_NTS) was specified in BufferLength.
SQL_INVALID_HANDLE	The handle indicated by HandleType and Handle is not a valid handle.
SQL_NO_DATA	RecNumber is greater than the number of diagnostic records in the handle specified by Handle or there was no diagnostic record that could be read in the handle specified by Handle.

(5) SQLSTATE

This ODBC function does not return SQLSTATE.

The function returns details of errors by using the values defined in (4) Return value.

(6) Notes

- For details about the error information, see 15.4 Information that is returned in the event of an error.
- The function does not return diagnostic information.

16.9.13 SQLGetDiagRec, SQLGetDiagRecW

(1) Function

This ODBC function returns the current values of fields in a diagnostic data structure that is associated with a handle and includes such information as error, warning, and status.

(2) Format

- For SQLGetDiagRec

```
SQLRETURN SQLGetDiagRec
(
  SQLSMALLINT     HandleType,         /* In */
  SQLHANDLE       Handle,             /* In */
  SQLSMALLINT     RecNumber,          /* In */
  SQLCHAR       * SQLState,           /* Out */
```

```
  SQLINTEGER    * NativeErrorPtr,    /* Out */
  SQLCHAR       * MessageText,       /* Out */
  SQLSMALLINT     BufferLength,      /* In */
  SQLSMALLINT   * TextLengthPtr      /* Out */
)
```

- For SQLGetDiagRecW

```
SQLRETURN SQLGetDiagRecW
(
  SQLSMALLINT     HandleType,        /* In */
  SQLHANDLE       Handle,            /* In */
  SQLSMALLINT     RecNumber,         /* In */
  SQLWCHAR      * SQLState,          /* Out */
  SQLINTEGER    * NativeErrorPtr,    /* Out */
  SQLWCHAR      * MessageText,       /* Out */
  SQLSMALLINT     BufferLength,      /* In */
  SQLSMALLINT   * TextLengthPtr      /* Out */
)
```

(3) Arguments

HandleType

Specifies one of the following handle types:

- SQL_HANDLE_ENV: Environment handle
- SQL_HANDLE_DBC: Connection handle
- SQL_HANDLE_STMT: Statement handle
- SQL_HANDLE_DESC: Descriptor handle

Handle

Specifies a handle value.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

RecNumber

Specifies the diagnostic information (status record) number from which the application is to acquire information.

Specify 1 or a greater value in this argument.

SQLState

Specifies a pointer to the buffer in which the SQLSTATE code for the diagnostic record indicated by RecNumber is to be returned. This pointer consists of five characters consisting of 2 characters for the class + 3 characters for the subclass.

The information stored in the SQL_DIAG_SQLSTATE diagnostic field is returned.

If NULL is specified in this parameter, the driver sets nothing.

NativeErrorPtr

Specifies a pointer to the buffer in which the native error code specific to the data source is to be returned.

The information stored in the SQL_DIAG_NATIVE diagnostic field is returned.

If NULL is specified in this parameter, the driver sets nothing.

MessageText

Specifies a pointer to the buffer in which the diagnostic message text character string is to be returned.

The information stored in the SQL_DIAG_MESSAGE_TEXT diagnostic field is returned.

If NULL is specified in this parameter, the driver sets nothing.

BufferLength

Specifies the length# of the MessageText buffer.

Although there is no maximum length for a diagnostic message text, specify a value that is at least 512 bytes.

This length includes the null terminating character. SQL_NTS cannot be specified.

TextLengthPtr

Specifies a pointer to the buffer that returns the total valid length# to be returned to MessageText. This length does not include the null terminating character.

Important

If the length# stored here is greater than the value of BufferLength without the length# of the null terminating character, the character string stored in MessageText is truncated to the length# equivalent to BufferLength without the null terminating character, and then the null terminating character is added at the end.

#

The length must be in bytes for SQLGetDiagRec and in characters for SQLGetDiagRecW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, or SQL_NO_DATA. This function does not set SQLSTATE but provides the following execution results as return values:

Return value	Meaning
SQL_SUCCESS	The processing was successful.
SQL_SUCCESS_WITH_INFO	One of the following errors occurred: • The size of MessageText was too small for the diagnostic message text, resulting in truncation. You can determine the truncated size by comparing BufferLength and TextLengthPtr. • A character code that cannot be converted was detected during conversion of character encoding, and was replaced with the specified character.
SQL_ERROR	One of the following errors occurred: • The value specified in RecNumber is less than or equal to 0. • The value specified in BufferLength is less than 0. • An invalid value (SQL_NTS) was specified in BufferLength.
SQL_INVALID_HANDLE	The handle indicated by HandleType and Handle is not a valid handle.
SQL_NO_DATA	RecNumber is greater than the number of diagnostic records in the handle specified by Handle or there was no diagnostic record that could be read in the handle specified by Handle.

(5) SQLSTATE

This ODBC function does not return SQLSTATE.

The function returns details of errors by using the values defined in (4) Return value.

(6) Notes

- For details about the error information, see 15.4 Information that is returned in the event of an error.
- `SQLCODE` is set in `NativeErrorPtr`. For details about `SQLCODE`, see *Interpreting SQLCODEs* in the manual *HADB Messages*.
- The function does not return diagnostic information.

16.10 Acquiring system information for the data source

This section explains the ODBC functions that are used to acquire system information for the data source.

16.10.1 SQLColumnPrivileges, SQLColumnPrivilegesW

(1) Function

These functions return a list of privileges that are associated with the columns of the specified table. The result is output in the form of an SQL result set.

(2) Format

- For `SQLColumnPrivileges`

```
SQLRETURN SQLColumnPrivileges
(
  SQLHSTMT        StatementHandle,   /* In */
  SQLCHAR       * CatalogName,       /* In */
  SQLSMALLINT     NameLength1,       /* In */
  SQLCHAR       * SchemaName,        /* In */
  SQLSMALLINT     NameLength2,       /* In */
  SQLCHAR       * TableName,         /* In */
  SQLSMALLINT     NameLength3,       /* In */
  SQLCHAR       * ColumnName,        /* In */
  SQLSMALLINT     NameLength4        /* In */
)
```

- For `SQLColumnPrivilegesW`

```
SQLRETURN SQLColumnPrivilegesW
(
  SQLHSTMT        StatementHandle,   /* In */
  SQLWCHAR      * CatalogName,       /* In */
  SQLSMALLINT     NameLength1,       /* In */
  SQLWCHAR      * SchemaName,        /* In */
  SQLSMALLINT     NameLength2,       /* In */
  SQLWCHAR      * TableName,         /* In */
  SQLSMALLINT     NameLength3,       /* In */
  SQLWCHAR      * ColumnName,        /* In */
  SQLSMALLINT     NameLength4        /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`CatalogName`

Specifies a catalog name for the table.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify a null character string (`""`) or `NULL`.

NameLength1

Specifies either the length of *CatalogName[#1] or the keyword SQL_NTS.

The specified information is ignored, but specification itself is required. Therefore, to use this function, specify 0.

SchemaName

Specifies the schema name of a table. You can use a pattern character string[#2] to specify the schema name.

If a NULL pointer or empty string ("") only is specified for SchemaName, the function assumes that only the pattern character string '%' is specified for SchemaName.

NameLength2

Specifies either the length of *SchemaName[#1] or the keyword SQL_NTS.

If 0 is specified for NameLength2, the function assumes that only the pattern character string '%' is specified for SchemaName.

TableName

Specifies a table name. The table name can be specified by using a pattern character string[#2].

If a NULL pointer or empty string ("") only is specified for TableName, the function assumes that only the pattern character string '%' is specified for TableName.

NameLength3

Specifies either the length of *TableName[#1] or the keyword SQL_NTS.

If 0 is specified for NameLength3, the function assumes that only the pattern character string '%' is specified for TableName.

ColumnName

Specifies the column name. The column name can be specified by using a pattern character string[#2].

If a NULL pointer or empty string ("") only is specified for ColumnName, the function assumes that the pattern character string '%' is specified for ColumnName.

NameLength4

Specifies either the length of *ColumnName[#1] or the keyword SQL_NTS.

If 0 is specified for NameLength4, the function assumes that only the pattern character string '%' is specified for ColumnName.

#1

The length must be in bytes for SQLColumnPrivileges and in characters for SQLColumnPrivilegesW.

#2

For details about the special characters that can be specified in pattern character strings, see Table 16-10: Special characters that can be specified in pattern character strings.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table describes the format of the result set that is returned when this function is executed.

Table 16-9: Format of the result set that is returned

Column No.	Data type	Column name	Description
1	Varchar	TABLE_CAT	Catalog name (A null value is always returned.)
2	Varchar	TABLE_SCHEM	Schema name

Column No.	Data type	Column name	Description
3	Varchar	TABLE_NAME	Table name
4	Varchar	COLUMN_NAME	Column name
5	Varchar	GRANTOR	User who granted the access privilege
6	Varchar	GRANTEE	User whom the access privilege is granted to
7	Varchar	PRIVILEGE	The granted access privilege is returned. • SELECT: SELECT privilege • INSERT: INSERT privilege • UPDATE: UPDATE privilege • DELETE: DELETE privilege • TRUNCATE: TRUNCATE privilege • REFERENCES: REFERENCES privilege • IMPORT TABLE: IMPORT TABLE privilege • REBUILD INDEX: REBUILD INDEX privilege • GET COSTINFO: GET COSTINFO privilege • EXPORT TABLE: EXPORT TABLE privilege • MERGE CHUNK: MERGE CHUNK privilege • CHANGE CHUNK COMMENT: CHANGE CHUNK COMMENT privilege • CHANGE CHUNK STATUS: CHANGE CHUNK STATUS privilege • ARCHIVE CHUNK: ARCHIVE CHUNK privilege • UNARCHIVE CHUNK: UNARCHIVE CHUNK privilege
8	Varchar	IS_GRANTABLE	Whether the access privilege grantee (user whom the access privilege is granted to) can grant the access privilege to other users is returned. • YES: The access privilege can be granted to other users. • NO: The access privilege cannot be granted to other users.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y

SQLSTATE	Description	Remarks	Returned
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	The value of an argument that stores a name length either exceeds the maximum length of the corresponding name or is a negative value other than SQL_NTS.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.10.2 SQLColumns, SQLColumnsW

(1) Function

This ODBC function returns a listing of column information as a result set.

(2) Format

- For SQLColumns

```
SQLRETURN SQLColumns
(
  SQLHSTMT          StatementHandle,   /* In */
  SQLCHAR         * CatalogName,       /* In */
  SQLSMALLINT       NameLength1,       /* In */
  SQLCHAR         * SchemaName,        /* In */
  SQLSMALLINT       NameLength2,       /* In */
  SQLCHAR         * TableName,         /* In */
  SQLSMALLINT       NameLength3,       /* In */
  SQLCHAR         * ColumnName,        /* In */
  SQLSMALLINT       NameLength4        /* In */
)
```

- For SQLColumnsW

```
SQLRETURN SQLColumnsW
(
  SQLHSTMT          StatementHandle,   /* In */
  SQLWCHAR        * CatalogName,       /* In */
  SQLSMALLINT       NameLength1,       /* In */
  SQLWCHAR        * SchemaName,        /* In */
  SQLSMALLINT       NameLength2,       /* In */
  SQLWCHAR         * TableName,        /* In */
  SQLSMALLINT       NameLength3,       /* In */
  SQLWCHAR        * ColumnName,        /* In */
  SQLSMALLINT       NameLength4        /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`CatalogName`

Specifies that the specified information is to be used as a catalog name.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify a null character string (`""`) or `NULL`.

`NameLength1`

Specifies the length of `*CatalogName`[#1] or `SQL_NTS`.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify `0`.

`SchemaName`

Specifies a pattern character string[#2] for schema names. If `NULL` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string.

`NameLength2`

Specifies the length of `*SchemaName`[#1] or `SQL_NTS`.

If `0` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string in `SchemaName`.

`TableName`

Specifies a pattern character string[#2] for table names. If `NULL` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string.

`NameLength3`

Specifies the length of `*TableName`[#1] or `SQL_NTS`.

If `0` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string in `TableName`.

`ColumnName`

Specifies a pattern character string[#2] for column names. If `NULL` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string.

`NameLength4`

Specifies the length of `*ColumnName`[#1] or `SQL_NTS`.

If `0` is specified, the driver processing is the same as when `'%'` is specified as the pattern character string in `ColumnName`.

#1

The length must be in bytes for SQLColumns and in characters for SQLColumnsW.

#2

The following table lists the special characters that can be specified in pattern character strings.

Table 16-10: Special characters that can be specified in pattern character strings

Special character	Meaning
_ (underscore)	Any single character.
%	A character string of any length, including zero characters.
\	An escape character. A special character that immediately follows an escape character in a pattern character string is handled as a normal character. The character \ is represented by the Shift-JIS character code 0x5c (or 0x5c00 in UTF-16LE). In UTF-8, specify the character displayed as a backslash (\).

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table shows the format of the result set that is returned after execution.

Table 16-11: Format of the result set that is returned

Column No.	Type	Column name	Description
1	Varchar	TABLE_CAT	The function always returns a null value.
2	Varchar	TABLE_SCHEM	Schema name
3	Varchar	TABLE_NAME	Table name
4	Varchar	COLUMN_NAME	Column name
5	smallint	DATA_TYPE	ODBC SQL data type identifier
6	Varchar	TYPE_NAME	Type name
7	Integer	COLUMN_SIZE	Column size
8	Integer	BUFFER_LENGTH	Column data definition length
9	Smallint	DECIMAL_DIGITS	Number of decimal places
10	Smallint	NUM_PREC_RADIX	Cardinal number • Exact numeric value: 10 • Nonnumeric value: Null value
11	Smallint	NULLABLE	Returns a value indicating whether a null value can be used for this type. • SQL_NO_NULLS: Null values might not be permitted. • SQL_NULLABLE: Null values are permitted.
12	Varchar	REMARKS	The function always returns a null value.
13	Varchar	COLUMN_DEF	The function returns the default value of a column. • When the returned value is enclosed by single quotation marks ('), it means the default value of the column is a character string.

Column No.	Type	Column name	Description
			• If NULL is specified as the default value of the column, the character string NULL is returned without single quotation marks ('). • If no default value is specified for the column, the function returns a null value. • For details about other return values, see the description of the DEFAULT_VALUE column under *Content of SQL_COLUMNS* in the *HADB Setup and Operation Guide*. If no default value has been specified, the function returns NULL.
14	Smallint	SQL_DATA_TYPE	ODBC SQL data type identifier
15	Smallint	SQL_DATETIME_SUB	Subcode of ODBC SQL data type identifier
16	Integer	CHAR_OCTET_LENGTH	Maximum length (in bytes) of a column whose data type is character string data
17	Integer	ORDINAL_POSITION	Column number • Begins with 1.
18	Varchar	IS_NULLABLE	Returns a value indicating whether a null value can be used for this type: • YES: Null values might be permitted. • NO: Null values are not permitted.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	N
HY000	General error	--	N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N

SQLSTATE	Description	Remarks	Returned
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The column information that can be acquired depends on the privileges of the HADB user who executes this function. For details about the privileges and the column information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.10.3 SQLForeignKeys, SQLForeignKeysW

(1) Function

This ODBC function returns column information for the following foreign keys as an SQL result set:

- List of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables)
- List of foreign keys in other tables that refer to the primary key in the specified table

(2) Format

- For SQLForeignKeys

```
SQLRETURN SQLForeignKeys
(
  SQLHSTMT         StatementHandle,     /* In */
  SQLCHAR        * PKCatalogName,       /* In */
  SQLSMALLINT      NameLength1,         /* In */
  SQLCHAR        * PKSchemaName,        /* In */
  SQLSMALLINT      NameLength2,         /* In */
  SQLCHAR        * PKTableName,         /* In */
  SQLSMALLINT      NameLength3,         /* In */
  SQLCHAR        * FKCatalogName,       /* In */
  SQLSMALLINT      NameLength4,         /* In */
  SQLCHAR        * FKSchemaName,        /* In */
  SQLSMALLINT      NameLength5,         /* In */
```

```
  SQLCHAR       * FKTableName,         /* In */
  SQLSMALLINT     NameLength6          /* In */
)
```

- For SQLForeignKeysW

```
SQLRETURN SQLForeignKeysW
(
  SQLHSTMT        StatementHandle,     /* In */
  SQLWCHAR      * PKCatalogName,       /* In */
  SQLSMALLINT     NameLength1,         /* In */
  SQLWCHAR      * PKSchemaName,        /* In */
  SQLSMALLINT     NameLength2,         /* In */
  SQLWCHAR      * PKTableName,         /* In */
  SQLSMALLINT     NameLength3,         /* In */
  SQLWCHAR      * FKCatalogName,       /* In */
  SQLSMALLINT     NameLength4,         /* In */
  SQLWCHAR      * FKSchemaName,        /* In */
  SQLSMALLINT     NameLength5,         /* In */
  SQLWCHAR      * FKTableName,         /* In */
  SQLSMALLINT     NameLength6          /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

When you use this function, specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

PKCatalogName

Specifies a primary key table catalog name. This argument is ignored, if specified. When you use this function, specify a null character string ("") or NULL.

NameLength1

Specifies the length of *PKCatalogName# or SQL_NTS. This argument is ignored, if specified. When you use this function, specify 0.

PKSchemaName

Specifies a primary key table schema name. If NULL is specified, all schema names are processed.

NameLength2

Specifies the length of *PKSchemaName# or SQL_NTS. If 0 is specified, all schema names are processed.

PKTableName

Specifies a primary key table name. For details about the value to be specified, see Table 16-12: Combinations of PKTableName and FKTableName values and the result set that is returned based on the combination. If a null character string is specified, the number of rows in the result set will be zero.

NameLength3

Specifies the length of *PKTableName# or SQL_NTS. If NULL is specified for PKTableName, this argument is ignored. If a non-null value is specified for PKTableName and 0 is specified for this argument, the result will be the same as when a null character string is specified for PKTableName.

FKCatalogName

Specifies a foreign key table catalog name. This argument is ignored, if specified. When you use this function, specify a null character string ("") or NULL.

NameLength4

Specifies the length of *FKCatalogName# or SQL_NTS. This argument is ignored, if specified. When you use this function, specify 0.

FKSchemaName

Specifies a foreign key table schema name. If NULL is specified, all schema names are processed.

NameLength5

Specifies the length of *FKSchemaName# or SQL_NTS. If 0 is specified, all schema names are processed.

FKTableName

Specifies a foreign key table name. For details about the value to be specified, see Table 16-12: Combinations of PKTableName and FKTableName values and the result set that is returned based on the combination. If a null character string is specified, the number of rows in the result set will be zero.

NameLength6

Specifies the length of *FKTableName# or SQL_NTS. If NULL is specified for PKTableName, this argument is ignored. If a value other than NULL is specified for PKTableName and 0 is specified for this argument, the result will be the same as when an empty string is specified for PKTableName.

#

The length must be in bytes for SQLForeignKeys and in characters for SQLForeignKeysW.

The following table shows the combinations of PKTableName and FKTableName values and the result set that is returned based on the combination.

Table 16-12: Combinations of PKTableName and FKTableName values and the result set that is returned based on the combination

PKTableName	FKTableName	Result set that is returned
NULL	NULL	-- (error)
Not NULL	NULL	Returns information about the primary key of the table specified for PKTableName and the foreign keys of other tables that reference that primary key. The ODBC function does not return information about the foreign keys of other tables that reference keys only for the unique constraint in the table specified for PKTableName.
NULL	Not NULL	Returns information about the foreign key of the table specified for FKTableName and the primary keys of other tables that the foreign key references. The ODBC function does not return information about the foreign key of the table specified for FKTableName that references keys only for the unique constraint in other tables.
Not NULL	Not NULL	Returns information about the foreign key of the table specified for FKTableName and the primary key of the table that the foreign key references. This foreign key references only the primary key of the table specified for PKTableName.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

A result set is created when SQLForeignKeys or SQLForeignKeysW is executed. The following table shows the format of the result set that is returned.

Table 16-13: Format of the result set that is returned

Column No.	Type	Column name	Description
1	Varchar	PKTABLE_CAT	The null value is always returned.
2	Varchar	PKTABLE_SCHEM	Schema name of the primary key. If there is no schema name, a null character string is returned.
3	Varchar	PKTABLE_NAME	Table name of the primary key.
4	Varchar	PKCOLUMN_NAME	Column name of the primary key. If there is no column name, a null character string is returned.
5	Varchar	FKTABLE_CAT	The null value is always returned.
6	Varchar	FKTABLE_SCHEM	Schema name of the foreign key. If there is no schema name, a null character string is returned.
7	Varchar	FKTABLE_NAME	Table name of the foreign key.
8	Varchar	FKCOLUMN_NAME	Column name of the foreign key. If there is no column name, a null character string is returned.
9	Smallint	KEY_SEQ	Sequence number of the foreign key columns beginning at 1.
10	Smallint	UPDATE_RULE	Action to be applied to the foreign key when an SQL statement that performs update processing is requested. • SQL_NO_ACTION If no primary key value corresponds to the foreign key value because the primary key value or the foreign key value was updated, the key value cannot be updated. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, the primary key value or the foreign key value can be updated.
11	Smallint	DELETE_RULE	Action to be applied to the foreign key when an SQL statement that performs deletion processing is requested. • SQL_NO_ACTION If no primary key value corresponds to the foreign key value because rows were deleted from the table to be referenced, rows cannot be deleted. However, if referential constraint check suppression (DISABLE) is specified in the referential constraint definition in the CREATE TABLE statement, rows can be deleted.
12	Varchar	FK_NAME	Foreign key name.
13	Varchar	PK_NAME	Primary key name.
14	Smallint	DEFERRABILITY	Value indicating whether constraint checking on the foreign key is to be delayed. • SQL_NOT_DEFERRABLE A constraint check is to be performed each time an SQL statement is executed. Nothing can be done to delay this constraint checking.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
HY000	General error	--	N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	PKTableName and FKTableName are both NULL.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The information about foreign keys that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between privileges and the information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.10.4 SQLPrimaryKeys, SQLPrimaryKeysW

(1) Function

This ODBC function returns the column names that make up the primary key for a table. The driver returns the information as a result set for the specified SQL statement.

This function does not support returning primary keys from multiple tables in a single call.

(2) Format

- For `SQLPrimaryKeys`

```
SQLRETURN SQLPrimaryKeys
(
  SQLHSTMT       StatementHandle,     /* In */
  SQLCHAR      * CatalogName,         /* In */
  SQLSMALLINT    NameLength1,         /* In */
  SQLCHAR      * SchemaName,          /* In */
  SQLSMALLINT    NameLength2,         /* In */
  SQLCHAR      * TableName,           /* In */
  SQLSMALLINT    NameLength3          /* In */
)
```

- For `SQLPrimaryKeysW`

```
SQLRETURN SQLPrimaryKeysW
(
  SQLHSTMT       StatementHandle,     /* In */
  SQLWCHAR     * CatalogName,         /* In */
  SQLSMALLINT    NameLength1,         /* In */
  SQLWCHAR     * SchemaName,          /* In */
  SQLSMALLINT    NameLength2,         /* In */
  SQLWCHAR     * TableName,           /* In */
  SQLSMALLINT    NameLength3          /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`CatalogName`

Specifies that the specified information is to be used as a catalog name.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify a null character string (`""`) or `NULL`.

`NameLength1`

Specifies the length of `*CatalogName`[#] or `SQL_NTS`.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify `0`.

`SchemaName`

Specifies a schema name. If `NULL` is specified, all schema names are processed.

NameLength2

Specifies the length of *SchemaName# or SQL_NTS.

If 0 is specified, SchemaName is not used for filtering.

TableName

Specifies a table name. If NULL is specified, all table names are processed.

NameLength3

Specifies the length of *TableName# or SQL_NTS.

If 0 is specified, the function assumes that NULL is specified for TableName.

#

The length must be in bytes for SQLPrimaryKeys and in characters for SQLPrimaryKeysW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

A result set is created when SQLPrimaryKeys or SQLPrimaryKeysW is executed. The following table shows the format of the result set that is returned.

Table 16-14: Format of the result set that is returned

Column No.	Type	Column name	Description
1	Varchar	TABLE_CAT	The null value is always returned.
2	Varchar	TABLE_SCHEM	Schema name
3	Varchar	TABLE_NAME	Table name
4	Varchar	COLUMN_NAME	Column name
5	Smallint	KEY_SEQ	Column's order number within the primary key
6	Varchar	PK_NAME	Primary key name

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N

SQLSTATE	Description	Remarks	Returned
HY009	Invalid use of null pointer	TableName is a null pointer.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The primary key information that can be acquired depends on the privileges of the HADB user who executes this function. For details about the privileges and the primary key information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.10.5 SQLProcedureColumns, SQLProcedureColumnsW

(1) Function

This ODBC function returns a list of input and output parameters and a list of columns that make up the result set for a specified procedure. The driver returns the information as a result set for the specified SQL statement.

Note that the number of rows in the retrieval result set is always 0 because the driver does not support procedures.

(2) Format

- For SQLProcedureColumns

```
SQLRETURN SQLProcedureColumns
(
  SQLHSTMT        StatementHandle,     /* In */
  SQLCHAR       * CatalogName,         /* In */
  SQLSMALLINT     NameLength1,         /* In */
  SQLCHAR       * SchemaName,          /* In */
  SQLSMALLINT     NameLength2,         /* In */
```

```
  SQLCHAR       * ProcName,           /* In */
  SQLSMALLINT     NameLength3,        /* In */
  SQLCHAR       * ColumnName,         /* In */
  SQLSMALLINT     NameLength4         /* In */
)
```

- For SQLProcedureColumnsW

```
SQLRETURN SQLProcedureColumnsW
(
  SQLHSTMT        StatementHandle,    /* In */
  SQLWCHAR      * CatalogName,        /* In */
  SQLSMALLINT     NameLength1,        /* In */
  SQLWCHAR      * SchemaName,         /* In */
  SQLSMALLINT     NameLength2,        /* In */
  SQLWCHAR      * ProcName,           /* In */
  SQLSMALLINT     NameLength3,        /* In */
  SQLWCHAR      * ColumnName,         /* In */
  SQLSMALLINT     NameLength4         /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

CatalogName

Specifies a procedure catalog name.

NameLength1

Specifies the length# of *CatalogName.

SchemaName

Specifies a pattern character string for procedure schema names.

NameLength2

Specifies the length# of *SchemaName.

ProcName

Specifies a pattern character string for procedure names.

NameLength3

Specifies the length# of *ProcName.

ColumnName

Specifies a pattern character string for column names.

NameLength4

Specifies the length# of *ColumnName.

#

The length must be in bytes for SQLProcedureColumns and in characters for SQLProcedureColumnsW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY090	Invalid string or buffer length		N
HY117	Connection suspended		N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.10.6 SQLProcedures, SQLProceduresW

(1) Function

This ODBC function returns a list of procedures stored at the data source. The driver returns the information as a result set for the specified SQL statement.

Note that the number of rows in the retrieval result set is always 0 because the driver does not support procedures.

(2) Format

- For SQLProcedures

```
SQLRETURN SQLProcedures
(
  SQLHSTMT       StatementHandle,   /* In */
  SQLCHAR      * CatalogName,       /* In */
  SQLSMALLINT    NameLength1,       /* In */
  SQLCHAR      * SchemaName,        /* In */
  SQLSMALLINT    NameLength2,       /* In */
  SQLCHAR      * ProcName,          /* In */
  SQLSMALLINT    NameLength3        /* In */
)
```

- For SQLProceduresW

```
SQLRETURN SQLProceduresW
(
  SQLHSTMT       StatementHandle,   /* In */
  SQLWCHAR     * CatalogName,       /* In */
  SQLSMALLINT    NameLength1,       /* In */
  SQLWCHAR     * SchemaName,        /* In */
  SQLSMALLINT    NameLength2,       /* In */
  SQLWCHAR     * ProcName,          /* In */
  SQLSMALLINT    NameLength3        /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

CatalogName

Specifies a procedure catalog name.

NameLength1

Specifies the length# of *CatalogName.

SchemaName

Specifies a pattern character string for procedure schema names.

NameLength2

Specifies the length# of *SchemaName.

ProcName

Specifies a pattern character string for procedure names.

NameLength3

Specifies the length# of *ProcName.

#

The length must be in bytes for SQLProcedures and in characters for SQLProceduresW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY117	Connection suspended		N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.10.7 SQLSpecialColumns, SQLSpecialColumnsW

(1) Function

This ODBC function acquires one of the following types of information about columns in a specified table:

- Optimal set of columns that uniquely identifies a row in the table
- Columns that are updated automatically when a value in the row is updated by a transaction

The driver returns the information as a result set for the specified SQL statement.

Note that the number of rows in the retrieval result set is always 0 because the driver does not support the function that updates columns automatically.

(2) Format

- For SQLSpecialColumns

```
SQLRETURN SQLSpecialColumns
(
  SQLHSTMT       StatementHandle,    /* In */
  SQLUSMALLINT   IdentifierType,     /* In */
  SQLCHAR      * CatalogName,        /* In */
  SQLSMALLINT    NameLength1,        /* In */
  SQLCHAR      * SchemaName,         /* In */
  SQLSMALLINT    NameLength2,        /* In */
  SQLCHAR      * TableName,          /* In */
  SQLSMALLINT    NameLength3,        /* In */
  SQLUSMALLINT   Scope,              /* In */
  SQLUSMALLINT   Nullable            /* In */
)
```

- For SQLSpecialColumnsW

```
SQLRETURN SQLSpecialColumnsW
(
  SQLHSTMT       StatementHandle,    /* In */
  SQLUSMALLINT   IdentifierType,     /* In */
  SQLWCHAR     * CatalogName,        /* In */
  SQLSMALLINT    NameLength1,        /* In */
  SQLWCHAR     * SchemaName,         /* In */
  SQLSMALLINT    NameLength2,        /* In */
  SQLWCHAR     * TableName,          /* In */
  SQLSMALLINT    NameLength3,        /* In */
  SQLUSMALLINT   Scope,              /* In */
  SQLUSMALLINT   Nullable            /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

IdentifierType

Specifies one of the following values as the type of column to be returned:

IdentifierType	Description
SQL_BEST_ROWID	Returns the optimal column or set of columns that enables a row in the specified table to be uniquely identified by acquiring values from the columns. If the table contains an indexed column that enables a row to be uniquely identified, the driver returns that column. If the table does not contain such a column, the driver returns a temporary pseudo-column designed to identify rows.
SQL_ROWVER	Returns a column in the specified table, if any, that is automatically updated by the data source when a value in the row is updated by a transaction.

CatalogName

Specifies a catalog name for the table.

NameLength1

Specifies the length# of *CatalogName.

SchemaName

Specifies a schema name for the table.

NameLength2

Specifies the length# of *SchemaName.

TableName

Specifies a table name.

NameLength3

Specifies the length# of *TableName.

Scope

Specifies one of the following values as the minimum required scope of the row ID:

Scope	Description
SQL_SCOPE_CURROW	The row ID is valid only while it is positioned on that row.
SQL_SCOPE_TRANSACTION	The row ID is valid only for the duration of the current transaction.
SQL_SCOPE_SESSION	The row ID is valid for the duration of the session (across transaction boundaries).

Nullable

Specifies whether special columns that store null values are to be returned.

Specify one of the following values:

Nullable	Description
SQL_NO_NULLS	Excludes special columns that can store null values. Some drivers cannot support SQL_NO_NULLS; these drivers return an empty result set if SQL_NO_NULLS is specified. We recommend that applications take this into account and specify SQL_NO_NULLS only if it is required.
SQL_NULLABLE	Returns special columns even if they store null values.

#

The length must be in bytes for SQLSpecialColumns and in characters for SQLSpecialColumnsW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N

SQLSTATE	Description	Remarks	Returned
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length		N
HY097	Invalid value specified in IdentifierType		N
HY098	Invalid value specified in Scope		N
HY099	Invalid value specified in Nullable		N
HY117	Connection suspended		N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.10.8 SQLStatistics, SQLStatisticsW

(1) Function

This ODBC function returns as a result set for the specified SQL statement a listing of index information associated with a table.

(2) Format

- For SQLStatistics

```
SQLRETURN SQLStatistics
(
  SQLHSTMT          StatementHandle,     /* In */
  SQLCHAR         * CatalogName,         /* In */
  SQLSMALLINT       NameLength1,         /* In */
  SQLCHAR         * SchemaName,          /* In */
  SQLSMALLINT       NameLength2,         /* In */
```

```
  SQLCHAR         * TableName,          /* In */
  SQLSMALLINT       NameLength3,        /* In */
  SQLUSMALLINT      Unique,             /* In */
  SQLUSMALLINT      Reserved            /* In */
)
```

- For SQLStatisticsW

```
SQLRETURN SQLStatisticsW
(
  SQLHSTMT          StatementHandle,    /* In */
  SQLWCHAR        * CatalogName,        /* In */
  SQLSMALLINT       NameLength1,        /* In */
  SQLWCHAR        * SchemaName,         /* In */
  SQLSMALLINT       NameLength2,        /* In */
  SQLWCHAR        * TableName,          /* In */
  SQLSMALLINT       NameLength3,        /* In */
  SQLUSMALLINT      Unique,             /* In */
  SQLUSMALLINT      Reserved            /* In */
)
```

(3) Arguments

StatementHandle

Specifies a statement handle.

CatalogName

Specifies that the specified information is to be used as a catalog name.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify a null character string ("") or NULL.

NameLength1

Specifies the length of *CatalogName# or SQL_NTS. The specified information is ignored, but the specification itself is required.

Therefore, when you use this function, specify 0.

SchemaName

Specifies a schema name. If NULL is specified, all schema names are processed.

NameLength2

Specifies the length of *SchemaName# or SQL_NTS.

If 0 is specified, SchemaName is not used for filtering.

TableName

Specifies a table name.

If NULL is specified, all table names are processed.

NameLength3

Specifies the length of *TableName# or SQL_NTS.

If 0 is specified, the driver assumes that NULL is specified for TableName.

Unique

Specifies the type of indexes:

- SQL_INDEX_UNIQUE

  Specifies that information about unique indexes only is to be acquired.

- SQL_INDEX_ALL

  Specifies that information about all indexes is to be acquired.

Reserved

Specifies that the specified information is to be used as the option that indicates the importance of the CARDINALITY and PAGES columns in the result set. Note that this argument is ignored, if specified.

#

The length must be in bytes for SQLStatistics and in characters for SQLStatisticsW.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table describes the format of result set that is returned.

Table 16-15: Format of result set that is returned

Column No.	Type	Column name	Description
1	Varchar	TABLE_CAT	The function always returns a null value.
2	Varchar	TABLE_SCHEM	Schema name
3	Varchar	TABLE_NAME	Table name
4	Smallint	NON_UNIQUE	If the key value defining the index (value of one column or a set of multiple columns that are defined as the index) can be nonunique, the driver returns SQL_TRUE; otherwise, the driver returns SQL_FALSE.
5	Varchar	INDEX_QUALIFIER	The function always returns a null value.
6	Varchar	INDEX_NAME	Index identifier
7	Smallint	TYPE	Index type. The driver always returns SQL_INDEX_OTHER.
8	Smallint	ORDINAL_POSITION	• For a single-column index, the function returns 1. • For a multiple-column index, the function returns the number indicating the order of the column in the index (integer beginning with 1 that identifies the order of the column names constituting the index).
9	Varchar	COLUMN_NAME	Column name
10	Char(1)	ASC_OR_DESC	• If the B-tree index is defined in ascending order, the function returns A. • If the B-tree index is defined in descending order, the function returns D. • For a text index or range index, the function returns a null value.
11	Integer	CARDINALITY	The function always returns 0.
12	Integer	PAGES	
13	Varchar	FILTER_CONDITION	The function always returns a null value.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	N
40001	Serialization failure	--	N
40003	Statement completion unknown		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	N
HY000	General error	--	N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer	A null pointer is specified in TableName, or 0 is specified in NameLength3.	Y
HY010	Function sequence error	--	Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HY100	Uniqueness option out of range	The value specified in Unique is invalid.	Y
HY101	Precision option out of range	--	N
HY117	Connection suspended		N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The information about indexes that can be acquired depends on the privileges of the HADB user who executes this method. For details about the privileges and the index information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.10.9 SQLTablePrivileges, SQLTablePrivilegesW

(1) Function

This ODBC function returns a list of tables and a list of access privileges associated with each table. The driver returns the information as a result set for the specified SQL statement.

(2) Format

- For `SQLTablePrivileges`

```
SQLRETURN SQLTablePrivileges
(
  SQLHSTMT       StatementHandle,    /* In */
  SQLCHAR      * CatalogName,        /* In */
  SQLSMALLINT    NameLength1,        /* In */
  SQLCHAR      * SchemaName,         /* In */
  SQLSMALLINT    NameLength2,        /* In */
  SQLCHAR      * TableName,          /* In */
  SQLSMALLINT    NameLength3         /* In */
)
```

- For `SQLTablePrivilegesW`

```
SQLRETURN SQLTablePrivilegesW
(
  SQLHSTMT       StatementHandle,    /* In */
  SQLWCHAR     * CatalogName,        /* In */
  SQLSMALLINT    NameLength1,        /* In */
  SQLWCHAR     * SchemaName,         /* In */
  SQLSMALLINT    NameLength2,        /* In */
  SQLWCHAR     * TableName,          /* In */
  SQLSMALLINT    NameLength3         /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`CatalogName`

Specifies a catalog name of the table.

This argument is ignored, if specified. When you use this function, specify a null character string ("") or `NULL`.

`NameLength1`

Specifies the length of `*CatalogName`[#1] or `SQL_NTS`.

This argument is ignored, if specified. When you use this function, specify `0`.

SchemaName

Specifies a pattern character string[#2] for schema names of the table. If NULL is specified, all schema names are processed.

NameLength2

Specifies the length of *SchemaName[#1] or SQL_NTS. If 0 is specified, the search is not narrowed by the specification of SchemaName.

TableName

Specifies a pattern character string[#2] for table names. If NULL is specified, all table names are processed.

NameLength3

Specifies the length of *TableName[#1] or SQL_NTS. If 0 is specified, the search is not narrowed by the specification of TableName.

#1

The length must be in bytes for SQLTablePrivileges and in characters for SQLTablePrivilegesW.

#2

For details about the special characters that can be specified in pattern character strings, see Table 16-10: Special characters that can be specified in pattern character strings.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

When SQLTablePrivileges or SQLTablePrivilegesW is executed, a result set is created. The following table shows the format of the result set that is returned.

Table 16-16: Format of the result set that is returned.

Column No.	Type	Column name	Description
1	Varchar	TABLE_CAT	Catalog name The function always returns a null value.
2	Varchar	TABLE_SCHEM	Schema name
3	Varchar	TABLE_NAME	Table name
4	Varchar	GRANTOR	User who grants access privileges
5	Varchar	GRANTEE	User who receives access privileges
6	Varchar	PRIVILEGE	Granted access privileges: • "SELECT": SELECT privilege • "INSERT": INSERT privilege • "UPDATE": UPDATE privilege • "DELETE": DELETE privilege • "TRUNCATE": TRUNCATE privilege • "REFERENCES": REFERENCES privilege • "IMPORT TABLE": IMPORT TABLE privilege • "REBUILD INDEX": REBUILD INDEX privilege • "GET COSTINFO": GET COSTINFO privilege • "EXPORT TABLE": EXPORT TABLE privilege

Column No.	Type	Column name	Description
			• "MERGE CHUNK": MERGE CHUNK privilege • "CHANGE CHUNK COMMENT": CHANGE CHUNK COMMENT privilege • "CHANGE CHUNK STATUS": CHANGE CHUNK STATUS privilege • "ARCHIVE CHUNK": ARCHIVE CHUNK privilege • "UNARCHIVE CHUNK": UNARCHIVE CHUNK privilege
7	Varchar	IS_GRANTABLE	Whether a user who has received an access privilege can grant the access privilege to other HADB users • "YES": The user can grant the access privilege to other HADB users. • "NO": The user cannot grant the access privilege to other HADB users.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status	The function was executed while a cursor was open.	Y
40001	Serialization failure	--	N
40003	Statement completion unknown		N
HY000	General error		N
HY001	Memory allocation error		Y
HY008	Operation cancelled		N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HY117	Connection suspended	--	N
HYC00	Optional feature not implemented		N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The information that can be acquired depends on the privileges of the HADB user who executes this method. For details about the relationship between privileges and the information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.10.10 SQLTables, SQLTablesW

(1) Function

This ODBC function returns a list of the table names, schema names, and table types stored at the HADB server.

The driver returns the information as a result set for the specified SQL statement.

(2) Format

- For SQLTables

```
SQLRETURN SQLTables
(
  SQLHSTMT         StatementHandle,        /* In */
  SQLCHAR        * CatalogName,            /* In */
  SQLSMALLINT      NameLength1,            /* In */
  SQLCHAR        * SchemaName,             /* In */
  SQLSMALLINT      NameLength2,            /* In */
  SQLCHAR        * TableName,              /* In */
  SQLSMALLINT      NameLength3,            /* In */
  SQLCHAR        * TableType,              /* In */
  SQLSMALLINT      NameLength4             /* In */
)
```

- For SQLTablesW

```
SQLRETURN SQLTablesW
(
  SQLHSTMT         StatementHandle,        /* In */
  SQLWCHAR       * CatalogName,            /* In */
  SQLSMALLINT      NameLength1,            /* In */
  SQLWCHAR       * SchemaName,             /* In */
  SQLSMALLINT      NameLength2,            /* In */
  SQLWCHAR       * TableName,              /* In */
  SQLSMALLINT      NameLength3,            /* In */
  SQLWCHAR       * TableType,              /* In */
  SQLSMALLINT      NameLength4             /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

`CatalogName`

Specifies that the specified information is to be used as a catalog name.

The specified information is ignored, but the specification itself is required. Therefore, when you use this function, specify a null character string (`""`) or `NULL`.

`NameLength1`

Specifies the length of `*CatalogName`[#1] or `SQL_NTS`. The specified information is ignored, but the specification itself is required.

Therefore, when you use this function, specify `0`.

`SchemaName`

Specifies an authorization identifier or a pattern character string[#2] for authorization identifiers. If a null pointer or a null character string (`""`) is specified, the driver assumes that `'%'` is specified as the pattern character string.

`NameLength2`

Specifies the length of `*SchemaName`[#1] or `SQL_NTS`.

If `0` is specified, the processing is the same as when `'%'` is specified as the pattern character string in `SchemaName`.

`TableName`

Specifies a table identifier or a pattern character string[#2] for table identifiers. If a null pointer or a null character string (`""`) is specified, the driver assumes that `'%'` is specified as the pattern character string.

`NameLength3`

Specifies the length of `*TableName`[#1] or `SQL_NTS`.

If `0` is specified, the processing is the same as when `'%'` is specified as the pattern character string in `TableName`.

`TableType`

Specifies a pointer to the character string that indicates the matching table type. The HADB ODBC driver recognizes the following character strings as table types:

- `SYSTEM TABLE`
- `TABLE`
- `VIEW`

When you specify this argument, optionally enclose each character string in single quotation marks (`'`). To acquire the results of multiple table types, delimit the values with the comma (`,`).

Examples:

- To acquire dictionary tables and system tables
  Specify `SYSTEM TABLE` or `'SYSTEM TABLE'`.
- To acquire base and viewed tables
  Specify `TABLE,VIEW` or `'TABLE','VIEW'`.

If a null pointer, a null character string (`""`), or `SQL_ALL_TABLE_TYPES` is specified in this argument, `'SYSTEM TABLE','TABLE','VIEW'` is assumed as the table types.

If the specified value contains a character string that is not recognized as a table type by the HADB ODBC driver, only any character strings recognized by the HADB ODBC driver take effect, and any others are ignored. If the specified value contains only invalid character strings, the HADB ODBC driver assumes that `'SYSTEM`

TABLE','TABLE','VIEW' is specified, in the same manner as when a null pointer or a null character string ("") is specified.

NameLength4

Specifies the length of *TableType[#1] or SQL_NTS.

If 0 is specified, 'SYSTEM TABLE','TABLE','VIEW' is assumed for TableType.

#1

The length must be in bytes for SQLTables and in characters for SQLTablesW.

#2

For details about the special characters that can be specified in pattern character strings, see Table 16-10: Special characters that can be specified in pattern character strings.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

The following table describes the format of the result set that is returned.

Table 16-17: Format of result set that is returned

Column No.	Type	Column name	Description
1	Varchar	TABLE_CAT	The function always returns a null value.
2	Varchar	TABLE_SCHEM	Schema name
3	Varchar	TABLE_NAME	Table name
4	Varchar	TABLE_TYPE	Table type: • TABLE: Base table • VIEW: Viewed table • SYSTEM TABLE: Dictionary table or system table
5	Varchar	REMARKS	The function always returns a null value.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08S01	Communication link failure		N
24000	Invalid cursor status		N
40001	Serialization failure		N
40003	Statement completion unknown		N
5C002	Character encoding conversion error	A character code that cannot be converted was detected.	Y
5C041	Unsupported data type error	The driver does not support the specified data type.	N

SQLSTATE	Description	Remarks	Returned
HY000	General error	--	N
HY001	Memory allocation error	The memory required to execute or complete the function has not been allocated for the HADB ODBC driver.	Y
HY008	Operation cancelled	--	N
HY009	Invalid use of null pointer		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HY090	Invalid string or buffer length	One of the arguments that stores a name length exceeded the maximum length for the corresponding name.	Y
HYC00	Optional feature not implemented	--	N
HYT00	Timeout expired		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

The table information that can be acquired depends on the privileges of the HADB user who executes this function. For details about the privileges and the table information that can be acquired, see the topic *Scope of information in dictionary tables and system tables that can be referenced by HADB users* in the *HADB Setup and Operation Guide*.

16.11 Terminating execution of SQL statements

This section explains the ODBC functions that are used when execution of SQL statements is terminated.

16.11.1 SQLFreeStmt

(1) Function

This ODBC function performs on the specified statement the processing specified by an option. It stops the processing associated with the statement, closes cursors, discards unprocessed results, and releases all resources associated with the statement handle.

(2) Format

```
SQLRETURN SQLFreeStmt
(
  SQLHSTMT       StatementHandle,       /* In */
  SQLUSMALLINT   Option                 /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`Option`

Specifies one of the following options:

Option	Description
`SQL_CLOSE`	Closes the cursor managed by `StatementHandle`. The processing is the same as is performed by `SQLCloseCursor`, but with this function no error occurs if no cursor is open. The function terminates with `SQL_SUCCESS`.
`SQL_UNBIND`	Sets the `SQL_DESC_COUNT` field of the ARD to `0` and releases all columns bound by `SQLBindCol` for `StatementHandle`.
`SQL_RESET_PARAMS`	Sets the `SQL_DESC_COUNT` field of the APD to `0` and releases all parameters bound by `SQLBindParameter` for `StatementHandle`.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
24000	Invalid cursor status		Y
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error	Before this function was executed, SQLExecute, SQLExecDirect, SQLExecDirectW, or SQLParamData was called for StatementHandle and returned SQL_NEED_DATA. Since then, the setting of runtime data parameters or runtime data columns has not been completed.	Y
HY013	Memory management error	--	N
HY092	Invalid attribute or option identifier	SQL_DROP or an invalid identifier was specified in Option.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

This function does not support SQL_DROP in Option. Use SQLFreeHandle to release the statement handle.

SQL_UNBIND and SQL_RESET_PARAMS in Option only perform unbind operations; they do not release memory. Therefore, use the application program to release the memory area for data that was passed to the HADB ODBC driver by SQLBindCol and SQLBindParameter.

16.11.2 SQLCloseCursor

(1) Function

This ODBC function closes a cursor that has been opened by a specified statement handle and discards unprocessed results.

(2) Format

```
SQLRETURN SQLCloseCursor
(
  SQLHSTMT         StatementHandle       /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
`01000`	General warning	--	N
`24000`	Invalid cursor status	No cursor has been opened by `StatementHandle`.	Y
`HY000`	General error	--	N
`HY001`	Memory allocation error	The memory required to execute or complete the function has not been allocated for the HADB ODBC driver.	N
`HY010`	Function sequence error	--	N
`HY013`	Memory management error		N
`HYT01`	Connection timeout expired		N
`IM001`	Driver does not support this function		N

Legend:

Y: This `SQLSTATE` might be returned by the HADB ODBC driver.

N: This `SQLSTATE` is not returned by the HADB ODBC driver.

--: None

16.11.3 SQLCancel

(1) Function

This ODBC function cancels the SQL statement processing that is underway.

(2) Format

```
SQLRETURN SQLCancel
(
  SQLHSTMT        StatementHandle         /* In */
)
```

(3) Arguments

`StatementHandle`

Specifies a statement handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function does not return `SQLSTATE`.

16.11.4 SQLEndTran

(1) Function

This ODBC function requests commit or rollback processing for all active operations on all statements associated with a connection.

(2) Format

```
SQLRETURN SQLEndTran
(
  SQLSMALLINT    HandleType,         /* In */
  SQLHANDLE      Handle,             /* In */
  SQLSMALLINT    CompletionType      /* In */
)
```

(3) Arguments

`HandleType`

Specifies the following handle type:

- `SQL_HANDLE_DBC`: Connection handle

`Handle`

Specifies a connection handle.

Specify a value that was output by `*OutputHandlePtr` of `SQLAllocHandle` before this function is executed.

`CompletionType`

Specifies one of the following processing:

- `SQL_COMMIT`
  Commit processing
- `SQL_ROLLBACK`
  Rollback processing

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
08003	Connection does not exist		Y
08007	Connection failure during transaction		N
25S01	Transaction status unknown		N
25S02	Transaction is still active		N
25S03	Transaction is rolled back		N
40001	Serialization failure		N
40002	Integrity mismatch		N
HY000	General error		N
HY001	Memory allocation error		N
HY008	Operation cancelled		N
HY010	Function sequence error		Y
HY012	Invalid transaction processing code		N
HY013	Memory management error		N
HY092	Invalid attribute or option identifier	HandleType or CompletionType was set to an invalid value.	Y
HY115	Environment that contains a connection under asynchronous execution was specified	--	N
HYC00	Optional feature not implemented		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N
IM017	Invalid asynchronous polling		N
IM018	Incomplete asynchronous execution		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

(6) Notes

This ODBC function does not support the following handle type:

- SQL_HANDLE_ENV: Environment handle

If either of the following handle types is specified, the driver might return SQL_INVALID_HANDLE:

- SQL_HANDLE_STMT: Statement handle
- SQL_HANDLE_DESC: Descriptor handle

16.12 Disconnecting from the data source

This section explains the ODBC functions that are used to disconnect the HADB ODBC driver from the data source.

16.12.1 SQLDisconnect

(1) Function

This ODBC function closes the connection associated with a specific connection handle.

(2) Format

```
SQLRETURN SQLDisconnect
(
  SQLHDBC           ConnectionHandle    /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

(4) Return value

This ODBC function returns `SQL_SUCCESS`, `SQL_SUCCESS_WITH_INFO`, `SQL_ERROR`, or `SQL_INVALID_HANDLE`.

(5) SQLSTATE

This ODBC function returns one of the following `SQLSTATE` values:

SQLSTATE	Description	Remarks	Returned
01000	General warning	--	N
01002	Disconnection error		N
08003	Connection does not exist		Y
25000	Invalid transaction status		Y
HY000	General error		N
HY001	Memory allocation error		N
HY010	Function sequence error		Y
HY013	Memory management error		N
HYT01	Connection timeout expired		N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.12.2 SQLFreeHandle

(1) Function

This ODBC function releases resources associated with a specific environment, connection, statement, or descriptor handle.

(2) Format

```
SQLRETURN SQLFreeHandle
(
  SQLSMALLINT     HandleType,         /* In */
  SQLHANDLE       Handle              /* In */
)
```

(3) Arguments

`HandleType`

Specifies one of the following handle types:

- SQL_HANDLE_ENV: Environment handle
- SQL_HANDLE_DBC: Connection handle
- SQL_HANDLE_STMT: Statement handle
- SQL_HANDLE_DESC: Descriptor handle

`Handle`

Specifies the handle to be released.

Specify a value that was output by *OutputHandlePtr of SQLAllocHandle before this function is executed.

(4) Return value

This ODBC function returns SQL_SUCCESS, SQL_ERROR, or SQL_INVALID_HANDLE.

(5) SQLSTATE

This ODBC function returns one of the following SQLSTATE values:

SQLSTATE	Description	Remarks	Return
01000	General warning	--	N
HY001	Memory allocation error		N

SQLSTATE	Description	Remarks	Return
HY010	Function sequence error	• Not all subsidiary handles and other resources, including connections, have been released yet. • HandleType is SQL_HANDLE_STMT, but the asynchronous execution function for that statement handle has not terminated.	Y
HY013	Memory management error	--	N
HY017	Invalid use of an automatically allocated descriptor handle		N
HY092	Invalid attribute or option identifier	An invalid value was specified for HandleType.	Y
HYT01	Connection timeout expired	--	N
IM001	Driver does not support this function		N

Legend:

Y: This SQLSTATE might be returned by the HADB ODBC driver.

N: This SQLSTATE is not returned by the HADB ODBC driver.

--: None

16.13 Information types that can be specified for InfoType in SQLGetInfo and SQLGetInfoW

The following table lists and describes the information types that can be specified for InfoType.

Table 16-18: Information types that can be specified for InfoType

No.	Information type (InfoType)	Description (convention)	Return value	Data type
1	SQL_ACCESSIBLE_PROCEDURES	One of the following values: Y SQLProcedures can execute all procedures. N There is at least one procedure that SQLProcedures cannot execute.	Returns N.	• SQLCHAR * • SQLWCHAR *
2	SQL_ACCESSIBLE_TABLES	One of the following values: Y The SELECT privileges are guaranteed to all tables returned by SQLTables. N There is at least one table returned by SQLTables that the application program cannot access.	Returns Y.	• SQLCHAR * • SQLWCHAR *
3	SQL_ACTIVE_ENVIRONMENTS	Maximum number of active environments that the driver supports. If no limit is specified or the limit is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
4	SQL_AGGREGATE_FUNCTIONS	Support status for aggregation functions.	Returns the following bit strings: • SQL_AF_ALL • SQL_AF_AVG • SQL_AF_COUNT • SQL_AF_DISTINCT • SQL_AF_MAX • SQL_AF_MIN • SQL_AF_SUM	SQLUINTEGER
5	SQL_ALTER_DOMAIN	The clauses in the ALTER DOMAIN statement that are supported by the data source. A return value of 0 means that the ALTER DOMAIN statement is not supported.	Returns 0.	SQLUINTEGER
6	SQL_ALTER_TABLE	The clauses in the ALTER TABLE statement that are supported by the data source.	Returns the following bit string according to the value specified for the SQL_ATTR_ODBC_VERSION environment handle:	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			For SQL_OV_ODBC2 SQL_AT_ADD_COLUMN For SQL_OV_ODBC3 SQL_AT_ADD_COLUMN_SINGLE	
7	SQL_ASYNC_MODE	The driver's asynchronous support level.	Returns SQL_AM_NONE.	SQLUINTEGER
8	SQL_BATCH_ROW_COUNT	The driver's behavior on available row counts.	Returns 0.	SQLUINTEGER
9	SQL_BATCH_SUPPORT	The driver's support for batches.	Returns 0.	SQLUINTEGER
10	SQL_BOOKMARK_PERSISTENCE	Processing for which bookmarks are enabled.	Returns 0.	SQLUINTEGER
11	SQL_CATALOG_LOCATION	A value indicating the position of the catalog in a qualified table name.	Returns 0.	SQLUSMALLINT
12	SQL_CATALOG_NAME	One of the following values: Y The server supports catalog names. N The server does not support catalog names.	Returns N.	• SQLCHAR * • SQLWCHAR *
13	SQL_CATALOG_NAME_SEPARATOR	A string of one or more characters that is defined as a separator. This separator is placed between a catalog name and the element that follows or precedes it. If the data source does not support catalogs, a null character string is returned.	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
14	SQL_CATALOG_TERM	The data source vendor's name as a catalog. If the data source does not support catalogs, a null character string is returned.	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
15	SQL_CATALOG_USAGE	The statements in which catalogs can be used. If the data source does not support catalogs, 0 is returned.	Returns 0.	SQLUINTEGER
16	SQL_COLLATION_SEQ	The name of the collation sequence (default collation for the server's default character set). If the name is unknown, a null character string is returned.	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
17	SQL_COLUMN_ALIAS	One of the following values: Y The data source supports column aliases.	Returns Y.	• SQLCHAR * • SQLWCHAR *

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		N The data source does not support column aliases.		
18	SQL_CONCAT_NULL_BEHAVIOR	How the data source concatenates NULL-valued character string data type columns with non-NULL-valued character string data type columns.	Returns SQL_CB_NULL.	SQLUSMALLINT
19	SQL_CONVERT_BIGINT	Conversions by the CONVERT scalar function that are supported by the data source.	Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_TIMESTAMP • SQL_CVT_VARCHAR	SQLUINTEGER
20	SQL_CONVERT_BINARY		Returns the following bit strings: • SQL_CVT_BINARY • SQL_CVT_CHAR • SQL_CVT_VARBINARY • SQL_CVT_VARCHAR	SQLUINTEGER
21	SQL_CONVERT_BIT		Returns 0.	SQLUINTEGER
22	SQL_CONVERT_CHAR		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_BINARY • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_TIME • SQL_CVT_TIMESTAMP • SQL_CVT_VARBINARY • SQL_CVT_VARCHAR	SQLUINTEGER
23	SQL_CONVERT_DATE		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_INTEGER	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			• SQL_CVT_TIMESTAMP • SQL_CVT_VARCHAR	
24	SQL_CONVERT_DECIMAL		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_VARCHAR	SQLUINTEGER
25	SQL_CONVERT_DOUBLE		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_VARCHAR	SQLUINTEGER
26	SQL_CONVERT_FLOAT		Returns 0.	SQLUINTEGER
27	SQL_CONVERT_INTEGER		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_TIMESTAMP • SQL_CVT_VARCHAR	SQLUINTEGER
28	SQL_CONVERT_INTERVAL_YEAR_MONTH		Returns 0.	SQLUINTEGER
29	SQL_CONVERT_INTERVAL_DAY_TIME		Returns 0.	SQLUINTEGER
30	SQL_CONVERT_LONGVARBINARY		Returns 0.	SQLUINTEGER
31	SQL_CONVERT_LONGVARCHAR		Returns 0.	SQLUINTEGER
32	SQL_CONVERT_NUMERIC		Returns 0.	SQLUINTEGER
33	SQL_CONVERT_REAL		Returns 0.	SQLUINTEGER
34	SQL_CONVERT_SMALLINT		Returns 0.	SQLUINTEGER
35	SQL_CONVERT_TIME		Returns the following bit strings: • SQL_CVT_CHAR • SQL_CVT_TIME	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			• SQL_CVT_VARCHAR	
36	SQL_CONVERT_TIMESTAMP		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_INTEGER • SQL_CVT_TIMESTAMP • SQL_CVT_VARCHAR	SQLUINTEGER
37	SQL_CONVERT_TINYINT		Returns 0.	SQLUINTEGER
38	SQL_CONVERT_VARBINARY		Returns the following bit strings: • SQL_CVT_BINARY • SQL_CVT_CHAR • SQL_CVT_VARBINARY • SQL_CVT_VARCHAR	SQLUINTEGER
39	SQL_CONVERT_VARCHAR		Returns the following bit strings: • SQL_CVT_BIGINT • SQL_CVT_BINARY • SQL_CVT_CHAR • SQL_CVT_DATE • SQL_CVT_DECIMAL • SQL_CVT_DOUBLE • SQL_CVT_INTEGER • SQL_CVT_TIME • SQL_CVT_TIMESTAMP • SQL_CVT_VARBINARY • SQL_CVT_VARCHAR	SQLUINTEGER
40	SQL_CONVERT_FUNCTIONS	The scalar conversion functions that are supported by the driver and associated data source.	Returns the following bit strings: • SQL_FN_CVT_CAST • SQL_FN_CVT_CONVERT	SQLUINTEGER
41	SQL_CORRELATION_NAME	Whether table correlation names are supported.	Returns SQL_CN_ANY.	SQLUSMALLINT
42	SQL_CREATE_ASSERTION	The clauses in the CREATE ASSERTION statement that are supported by the data source. A return value of 0 means that the CREATE ASSERTION statement is not supported.	Returns 0.	SQLUINTEGER
43	SQL_CREATE_CHARACTER_SET	The clauses in the CREATE CHARACTER SET statement that are supported by the data source.	Returns 0.	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		A return value of 0 means that the CREATE CHARACTER SET statement is not supported.		
44	SQL_CREATE_COLLATION	The clauses in the CREATE COLLATION statement that are supported by the data source. A return value of 0 means that the CREATE COLLATION statement is not supported.	Returns 0.	SQLUINTEGER
45	SQL_CREATE_DOMAIN	The clauses in the CREATE DOMAIN statement that are supported by the data source. A return value of 0 means that the CREATE DOMAIN statement is not supported.	Returns 0.	SQLUINTEGER
46	SQL_CREATE_SCHEMA	The clauses in the CREATE SCHEMA statement that are supported by the data source. A return value of 0 means that the CREATE SCHEMA statement is not supported.	Returns SQL_CS_CREATE_SCHEMA.	SQLUINTEGER
47	SQL_CREATE_TABLE	The clauses in the CREATE TABLE statement that are supported by the data source. A return value of 0 means that the CREATE TABLE statement is not supported.	Returns the following bit strings: • SQL_CT_CREATE_TABLE • SQL_CT_TABLE_CONSTRAINT • SQL_CT_CONSTRAINT_NAME_DEFINITION • SQL_CT_COLUMN_DEFAULT	SQLUINTEGER
48	SQL_CREATE_TRANSLATION	The clauses in the CREATE TRANSLATION statement that are supported by the data source. A return value of 0 means that the CREATE TRANSLATION statement is not supported.	Returns 0.	SQLUINTEGER
49	SQL_CREATE_VIEW	The clauses in the CREATE VIEW statement that are supported by the data source. A return value of 0 means that the CREATE VIEW statement is not supported.	Returns SQL_CV_CREATE_VIEW.	SQLUINTEGER
50	SQL_CURSOR_COMMIT_BEHAVIOR	The effects of COMMIT processing on cursors and prepared statements in the data source.	Returns SQL_CB_CLOSE.	SQLUSMALLINT
51	SQL_CURSOR_ROLLBACK_BEHAVIOR	The effects of ROLLBACK processing on cursors and prepared statements in the data source.	Returns SQL_CB_CLOSE.	SQLUSMALLINT

No.	Information type (InfoType)	Description (convention)	Return value	Data type
52	SQL_CURSOR_SENSITIVITY	Support for cursor sensitivity.	Returns SQL_UNSPECIFIED.	SQLUINTEGER
53	SQL_DATA_SOURCE_NAME	The data source name character string that is used during connection.	Returns an empty string or the data source name that was used during connection.	• SQLCHAR * • SQLWCHAR *
54	SQL_DATA_SOURCE_READ_ONLY	One of the following values: Y The data source's mode is READ ONLY. N The data source's mode is not READ ONLY.	Returns N.	• SQLCHAR * • SQLWCHAR *
55	SQL_DATABASE_NAME	A character string with the name of the database in use, if the data source defines a named object called "database".	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
56	SQL_DATETIME_LITERALS	The SQL-92 datetime literals that are supported by the data source.	Returns the following bit strings: • SQL_DL_SQL92_DATE • SQL_DL_SQL92_TIME • SQL_DL_SQL92_TIMESTAMP	SQLUINTEGER
57	SQL_DBMS_NAME	A character string that indicates the name of the DBMS product accessed by the driver.	Returns "Hitachi Advanced Data Binder".	• SQLCHAR * • SQLWCHAR *
58	SQL_DBMS_VER	A character string that indicates the version of the DBMS product accessed by the driver. The version is displayed in the format ##.##.####.	Returns the DBMS product version.	• SQLCHAR * • SQLWCHAR *
59	SQL_DDL_INDEX	Support for creation and dropping of indexes.	Returns the following bit strings: • SQL_DI_CREATE_INDEX • SQL_DI_DROP_INDEX	SQLUINTEGER
60	SQL_DEFAULT_TXN_ISOLATION	The default transaction isolation level supported by the driver or data source.	Returns SQL_TXN_READ_COMMITTED.	SQLUINTEGER
61	SQL_DESCRIBE_PARAMETER	One of the following values: Y Parameters can be described. N Parameters cannot be described.	Returns Y.	• SQLCHAR * • SQLWCHAR *
62	SQL_DM_VER	A character string for the version of the driver manager. The version is represented in the	Returns the value that is set by the driver manager.	• SQLCHAR * • SQLWCHAR *

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		format ##.##.####.####. This information type is implemented by the driver manager.		
63	SQL_DRIVER_HDBC	The driver's connection handle, which can be determined by InfoType. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	SQLUINTEGER
64	SQL_DRIVER_HDESC	The driver's descriptor handle that is determined by the descriptor handle of the driver manager. This information must be set in *InfoValuePtr and passed by the application. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	SQLUINTEGER
65	SQL_DRIVER_HENV	The driver's environment handle, which can be determined by InfoType. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	SQLUINTEGER
66	SQL_DRIVER_HLIB	The hinst that is returned from the load library to the driver manager when the driver DLL or its equivalent is loaded. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	SQLUINTEGER
67	SQL_DRIVER_HSTMT	The driver's statement handle that is determined by the driver manager's statement handle. This information must be set in *InfoValuePtr and passed by the application. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	SQLUINTEGER
68	SQL_DRIVER_NAME	A character string that indicates the driver's file name used to access the data source.	Returns the driver's file name.	• SQLCHAR * • SQLWCHAR *
69	SQL_DRIVER_ODBC_VER	A character string that indicates the ODBC version supported by the driver. The version is represented in the format ##.##.	Returns the ODBC version supported by the driver.	• SQLCHAR * • SQLWCHAR *
70	SQL_DRIVER_VER	A character string that indicates the driver's version. Normally the driver returns this value. The minimum value for the version is represented in the format ##.##.####.	Returns the driver's version.	• SQLCHAR * • SQLWCHAR *
71	SQL_DROP_ASSERTION	The clauses in the DROP ASSERTION statement (defined in SQL-92) that are supported by the data source.	Returns 0.	SQLUINTEGER
72	SQL_DROP_CHARACTER_SET	The clauses in the DROP CHARACTER SET statement	Returns 0.	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		(defined in SQL-92) that are supported by the data source.		
73	SQL_DROP_COLLATION	The clauses in the DROP COLLATION statement (defined in SQL-92) that are supported by the data source.	Returns 0.	SQLUINTEGER
74	SQL_DROP_DOMAIN	The clauses in the DROP DOMAIN statement (defined in SQL-92) that are supported by the data source.	Returns 0.	SQLUINTEGER
75	SQL_DROP_SCHEMA	The clauses in the DROP SCHEMA statement (defined in SQL-92) that are supported by the data source.	Returns the following bit strings: • SQL_DS_DROP_SCHEMA • SQL_DS_CASCADE • SQL_DS_RESTRICT	SQLUINTEGER
76	SQL_DROP_TABLE	The clauses in the DROP TABLE statement (defined in SQL-92) that are supported by the data source.	Returns the following bit strings: • SQL_DT_DROP_TABLE • SQL_DT_CASCADE • SQL_DT_RESTRICT	SQLUINTEGER
77	SQL_DROP_TRANSLATION	The clauses in the DROP TRANSLATION statement that are supported by the data source.	Returns 0.	SQLUINTEGER
78	SQL_DROP_VIEW	The clauses in the DROP VIEW statement that are supported by the data source.	Returns SQL_DV_DROP_VIEW.	SQLUINTEGER
79	SQL_DYNAMIC_CURSOR_ATTRIBUTES1	Attributes of a dynamic cursor that is supported by the driver. This bitmask contains the first subset of attributes.	Returns 0.	SQLUINTEGER
80	SQL_DYNAMIC_CURSOR_ATTRIBUTES2	Attributes of a dynamic cursor that is supported by the driver. This bitmask contains the second subset of attributes.	Returns 0.	SQLUINTEGER
81	SQL_EXPRESSIONS_IN_ORDERBY	One of the following values: Y The data source supports expressions in the ORDER BY list. N The data source does not support expressions in the ORDER BY list.	Returns Y.	• SQLCHAR * • SQLWCHAR *
82	SQL_FILE_USAGE	How the single-tier driver directly handles files in a data source.	Returns SQL_FILE_NOT_SUPPORTED.	SQLUSMALLINT

No.	Information type (InfoType)	Description (convention)	Return value	Data type
83	SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	Attributes of a forward-only cursor that is supported by the driver. This bitmask contains the first subset of attributes.	Returns 0.	SQLUINTEGER
84	SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	Attributes of a forward-only cursor that is supported by the driver. This bitmask contains the second subset of attributes.	Returns 0.	SQLUINTEGER
85	SQL_GETDATA_EXTENSIONS	Extensions to SQLGetData.	Returns the following values: • SQL_GD_ANY_COLUMN • SQL_GD_ANY_ORDER	SQLUINTEGER
86	SQL_GROUP_BY	The relationship between the columns in the GROUP BY clause and the columns in the selection list to which a set function has not been applied.	Returns SQL_GB_GROUP_BY_CONTAINS_SELECT.	SQLUSMALLINT
87	SQL_IDENTIFIER_CASE	Information about identifiers in SQL statements.	Returns SQL_IC_UPPER.	SQLUSMALLINT
88	SQL_IDENTIFIER_QUOTE_CHAR	The character string that is used as the starting and ending delimiter of a quoted identifier in SQL statements. If the data source does not support quoted identifiers, a blank is returned.	Returns a double quotation mark (").	• SQLCHAR * • SQLWCHAR *
89	SQL_INDEX_KEYWORDS	Keywords in the CREATE INDEX statement that are supported by the driver.	Returns the following bit strings: • SQL_IK_ASC • SQL_IK_DESC	SQLUINTEGER
90	SQL_INFO_SCHEMA_VIEWS	The views in INFORMATION_SCHEMA that are supported by the driver.	Returns 0.	SQLUINTEGER
91	SQL_INSERT_STATEMENT	Support for INSERT statements.	Returns the following bit strings: • SQL_IS_INSERT_LITERALS • SQL_IS_INSERT_SEARCHED	SQLUINTEGER
92	SQL_INTEGRITY	One of the following values: Y The data source supports IEF. N The data source does not support IEF.	Returns N.	• SQLCHAR * • SQLWCHAR *
93	SQL_KEYSET_CURSOR_ATTRIBUTES1	Attributes of a keyset cursor that is supported by the driver. This bitmask contains the first subset of attributes.	Returns 0.	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
94	SQL_KEYSET_CURSOR_ATTRIBUTES2	Attributes of a keyset cursor that is supported by the driver. This bitmask contains the second subset of attributes.	Returns 0.	SQLUINTEGER
95	SQL_KEYWORDS	A character string that contains a comma-separated list of all keywords specific to the data source. This list does not contain keywords specific to ODBC or keywords used by both the data source and ODBC. This list represents all the reserved keywords.	Returns HADB's reserved words without the ODBC-specific keywords.	• SQLCHAR * • SQLWCHAR *
96	SQL_LIKE_ESCAPE_CLAUSE	One of the following values: Y The following conditions are both satisfied: • The data source supports an escape character for the percent sign (%) and underscore (_) in a LIKE predicate. • The driver supports the ODBC syntax for defining a LIKE predicate escape character. N Other.	Returns Y.	• SQLCHAR * • SQLWCHAR *
97	SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	Maximum number of active concurrent statements in asynchronous mode that the driver can support on a specified connection. If there is no specific limit or the limit is unknown, 0 is returned.	Returns 0.	SQLUINTEGER
98	SQL_MAX_BINARY_LITERAL_LEN	Maximum length of a binary literal in an SQL statement. If there is no maximum length or the length is unknown, 0 is returned.	Returns 64000.	SQLUINTEGER
99	SQL_MAX_CATALOG_NAME_LEN	Maximum length of a catalog name. If there is no maximum length or the length is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
100	SQL_MAX_CHAR_LITERAL_LEN	Maximum length of a character literal in an SQL statement. If there is no maximum length or the length is unknown, 0 is returned.	Returns 32000.	SQLUINTEGER
101	SQL_MAX_COLUMN_NAME_LEN	Maximum length of a column name in the data source. If there is no maximum length or the length is unknown, 0 is returned.	Returns 100.	SQLUSMALLINT

No.	Information type (InfoType)	Description (convention)	Return value	Data type
102	SQL_MAX_COLUMNS_IN_GROUP_BY	Maximum number of grouping columns permitted in a GROUP BY clause. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 64.	SQLUSMALLINT
103	SQL_MAX_COLUMNS_IN_INDEX	Maximum number of columns permitted in an index. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 16.	SQLUSMALLINT
104	SQL_MAX_COLUMNS_IN_ORDER_BY	Maximum number of columns permitted in an ORDER BY clause. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 16.	SQLUSMALLINT
105	SQL_MAX_COLUMNS_IN_SELECT	Maximum number of columns permitted in a selection list. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 1000.	SQLUSMALLINT
106	SQL_MAX_COLUMNS_IN_TABLE	Maximum number of columns permitted in a table. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 1000.	SQLUSMALLINT
107	SQL_MAX_CONCURRENT_ACTIVITIES	Maximum number of active statements that the driver supports for a connection. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
108	SQL_MAX_CURSOR_NAME_LEN	Maximum length of a cursor name in the data source. If there is no maximum length or the length is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
109	SQL_MAX_DRIVER_CONNECTIONS	Maximum number of active connections that the driver supports for an environment. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
110	SQL_MAX_IDENTIFIER_LEN	Maximum length of a character string that the data source supports for user-defined names.	Returns 100.	SQLUSMALLINT
111	SQL_MAX_INDEX_SIZE	Maximum length (in bytes) permitted for the combined fields of an index. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 4036.	SQLUINTEGER
112	SQL_MAX_PROCEDURE_NAME_LEN	Maximum length of a procedure name in the data source. If there is no maximum length or the length is unknown, 0 is returned.	Returns 0.	SQLUSMALLINT
113	SQL_MAX_ROW_SIZE	Maximum length of a row permitted in a table. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 0.	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
114	SQL_MAX_ROW_SIZE_INCLUDES_LONG	One of the following values: Y The maximum row size returned for the SQL_MAX_ROW_SIZE information type includes the length of all columns of the SQL_LONGVARCHAR and SQL_LONGVARBINARY types. N Other	Returns N.	• SQLCHAR * • SQLWCHAR *
115	SQL_MAX_SCHEMA_NAME_LEN	Maximum length of a schema name in the data source. If there is no maximum length or the length is unknown, 0 is returned.	Returns 100.	SQLUSMALLINT
116	SQL_MAX_STATEMENT_LEN	Maximum length (in characters) of an SQL statement. If there is no maximum length or the length is unknown, 0 is returned.	Returns 16000000.	SQLUINTEGER
117	SQL_MAX_TABLE_NAME_LEN	Maximum length of a data source table name. If there is no maximum length or the length is unknown, 0 is returned.	Returns 100.	SQLUSMALLINT
118	SQL_MAX_TABLES_IN_SELECT	Maximum number of tables permitted in the FROM clause of a SELECT statement. If there is no specified limit or the limit is unknown, 0 is returned.	Returns 64.	SQLUSMALLINT
119	SQL_MAX_USER_NAME_LEN	Maximum length of a user name in the data source. If there is no maximum length or the length is unknown, 0 is returned.	Returns 100.	SQLUSMALLINT
120	SQL_MULT_RESULT_SETS	One of the following values: Y The data source supports multiple result sets. N The data source does not support multiple result sets.	Returns N.	• SQLCHAR * • SQLWCHAR *
121	SQL_MULTIPLE_ACTIVE_TXN	One of the following values: Y The driver supports concurrent execution of multiple active transactions. N The driver supports execution of only one active transaction at a time.	Returns Y.	• SQLCHAR * • SQLWCHAR *
122	SQL_NEED_LONG_DATA_LEN	One of the following values:	Returns Y.	• SQLCHAR * • SQLWCHAR *

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		Y The data source needs the length of a long data value before that value is sent. N The data source does not need the length of a long data value before that value is sent.		
123	SQL_NON_NULLABLE_COLUMNS	Whether NOT NULL columns in the data source are supported.	Returns SQL_NNC_NON_NULL.	SQLUSMALLINT
124	SQL_NULL_COLLATION	Where NULL is sorted in a result set.	Returns SQL_NC_HIGH.	SQLUSMALLINT
125	SQL_NUMERIC_FUNCTIONS	The numeric functions supported by the driver and data source.	Returns the following bit strings: • SQL_FN_NUM_ABS • SQL_FN_NUM_ACOS • SQL_FN_NUM_ASIN • SQL_FN_NUM_ATAN • SQL_FN_NUM_ATAN2 • SQL_FN_NUM_CEILING • SQL_FN_NUM_COS • SQL_FN_NUM_DEGREES • SQL_FN_NUM_EXP • SQL_FN_NUM_FLOOR • SQL_FN_NUM_LOG • SQL_FN_NUM_MOD • SQL_FN_NUM_PI • SQL_FN_NUM_POWER • SQL_FN_NUM_RADIANS • SQL_FN_NUM_RAND • SQL_FN_NUM_ROUND • SQL_FN_NUM_SIGN • SQL_FN_NUM_SIN • SQL_FN_NUM_SQRT • SQL_FN_NUM_TAN • SQL_FN_NUM_TRUNCATE	SQLUINTEGER
126	SQL_ODBC_INTERFACE_CONFORMANCE	Level of the ODBC 3.x interface that the driver complies with.	Returns SQL_OIC_CORE.	SQLUINTEGER
127	SQL_ODBC_VER	A character string containing the version of ODBC with which the driver manager conforms. The version is represented in the format ##.##.0000. This information type is implemented by the driver manager.	Returns the value that is set by the driver manager.	• SQLCHAR * • SQLWCHAR *

No.	Information type (InfoType)	Description (convention)	Return value	Data type
128	SQL_OJ_CAPABILITIES	The types of outer joins supported by the driver and data source.	Returns the following bit strings: • SQL_OJ_ALL_COMPARISON_OPS • SQL_OJ_FULL • SQL_OJ_INNER • SQL_OJ_LEFT • SQL_OJ_NESTED • SQL_OJ_RIGHT	SQLUINTEGER
129	SQL_ORDER_BY_COLUMNS_IN_SELECT	One of the following values: Y The columns in the ORDER BY clause are in the selection list. N The columns in the ORDER BY clause are not in the selection list.	Returns N.	• SQLCHAR * • SQLWCHAR *
130	SQL_PARAM_ARRAY_ROW_COUNTS	The driver's properties regarding the availability of row counts in a parameterized execution.	Returns 0.	SQLUINTEGER
131	SQL_PARAM_ARRAY_SELECTS	The driver's properties regarding the availability of result sets in a parameterized execution.	Returns SQL_PAS_NO_SELECT.	SQLUINTEGER
132	SQL_PROCEDURE_TERM	A character string containing the data source vendor's name for a procedure.	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
133	SQL_PROCEDURES	One of the following values: Y The following conditions are both satisfied: The data source supports procedures. The driver supports the ODBC procedure invocation syntax. N Other.	Returns N.	• SQLCHAR * • SQLWCHAR *
134	SQL_QUOTED_IDENTIFIER_CASE	Information about quoted identifiers in SQL statements.	Returns SQL_IC_SENSITIVE.	SQLUSMALLINT
135	SQL_ROW_UPDATES	One of the following values: Y A keyset or mixed cursor retains row versions or values for all fetched rows and can detect any updates made to a row by any application program since the row was last fetched.	Returns N.	• SQLCHAR * • SQLWCHAR *

No.	Information type (InfoType)	Description (convention)	Return value	Data type
		N Other.		
136	SQL_SCHEMA_TERM	A character string containing the data source vendor's name for a schema.	Returns "schema".	• SQLCHAR * • SQLWCHAR *
137	SQL_SCHEMA_USAGE	The statements in which schemas can be used.	Returns the following bit strings: • SQL_SU_DML_STATEMENTS • SQL_SU_INDEX_DEFINITION • SQL_SU_TABLE_DEFINITION • SQL_SU_PRIVILEGE_DEFINITION	SQLUINTEGER
138	SQL_SCROLL_OPTIONS	Scroll options supported for scrollable cursors.	Returns SQL_SO_FORWARD_ONLY.	SQLUINTEGER
139	SQL_SEARCH_PATTERN_ESCAPE	A character string indicating that the driver supports an escape character that uses the pattern search metacharacters underscore (_) and percent sign (%) as valid characters in search patterns. This indicates an escape character that handles metacharacters underscore (_) and percent sign (%) for pattern search as valid search characters. This escape character applies only to the arguments of catalog functions that support search character strings. If a null character string is specified, the driver cannot handle underscore (_) and percent sign (%) as a part of search pattern character strings.	Returns "\".	• SQLCHAR * • SQLWCHAR *
140	SQL_SERVER_NAME	A character string containing the actual data source-specific server name.	Returns "Hitachi Advanced Data Binder".	• SQLCHAR * • SQLWCHAR *
141	SQL_SPECIAL_CHARACTERS	A character string that contains all special characters that can be used in an identifier name (such as a table name, column name, or index name) in the data source.	Returns a null character string.	• SQLCHAR * • SQLWCHAR *
142	SQL_SQL_CONFORMANCE	The level of SQL-92 supported by the driver.	Returns SQL_SC_SQL92_ENTRY.	SQLUINTEGER
143	SQL_SQL92_DATETIME_FUNCTIONS	The datetime scalar functions that are supported by the driver and the associated data source.	Returns the following bit strings: • SQL_SDF_CURRENT_DATE	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			• SQL_SDF_CURRENT_TIME • SQL_SDF_CURRENT_TIMESTAMP	
144	SQL_SQL92_FOREIGN_KEY_DELETE_RULE	The rules supported for a foreign key in a DELETE statement.	Returns 0.	SQLUINTEGER
145	SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	The rules supported for a foreign key in an UPDATE statement.	Returns 0.	SQLUINTEGER
146	SQL_SQL92_GRANT	The clauses supported in a GRANT statement.	Returns the following bit strings: • SQL_SG_DELETE_TABLE • SQL_SG_INSERT_TABLE • SQL_SG_REFERENCES_TABLE • SQL_SG_SELECT_TABLE • SQL_SG_UPDATE_TABLE	SQLUINTEGER
147	SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	The numeric scalar functions supported by the driver and associated data source.	Returns the following bit string: • SQL_SNVF_EXTRACT	SQLUINTEGER
148	SQL_SQL92_PREDICATES	The predicates supported in a SELECT statement.	Returns the following bit strings: • SQL_SP_BETWEEN • SQL_SP_COMPARISON • SQL_SP_EXISTS • SQL_SP_IN • SQL_SP_ISNOTNULL • SQL_SP_ISNULL • SQL_SP_LIKE • SQL_SP_QUANTIFIED_COMPARISON	SQLUINTEGER
149	SQL_SQL92_RELATIONAL_JOIN_OPERATORS	The relational join operators supported in a SELECT statement.	Returns the following bit strings: • SQL_SRJO_CROSS_JOIN • SQL_SRJO_FULL_OUTER_JOIN • SQL_SRJO_INNER_JOIN • SQL_SRJO_LEFT_OUTER_JOIN • SQL_SRJO_RIGHT_OUTER_JOIN	SQLUINTEGER
150	SQL_SQL92_REVOKE	The clauses supported in a REVOKE statement that is supported by the data source.	Returns the following bit strings: • SQL_SR_CASCADE	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			• SQL_SR_DELETE_TABLE • SQL_SR_INSERT_TABLE • SQL_SR_REFERENCES_TABLE • SQL_SR_RESTRICT • SQL_SR_SELECT_TABLE • SQL_SR_UPDATE_TABLE	
151	SQL_SQL92_ROW_VALUE_CONSTRUCTOR	The row value constructor expressions supported in a SELECT statement.	Returns 0.	SQLUINTEGER
152	SQL_SQL92_STRING_FUNCTIONS	The string scalar functions supported by the driver and associated data source.	Returns the following bit strings: • SQL_SSF_CONVERT • SQL_SSF_LOWER • SQL_SSF_SUBSTRING • SQL_SSF_TRIM_BOTH • SQL_SSF_TRIM_LEADING • SQL_SSF_TRIM_TRAILING • SQL_SSF_UPPER	SQLUINTEGER
153	SQL_SQL92_VALUE_EXPRESSIONS	Supported value expressions.	Returns the following bit strings: • SQL_SVE_CASE • SQL_SVE_CAST • SQL_SVE_COALESCE • SQL_SVE_NULLIF	SQLUINTEGER
154	SQL_STANDARD_CLI_CONFORMANCE	The CLI standard with which the driver conforms.	Returns SQL_SCC_ISO92_CLI.	SQLUINTEGER
155	SQL_STATIC_CURSOR_ATTRIBUTES1	Attributes of a static cursor that is supported by the driver. This bitmask contains the first subset of attributes.	Returns 0.	SQLUINTEGER
156	SQL_STATIC_CURSOR_ATTRIBUTES2	Attributes of a static cursor that is supported by the driver. This bitmask contains the second subset of attributes.	Returns 0.	SQLUINTEGER
157	SQL_STRING_FUNCTIONS	The string scalar functions supported by the data source.	Returns the following bit strings: • SQL_FN_STR_ASCII • SQL_FN_STR_CONCAT • SQL_FN_STR_LCASE • SQL_FN_STR_LEFT	SQLUINTEGER

No.	Information type (InfoType)	Description (convention)	Return value	Data type
			• SQL_FN_STR_LENGTH • SQL_FN_STR_LTRIM • SQL_FN_STR_REPLACE • SQL_FN_STR_RIGHT • SQL_FN_STR_RTRIM • SQL_FN_STR_SUBSTRING • SQL_FN_STR_UCASE	
158	SQL_SUBQUERIES	The predicates that support subqueries.	Returns the following bit strings: • SQL_SQ_COMPARISON • SQL_SQ_CORRELATED_SUBQUERIES • SQL_SQ_EXISTS • SQL_SQ_IN • SQL_SQ_QUANTIFIED	SQLUINTEGER
159	SQL_SYSTEM_FUNCTIONS	The system scalar functions supported by the driver and associated data source.	Returns 0.	SQLUINTEGER
160	SQL_TABLE_TERM	A character string containing the data source vendor's name for a table.	Returns "table".	• SQLCHAR * • SQLWCHAR *
161	SQL_TIMEDATE_ADD_INTERVALS	The time stamp interval supported by the driver and associated data source for the TIMESTAMPADD scalar function.	Returns 0.	SQLUINTEGER
162	SQL_TIMEDATE_DIFF_INTERVALS	The time stamp interval supported by the driver and associated data source for the TIMESTAMPDIFF scalar function.	Returns 0.	SQLUINTEGER
163	SQL_TIMEDATE_FUNCTIONS	The date and time scalar functions supported by the driver and associated data source.	Returns the following bit strings: • SQL_FN_TD_CURRENT_DATE • SQL_FN_TD_CURRENT_TIME • SQL_FN_TD_CURRENT_TIMESTAMP • SQL_FN_TD_DAYOFWEEK • SQL_FN_TD_DAYOFYEAR • SQL_FN_TD_EXTRACT	SQLUINTEGER
164	SQL_TXN_CAPABLE	The transactions supported by the driver or data source.	Returns SQL_TC_DDL_COMMIT.	SQLUSMALLINT

No.	Information type (InfoType)	Description (convention)	Return value	Data type
165	SQL_TXN_ISOLATION_OPTION	The transaction isolation levels available from the driver or data source.	Returns the following bit strings: • SQL_TXN_READ_COMMITTED • SQL_TXN_REPEATABLE_READ	SQLUINTEGER
166	SQL_UNION	Support for the UNION clause.	Returns the following bit strings: • SQL_U_UNION • SQL_U_UNION_ALL	SQLUINTEGER
167	SQL_USER_NAME	A character string containing the name that is used in a specific database. This might be different from the login name.	Returns the user name that was used for connection.	• SQLCHAR * • SQLWCHAR *
168	SQL_XOPEN_CLI_YEAR	A character string that indicates the year of publication of the X/Open specification with which the version of the ODBC driver manager fully compiles.	Returns the value that is set by the driver manager.	• SQLCHAR * • SQLWCHAR *
169	SQL_ODBC_API_CONFORMANCE	Value of SQLSMALLINT that indicates the ODBC conformance level.	Returns SQL_OAC_LEVEL1.	SQLUSMALLINT
170	SQL_FETCH_DIRECTION	The supported fetch direction options.	Returns SQL_FD_FETCH_NEXT.	SQLUINTEGER
171	SQL_LOCK_TYPES	The lock types that can be specified in the fLock argument of SQLSetPos.	Returns 0.	SQLUINTEGER
172	SQL_ODBC_SAG_CLI_CONFORMANCE	The compliance level to functions of the SAG specification.	Returns SQL_OSCC_COMPLIANT.	SQLUSMALLINT
173	SQL_ODBC_SQL_CONFORMANCE	The SQL grammar supported by the driver.	Returns SQL_OSC_CORE.	SQLUSMALLINT
174	SQL_OUTER_JOINS	Support status for outer joins of data sources.	Returns F.	• SQLCHAR * • SQLWCHAR *
175	SQL_POS_OPERATIONS	The operations supported by SQLSetPos.	Returns 0.	SQLUINTEGER
176	SQL_POSITIONED_STATEMENTS	The positioned SQL statements supported by the data source.	Returns 0.	SQLUINTEGER
177	SQL_SCROLL_CONCURRENCY	The concurrency control options supported for scrollable cursors.	Returns SQL_SCCO_READ_ONLY.	SQLUINTEGER
178	SQL_STATIC_SENSITIVITY	Whether an application can detect changes made to a static or keyset-driven cursor through the following: • SQLSetPos function • Positioned update or delete statements	Returns 0.	SQLUINTEGER

16.14 Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW

The following table lists and describes the attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW.

Table 16-19: Attributes that can be specified in SQLSetConnectAttr, SQLSetConnectAttrW, SQLGetConnectAttr, and SQLGetConnectAttrW

No.	Attribute	Description (convention)	Support status	Data type
1	SQL_ATTR_ACCESS_MODE	Whether the SQL statements that perform update operations are supported. SQL_MODE_READ_ONLY SQL statements that perform update operations are not supported. SQL_MODE_READ_WRITE SQL statements that perform update operations are supported.	Supports the following value: • SQL_MODE_READ_ONLY • SQL_MODE_READ_WRITE	SQLUINTEGER
2	SQL_ATTR_ANSI_APP	Switching between Unicode and ANSI functions. SQL_AA_TRUE Uses the ANSI function. SQL_AA_FALSE Uses the Unicode function.	Supports the following value: • SQL_AA_TRUE • SQL_AA_FALSE This attribute is used for the driver manager. This attribute cannot be used for applications. If this attribute is used in SQLGetConnectAttr or SQLGetConnectAttrW, an unsupported attribute error is returned.	SQLUINTEGER
3	SQL_ATTR_ASYNC_ENABLE	Whether a function called with a statement on the specified connection is executed asynchronously. SQL_ASYNC_ENABLE_OFF Disables asynchronous execution.	Supports the following value: • SQL_ASYNC_ENABLE_OFF	SQLUINTEGER
4	SQL_ATTR_AUTO_IPD	Whether automatic specification of the IPD after a call to SQLPrepare is supported.	This attribute is not supported.	--
5	SQL_ATTR_AUTOCOMMIT	Whether the automatic commit mode is to be used. SQL_AUTOCOMMIT_OFF The driver uses the manual commit mode. The application must explicitly commit or roll back	Supports the following values: • SQL_AUTOCOMMIT_OFF • SQL_AUTOCOMMIT_ON	SQLUINTEGER

No.	Attribute	Description (convention)	Support status	Data type
		transactions with SQLEndTran. SQL_AUTOCOMMIT_ON The driver uses the automatic commit mode. Each statement is committed immediately after it is executed.		
6	SQL_ATTR_CONNECTION_DEAD	Value indicating the connection status. This attribute can be set in SQLGetConnectAttr or SQLGetConnectAttrW. This attribute cannot be set in SQLSetConnectAttr or SQLSetConnectAttrW.	This attribute is not supported. This attribute is not supported in SQLSetConnectAttr or SQLSetConnectAttrW. If this attribute is used in SQLGetConnectAttr or SQLGetConnectAttrW, an unsupported attribute error is returned.	SQLUINTEGER
7	SQL_ATTR_CONNECTION_TIMEOUT	The amount of time (in seconds) the application is to wait for a response to a connection request. 0 The ODBC driver does not detect a timeout. The value to be specified depends on the timeout control of the client library.	Supports the following values: • 0	SQLUINTEGER
8	SQL_ATTR_CURRENT_CATALOG	Catalog name used by the data source.	This attribute is not supported.	--
9	SQL_ATTR_ENLIST_IN_DTC	Whether Microsoft Component Services use the ODBC driver for distributed transactions.	This attribute is not supported.	--
10	SQL_ATTR_LOGIN_TIMEOUT	The amount of time (in seconds) the application is to wait for a response to a login request. 0 The ODBC driver does not detect a timeout. The value to be specified depends on the timeout control of the client library.	Supports the following values: • 0 If a value other than 0 is set, the HADB ODBC driver rewrites the value to 0, and then returns SQL_SUCCESS_WITH_INFO.	SQLUINTEGER
11	SQL_ATTR_METADATA_ID	How to handle character string arguments in catalog functions. SQL_FALSE The character string arguments in catalog functions are not assumed as identifiers. The character string arguments are case sensitive. Some arguments include a character string search pattern, and other arguments do not.	Supports the following value: • SQL_FALSE If this attribute is used in SQLSetConnectAttr, an unsupported attribute error is returned.	--

No.	Attribute	Description (convention)	Support status	Data type
12	SQL_ATTR_ODBC_CURSORS	How the driver manager uses the ODBC cursor library. SQL_CUR_USE_IF_NEEDED The driver manager uses the ODBC cursor library as needed. SQL_CUR_USE_ODBC The driver manager always uses the ODBC cursor library.	Supports the following values: • SQL_CUR_USE_IF_NEEDED • SQL_CUR_USE_ODBC	SQLINTEGER
13	SQL_ATTR_PACKET_SIZE	Network packet size (in bytes).	This attribute is not supported.	--
14	SQL_ATTR_QUIET_MODE	32-bit window handle.	This attribute is not supported.	--
15	SQL_ATTR_TRACE	Tracing notifications to the driver manager. SQL_OPT_TRACE_OFF Disables traces.	Supports the following values: • SQL_OPT_TRACE_OFF	SQLUINTEGER
16	SQL_ATTR_TRACEFILE	Trace file name.	This attribute is not supported.	--
17	SQL_ATTR_TRANSLATE_LIB	Library name.	This attribute is not supported.	--
18	SQL_ATTR_TRANSLATE_OPTION	A 32-bit flag value that is passed to the translator DLL.	This attribute is not supported.	--
19	SQL_ATTR_TXN_ISOLATION	The transaction isolation level for the current connection. Before calling this function, the application must call SQLEndTran to commit or roll back all open transactions on a connection. SQL_TXN_READ_COMMITTED Sets the transaction isolation level in READ_COMMITTED. SQL_TXN_REPEATABLE_READ Sets the transaction isolation level in REPEATABLE_READ.	Supports the following values: • SQL_TXN_READ_COMMITTED • SQL_TXN_REPEATABLE_READ	SQLUINTEGER
20	SQL_ATTR_HADB_ORDER_MODE	HADB-specific attribute that is not in the ODBC conventions. This attribute sets for the current connection the sort order for character string data in a SELECT statement in which the ORDER BY clause is specified.	Supports the following values: • SQL_HADB_ORDER_MODE_BYTE • SQL_HADB_ORDER_MODE_ISO	SQLUINTEGER

No.	Attribute	Description (convention)	Support status	Data type
		SQL_HADB_ORDER_MODE_BYTE Sort character string data by bytecode. SQL_HADB_ORDER_MODE_ISO Sort character string data by sort code (ISO/IEC 14651:2011 compliance).		

Legend: --: Not applicable.

16.15 Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr

The following table lists and describes the attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr.

Table 16-20: Attributes that can be specified in SQLSetEnvAttr and SQLGetEnvAttr

No.	Attribute	Description (convention)	Support status	Data type
1	SQL_ATTR_CONNECTION_POOLING	Connection pool settings at an environment level.	This attribute is not supported.	--
2	SQL_ATTR_CP_MATCH	How to select a connection from a connection pool.	This attribute is not supported.	--
3	SQL_ATTR_ODBC_VERSION	The ODBC version that the driver complies with. SQL_OV_ODBC3 The driver manager and driver have ODBC 3.0 behaviors as follows: • The driver returns ODBC 3.0 codes for date, time, and time stamp and assumes ODBC 3.0 codes. • When SQLGetDiagField or SQLGetDiagRec is called, the driver returns ODBC 3.0 SQLSTATE code. SQL_OV_ODBC2 The driver manager and driver have ODBC 2.x behaviors as follows: • The driver returns ODBC 2.x codes for dates, times, and time stamps, and assumes ODBC 2.x codes. • When SQLGetDiagField or SQLGetDiagRec is called, the driver returns ODBC 2.x SQLSTATE codes. An application must set this environment attribute before calling a function that specifies an argument of type SQLHENV.	Supports the following value: • SQL_OV_ODBC3 • SQL_OV_ODBC2	SQLUINTEGER
4	SQL_ATTR_OUTPUT_NTS	How the driver returns character string data. SQL_TRUE The driver returns null terminating character string data.	Supports the following value: • SQL_TRUE	SQLUINTEGER

Legend:

--: Not applicable.

16.16 Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW

The following table lists and describes the attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW.

Table 16-21: Attributes that can be specified in SQLSetStmtAttr, SQLSetStmtAttrW, SQLGetStmtAttr, and SQLGetStmtAttrW

No.	Attribute	Description (convention)	Support status	Data type
1	SQL_ATTR_APP_PARAM_DESC	The handle to the APD that can be used for the next call to SQLExecute, SQLExecDirect, or SQLExecDirectW. The value is one of the following: • **Explicitly allocated APD** • SQL_NULL_HDESC Forces automatic allocation of the APD. • **Automatically allocated APD**	Supports the following values: • Explicitly allocated APD. • SQL_NULL_HDESC • Automatically allocated APD	SQLHDESC
2	SQL_ATTR_APP_ROW_DESC	The handle to the ARD that can be used for the next fetch. The value is one of the following: • **Explicitly allocated ARD** • SQL_NULL_HDESC Forces automatic allocation of the ARD. • **Automatically allocated ARD**	Supports the following values: • Explicitly allocated ARD. • SQL_NULL_HDESC • Automatically allocated ARD	SQLHDESC
3	SQL_ATTR_ASYNC_ENABLE	Whether a function called with the statement handle specified in StatementHandle of this function as an argument is to be executed asynchronously. SQL_ASYNC_ENABLE_OFF Does not execute the function asynchronously.	Supports the following value: • SQL_ASYNC_ENABLE_OFF	SQLULEN
4	SQL_ATTR_CONCURRENCY	Specification on cursor concurrency. SQL_CONCUR_READ_ONLY The cursor is read-only. No updates are allowed.	Supports the following value: • SQL_CONCUR_READ_ONLY If a value other than SQL_CONCUR_READ_ONLY is set, the HADB ODBC driver rewrites the value to SQL_CONCUR_READ_ONLY, and then returns SQL_SUCCESS_WITH_INFO.	SQLULEN
5	SQL_ATTR_CURSOR_SCROLLABLE	The level of support that the application requires.	Supports the following value: • SQL_NONSCROLLABLE	SQLULEN

No.	Attribute	Description (convention)	Support status	Data type
		SQL_NONSCROLLABLE Scrollable cursors are not required on the statement handle.	If a value other than SQL_NONSCROLLABLE is set, the HADB ODBC driver rewrites the value to SQL_NONSCROLLABLE, and then returns SQL_SUCCESS_WITH_INFO.	
6	SQL_ATTR_CURSOR_SENSITIVITY	Whether cursors on the statement handle detect changes made to a result set by another cursor. SQL_UNSPECIFIED It is not specified what the cursor type is and whether cursors on the statement handle detect changes made to a result set by another cursor.	Supports the following value: • SQL_UNSPECIFIED If a value other than SQL_UNSPECIFIED is set, the HADB ODBC driver rewrites the value to SQL_UNSPECIFIED, and then returns SQL_SUCCESS_WITH_INFO.	SQLULEN
7	SQL_ATTR_CURSOR_TYPE	Cursor type. SQL_CURSOR_FORWARD_ONLY The cursor scrolls only forward.	Supports the following value: • SQL_CURSOR_FORWARD_ONLY If a value other than SQL_CURSOR_FORWARD_ONLY is set, the HADB ODBC driver rewrites the value to SQL_CURSOR_FORWARD_ONLY, and then returns SQL_SUCCESS_WITH_INFO.	SQLULEN
8	SQL_ATTR_ENABLE_AUTO_IPD	Whether the IPD value is set automatically.	This attribute is not supported.	--
9	SQL_ATTR_FETCH_BOOKMARK_PTR	Pointer to a binary bookmark value.	This attribute is not supported.	--
10	SQL_ATTR_IMP_PARAM_DESC	Handle to the IPD that is allocated the first time the statement handle is allocated.	Supports the following value: • Automatically allocated IPD.	SQLHDESC
11	SQL_ATTR_IMP_ROW_DESC	Handle to the IRD that is allocated the first time the statement handle is allocated.	Supports the following value: • Automatically allocated IRD.	SQLHDESC
12	SQL_ATTR_KEYSET_SIZE	Number of rows in the keyset for a keyset cursor.	This attribute is not supported.	--
13	SQL_ATTR_MAX_LENGTH	Maximum size of data for a character string column or binary column during communication between data sources (between server and host).	Supports the following value: • 0	SQLULEN

No.	Attribute	Description (convention)	Support status	Data type
		0 The data source returns all valid data.		
14	SQL_ATTR_MAX_ROWS	Maximum number of rows that is returned to a SELECT statement during communication between data sources (between server and host). 0 The data source returns all rows.	Supports the following value: • 0	SQLULEN
15	SQL_ATTR_METADATA_ID	How to handle character string arguments of catalog functions. SQL_FALSE The case is significant and the character string arguments of catalog functions are not treated as identifiers.	Supports the following value: • SQL_FALSE	SQLULEN
16	SQL_ATTR_NOSCAN	Whether the driver performs syntax analysis on escape sequences and converts them to a DBMS-specific syntax. SQL_NOSCAN_OFF The driver converts escape sequences to a DBMS-specific syntax. SQL_NOSCAN_ON The driver does not convert escape sequences to a DBMS-specific syntax.	Supports the following value: • SQL_NOSCAN_OFF • SQL_NOSCAN_ON	SQLULEN
17	SQL_ATTR_PARAM_BIND_OFFSET_PTR	A pointer to the offset that is added to the pointer used when dynamic parameters are bound. NULL The driver does not add the value.	Supports the following value: • NULL	SQLULEN *
18	SQL_ATTR_PARAM_BIND_TYPE	Binding orientation to be used for dynamic parameters. SQL_PARAM_BIND_BY_COLUMN Specifies column-wise binding.	Supports the following value: • SQL_PARAM_BIND_BY_COLUMN	SQLULEN
19	SQL_ATTR_PARAM_OPERATION_PTR	An array used to ignore parameters during execution of an SQL statement. NULL The driver does not return parameter status values.	Supports the following value: • NULL	SQLUSMALLINT *
20	SQL_ATTR_PARAM_STATUS_PTR	An array that stores status information for each row of parameter values after a call to SQLExecute, SQLExecDirect, or SQLExecDirectW.	Supports the following value: • NULL	SQLUSMALLINT *

No.	Attribute	Description (convention)	Support status	Data type
		NULL SQLUSMALLINT pointer The driver does not return parameter status values.	• SQLUSMALLINT pointer (other than NULL)	
21	SQL_ATTR_PARAMS_PROCESSED_PTR	Address of the variable to which the driver returns the number of sets of parameters that have been processed (including error sets). NULL SQLULEN pointer The number of parameter sets is not returned.	Supports the following value: • NULL • SQLULEN pointer (other than NULL)	SQLULEN *
22	SQL_ATTR_PARAMSET_SIZE	Number of values for each parameter set.	Supports the following value: • 1 If a value other than 1 is set, the HADB ODBC driver rewrites the value to 1, and then returns SQL_SUCCESS_WITH_INFO.	SQLULEN
23	SQL_ATTR_QUERY_TIMEOUT	The amount of time (in seconds) the application is to wait for an SQL statement to execute.	This attribute is not supported.	--
24	SQL_ATTR_RETRIEVE_DATA	Whether SQLFetch is to acquire data after it has positioned the cursor to the specified location. SQL_RD_ON SQLFetch acquires data after it has positioned the cursor to the specified location.	Supports the following value: • SQL_RD_ON	SQLULEN
25	SQL_ATTR_ROW_ARRAY_SIZE	Number of rows returned by SQLFetch or SQLFetchScroll	Supports the following value: • 1	SQLULEN
26	SQL_ATTR_ROW_BIND_OFFSET_PTR	A pointer to the offset that is added to the pointer used to change the binding of column data. NULL The driver does not add the value.	Supports the following value: • NULL	SQLULEN *
27	SQL_ATTR_ROW_BIND_TYPE	Binding orientation to be used when SQLFetch or SQLFetchScroll is called on the associated statement. SQL_BIND_BY_COLUMN Specifies column-wise binding.	Supports the following value: • SQL_BIND_BY_COLUMN	SQLULEN
28	SQL_ATTR_ROW_NUMBER	Current row number in the result set.	This attribute is not supported.	--
29	SQL_ATTR_ROW_OPERATION_PTR	An array used to ignore rows during a batch operation using SQLSetPos.	Supports the following value: • NULL	SQLUSMALLINT *

No.	Attribute	Description (convention)	Support status	Data type
		NULL The driver does not ignore rows.		
30	SQL_ATTR_ROW_STATUS_PTR	A pointer to an array used to store row status values after a call to SQLFetch or SQLFetchScroll. NULL The driver does not return row status values.	Supports the following value: • NULL • SQLUSMALLINT pointer (other than NULL)	SQLUSMALLINT *
31	SQL_ATTR_ROWS_FETCHED_PTR	A pointer to the buffer that stores the number of fetched rows after a call to SQLFetch or SQLFetchScroll. NULL The driver does not return the number of fetched rows.	Supports the following value: • NULL • SQLULEN pointer (other than NULL)	SQLULEN *
32	SQL_ATTR_SIMULATE_CURSOR	Whether a driver that simulates positioned update and delete statements guarantees that such statements change only a single row.	This attribute is not supported.	--
33	SQL_ATTR_USE_BOOKMARKS	Whether an application uses bookmarks with a cursor.	This attribute is not supported.	--

Legend: --: Not applicable.

16.17 Attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW

The following table lists and describes the attributes that can be specified in SQLGetDescField, SQLGetDescFieldW, SQLSetDescField, and SQLSetDescFieldW.

Table 16-22: Values of header fields in a descriptor

No.	Attribute	Description (convention)	Support status	Data type
1	SQL_DESC_ALLOC_TYPE	Whether the descriptor was allocated automatically or explicitly. SQL_DESC_ALLOC_AUTO The descriptor was allocated automatically by the driver. SQL_DESC_ALLOC_USER The descriptor was allocated explicitly by the application.	Supports the following values: • SQL_DESC_ALLOC_AUTO • SQL_DESC_ALLOC_USER	SQLSMALLINT
2	SQL_DESC_ARRAY_SIZE	ARD Number of rows in the row set (value returned by SQLFetch). APD Number of values for each parameter.	Supports the following value: • 1	SQLULEN
3	SQL_DESC_ARRAY_STATUS_PTR	IRD Pointer to a row status array containing status values after an execution of SQLFetch. IPD Pointer to a parameter status array containing status information for each set of parameter values after an execution of SQLExecute, SQLExecDirect, or SQLExecDirectW. ARD Pointer to an operation array in which the application can set a value to specify that rows are to be ignored for SQLSetPos processing. APD Pointer to a parameter operation array in which the application can set a value to specify that parameter sets are to be ignored during an execution of SQLExecute, SQLExecDirect, or SQLExecDirectW.	Supports the following value: • NULL	SQLUSMALLINT *

No.	Attribute	Description (convention)	Support status	Data type
		NULL The driver does not store status values or status information, nor does it ignore rows or parameter sets.		
4	SQL_DESC_BIND_OFFSET_PTR	Binding offset specified by the application program. This value is added to the delay field during a fetch operation. NULL The driver does not add the value.	Supports the following value: • NULL	SQLLEN *
5	SQL_DESC_BIND_TYPE	Binding orientation for columns or parameters. ARD Binding orientation during an execution of SQLFetch. APD Binding orientation for dynamic parameters. SQL_BIND_BY_COLUMN Specifies column-wise binding.	Supports the following value: • SQL_BIND_BY_COLUMN	SQLINTEGER
6	SQL_DESC_COUNT	Highest number of the record that stores data. The permitted value is 1 or greater.	Supports the following values: • Integer value of 0 or greater	SQLSMALLINT
7	SQL_DESC_ROWS_PROCESSED_PTR	IRD Pointer to the buffer that stores the number of fetched rows after an execution of SQLFetch. IPD Pointer to the buffer that stores the number of parameter sets processed during an execution of SQLExecute, SQLExecDirect, or SQLExecDirectW. NULL The driver does not store the number of fetched rows or the number of parameter sets.	Supports the following value: • NULL	SQLULEN *

Table 16-23: Values of record fields in a descriptor

No.	Attribute	Description (convention)	Support status	Data type
1	SQL_DESC_AUTO_UNIQUE_VALUE	Whether columns are incremented automatically.	Supports the following value: • SQL_FALSE	SQLINTEGER

No.	Attribute	Description (convention)	Support status	Data type
		SQL_FALSE Columns are not incremented automatically.		
2	SQL_DESC_BASE_COLUMN_NAME	Base column name for the result set column.	Supports the following values: • Column name. • "EXP*nnnn*_NO_NAME" (*nnnn*: Unsigned integer in the range from 0001 to 1000) The retrieval item column is not a column specification.	• SQLCHAR * • SQLWCHAR *
3	SQL_DESC_BASE_TABLE_NAME	Base table name for the result set column.	Supports the following value: • Null character string	• SQLCHAR * • SQLWCHAR *
4	SQL_DESC_CASE_SENSITIVE	Whether the column or parameter is treated as being case-sensitive during collations or comparisons. SQL_TRUE The column or parameter is treated as being case-sensitive during collations or comparisons. SQL_FALSE One of the following: The column or parameter is not treated as being case-sensitive. It is a noncharacter column.	Supports the following values: • SQL_TRUE • SQL_FALSE	SQLINTEGER
5	SQL_DESC_CATALOG_NAME	Catalog name for the base table that contains the column.	Supports the following value: • Null character string	• SQLCHAR * • SQLWCHAR *
6	SQL_DESC_CONCISE_TYPE	Concise data type for all data types.	Supports the following value: • The data types supported by this driver.	SQLSMALLINT
7	SQL_DESC_DATA_PTR	Pointer to the buffer that stores the parameter value (APD) or column value (ARD).	Supports the following values: • NULL • Pointer to the buffer	SQLPOINTER
8	SQL_DESC_DATETIME_INTERVAL_CODE	Subcode for the specific datetime or interval data type when the SQL_DESC_TYPE field value is SQL_DATETIME or SQL_INTERVAL. Otherwise, this field is 0.	Supports the following values: • Subcode for the specific datetime or interval data type. • 0	SQLSMALLINT
9	SQL_DESC_DATETIME_INTERVAL_PRECISION	Interval leading precision when the SQL_DESC_TYPE field is SQL_INTERVAL. Otherwise, this field is 0.	Supports the following value: • 0	SQLINTEGER

No.	Attribute	Description (convention)	Support status	Data type
10	SQL_DESC_DISPLAY_SIZE	Maximum number of characters required to display the data from the column.	Supports the following values according to the data type: • SQL_CHAR, SQL_VARCHAR Definition length (in bytes). • SQL_BIGINT, SQL_INTEGER Number of digits. • SQL_DECIMAL Precision (total number of digits) + 2 • SQL_TYPE_DATE, SQL_DATE 10. • SQL_TYPE_TIME, SQL_TIME 8 + (*p* + 1)#. • SQL_TYPE_TIMESTAMP, SQL_TIMESTAMP 19 + (*p* + 1)#. • SQL_DOUBLE 24. • SQL_BINARY, SQL_VARBINARY Definition length (in bytes) × 2.	SQLLEN
11	SQL_DESC_FIXED_PREC_SCALE	Whether the column is an exact numeric column. SQL_FALSE The column is not an exact numeric column with a fixed precision and scale set.	Supports the following value: • SQL_FALSE	SQLSMALLINT
12	SQL_DESC_INDICATOR_PTR	Whether the column or parameter value is NULL.	Supports the following values: • NULL • Pointer to the buffer	SQLLEN *
13	SQL_DESC_LABEL	Column label or title.	Supports the following values: • Column name. • "EXP*nnnn*_NO_NAME" (*nnnn*: Unsigned integer in the range from 0001 to 1000) The retrieval item column is not a column specification.	• SQLCHAR * • SQLWCHAR *
14	SQL_DESC_LENGTH	Maximum or actual length of a character string or a binary data type.	Supports the following value: • Definition length of each data type (in bytes).	SQLULEN

No.	Attribute	Description (convention)	Support status	Data type
15	SQL_DESC_LITERAL_PREFIX	The character or characters that the driver recognizes as a prefix for a data-type literal.	Supports the following value: • Null character string.	• SQLCHAR * • SQLWCHAR *
16	SQL_DESC_LITERAL_SUFFIX	The character or characters that the driver recognizes as a suffix for a data-type literal.	Supports the following value: • Null character string.	• SQLCHAR * • SQLWCHAR *
17	SQL_DESC_LOCAL_TYPE_NAME	Localized name for the data type that differs from the regular name of the data type.	Supports the following value: • Null character string.	• SQLCHAR * • SQLWCHAR *
18	SQL_DESC_NAME	Column alias. If the column alias does not apply, this is the column name.	Supports the following values: • Column name. • "EXP*nnnn*_NO_NAME" (*nnnn*: Unsigned integer in the range from 0001 to 1000) The retrieval item column is not a column specification.	• SQLCHAR * • SQLWCHAR *
19	SQL_DESC_NULLABLE	Whether NULL can be specified for the column. SQL_NULLABLE NULL values can be specified for the column. SQL_NO_NULLS NULL values cannot be specified for the column.	Supports the following values: • SQL_NULLABLE • SQL_NO_NULLS	SQLSMALLINT
20	SQL_DESC_NUM_PREC_RADIX	Whether the data type of the SQL_DESC_TYPE field is an approximate value data type or an exact numeric value data type. 10 Exact numeric value data type. 0 Nonnumeric data type.	Supports the following values: • 10 • 0	SQLINTEGER
21	SQL_DESC_OCTET_LENGTH	Length in bytes of a character string or binary data type.	Supports the following value: • Definition length of each data type (in bytes).	SQLLEN
22	SQL_DESC_OCTET_LENGTH_PTR	Variable that contains the total length in bytes of a dynamic argument (for parameter descriptors) or a bound column value (for row descriptors).	Supports the following values: • NULL • Pointer to the buffer	SQLLEN *
23	SQL_DESC_PARAMETER_TYPE	Parameter type (input, output, or input/output). SQL_PARAM_INPUT Input parameter.	Supports the following values: • SQL_PARAM_INPUT • SQL_PARAM_INPUT_OUTPUT	SQLSMALLINT

No.	Attribute	Description (convention)	Support status	Data type
		SQL_PARAM_INPUT_OUTPUT Input/output parameter.	Replaced with SQL_PARAM_INPUT.	
24	SQL_DESC_PRECISION	Number of digits for an exact numeric type and the number of bits in the mantissa (binary precision) for an approximate value type.	Supports the following values according to the data type: • SQL_BIGINT, SQL_INTEGER Number of digits. • SQL_DECIMAL Precision (total number of digits). • SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_TIME, SQL_TIMESTAMP Fractional seconds precision. • SQL_DOUBLE 15. • Other data type 0.	SQLSMALLINT
25	SQL_DESC_SCALE	Defined scale for decimal and numeric data types. Defined scale.	Supports the following values: • Defined scale. • 0 Nonnumeric data type.	SQLSMALLINT
26	SQL_DESC_SCHEMA_NAME	Schema name of the base table that stores the column.	Supports the following value: • Null character string.	• SQLCHAR * • SQLWCHAR *
27	SQL_DESC_SEARCHABLE	Whether the column can be used in a WHERE clause. SQL_PRED_SEARCHABLE The column can be used in any comparison operator in a WHERE clause.	Supports the following value: • SQL_PRED_SEARCHABLE	SQLSMALLINT
28	SQL_DESC_TABLE_NAME	Name of the base table that stores the column.	Supports the following value: • Null character string.	• SQLCHAR * • SQLWCHAR *
29	SQL_DESC_TYPE	• All data types except datetime and interval. Concise SQL data type or concise C data type. • Datetime and interval data types. Redundant data type (SQL_DATETIME or SQL_INTERVAL).	Supports the following values: • Concise SQL data type. • Concise C data type. • Redundant data type.	SQLSMALLINT
30	SQL_DESC_TYPE_NAME	Data type name that depends on the data source.	Supports the following values:	• SQLCHAR * • SQLWCHAR *

No.	Attribute	Description (convention)	Support status	Data type
		If the data type is unknown, a null character string is specified.	• Data type name that depends on the data source. • Null character string.	
31	SQL_DESC_UNNAMED	Whether a column name or a column alias was specified in the SQL_DESC_NAME field. SQL_NAMED A column name or a column alias was specified in the SQL_DESC_NAME field. SQL_UNNAMED A column name or a column alias was not specified in the SQL_DESC_NAME field.	Supports the following values: • SQL_NAMED • SQL_UNNAMED	SQLSMALLINT
32	SQL_DESC_UNSIGNED	Whether the column's data type is signed or unsigned. SQL_TRUE The column's data type is unsigned or nonnumeric. SQL_FALSE The column's data type is signed.	Supports the following values: • SQL_TRUE • SQL_FALSE	SQLSMALLINT
33	SQL_DESC_UPDATABLE	Whether the column in the result set can be updated. SQL_ATTR_READWRITE_UNKNOWN Whether the column in the result set can be updated is unknown.	Supports the following value: • SQL_ATTR_READWRITE_UNKNOWN	SQLSMALLINT
34	SQL_COLUMN_LENGTH	A field defined in ODBC 2.0.	Supports the following values according to the data type: • SQL_CHAR, SQL_VARCHAR, SQL_BINARY, SQL_VARBINARY Definition length (in bytes). • SQL_BIGINT, SQL_INTEGER Number of digits. • SQL_DECIMAL Precision (total number of digits) + 2 • SQL_TYPE_DATE 10. • SQL_TYPE_TIME, SQL_TIME 8 + (p + 1)#. • SQL_TYPE_TIMESTAMP, SQL_TIMESTAMP	SQLLEN

No.	Attribute	Description (convention)	Support status	Data type
			19 + (*p* + 1)#. • SQL_DOUBLE 24.	
35	SQL_COLUMN_PRECISION	A field defined in ODBC 2.0.	Supports the following values: • Same values as for SQL_DESC_PRECISION.	SQLSMALLINT
36	SQL_COLUMN_SCALE	A field defined in ODBC 2.0.	Supports the following values: • Same values as for SQL_DESC_SCALE.	SQLSMALLINT

#

p indicates the fractional seconds precision with a maximum length of 12 digits.
If there is a fractional second, add the value in parentheses.

16.18 Attributes that can be specified in DiagIdentifier of SQLGetDiagField and SQLGetDiagFieldW

The following table lists and describes the attributes that can be specified for DiagIdentifier.

Table 16-24: Attributes that can be specified for DiagIdentifier (for header fields)

No.	Diagnostic field identifier (DiagIdentifier)	Description (convention)	Return value	Data type
1	SQL_DIAG_CURSOR_ROW_COUNT	Returns the row count in the cursor. This applies only to statement handles. If any other handle is specified, SQL_ERROR is returned.	Always returns 0.	SQLLEN
2	SQL_DIAG_DYNAMIC_FUNCTION	Returns the SQL statement executed by the function. This applies only to statement handles. If any other handle is specified, SQL_ERROR is returned. You can acquire this value immediately after SQLExecute, SQLExecDirect, or SQLExecDirectW.	Returns the following values: • "ALTER TABLE" • "ALTER USER" • "ALTER VIEW" • "CREATE AUDIT" • "CREATE INDEX" • "CREATE SCHEMA" • "CREATE TABLE" • "CREATE USER" • "CREATE VIEW" • "DELETE" • "DROP AUDIT" • "DROP INDEX" • "DROP SCHEMA" • "DROP TABLE" • "DROP USER" • "DROP VIEW" • "GRANT" • "INSERT" • "PURGE CHUNK" • "REVOKE" • "SELECT CURSOR" • "TRUNCATE TABLE" • "UPDATE" • "" The function always returns "" unless it is executed immediately after SQLExecute, SQLExecDirect, or SQLExecDirectW.	• SQLCHAR * • SQLWCHAR *
3	SQL_DIAG_DYNAMIC_FUNCTION_CODE	Returns a value indicating the SQL statement executed by the function. This applies only to statement handles. If any other handle is	Returns the following values: • SQL_DIAG_INSERT • SQL_DIAG_CREATE_TABLE	SQLINTEGER

No.	Diagnostic field identifier (DiagIdentifier)	Description (convention)	Return value	Data type
		specified, SQL_ERROR is returned. You can acquire this value immediately after SQLExecute, SQLExecDirect, or SQLExecDirectW.	• SQL_DIAG_ALTER_TABLE • SQL_DIAG_DROP_TABLE • SQL_DIAG_CREATE_INDEX • SQL_DIAG_DROP_INDEX • SQL_DIAG_CREATE_SCHEMA • SQL_DIAG_DROP_SCHEMA • SQL_DIAG_CREATE_VIEW • SQL_DIAG_DROP_VIEW • SQL_DIAG_UNKNOWN_STATEMENT • SQL_DIAG_SELECT_CURSOR • SQL_DIAG_UPDATE_WHERE • SQL_DIAG_DELETE_WHERE • SQL_DIAG_GRANT • SQL_DIAG_REVOKE The function always returns SQL_DIAG_UNKNOWN_STATEMENT unless it is executed immediately after SQLExecute, SQLExecDirect, or SQLExecDirectW.	
4	SQL_DIAG_NUMBER	Returns the number of available status records.	Returns 0 or 1.	SQLINTEGER
5	SQL_DIAG_RETURNCODE	Returns the return code returned by the ODBC function that was executed immediately before SQLGetDiagField, SQLGetDiagFieldW, SQLGetDiagRec, or SQLGetDiagRecW.	Always returns SQL_SUCCESS.	SQLRETURN
6	SQL_DIAG_ROW_COUNT	Returns the number of rows affected by insert, delete, or update processing. This applies only to statement handles. If any other handle is specified, SQL_ERROR is returned.	Always returns 0.	SQLLEN

Table 16-25: Attributes that can be specified for DiagIdentifier (for record fields)

No.	Diagnostic field identifier (DiagIdentifier)	Description (convention)	Return value	Data type
1	SQL_DIAG_CLASS_ORIGIN	Returns a character string indicating the document that defines the class portion of SQLSTATE.	Returns the following values: • "ODBC 3.0" • "ISO 9075"	• SQLCHAR * • SQLWCHAR *
2	SQL_DIAG_COLUMN_NUMBER	Returns a value indicating a column number in the result set or a parameter number in the parameter set. This applies only to statement handles.	Always returns 0.	SQLINTEGER
3	SQL_DIAG_CONNECTION_NAME	Returns the name of the connection.	Always returns "".	• SQLCHAR * • SQLWCHAR *
4	SQL_DIAG_MESSAGE_TEXT	Returns a message about an error or warning.	A character string beginning with "[Hitachi Advanced Data Binder ODBC Driver]..." is returned. Some information might be added.	• SQLCHAR * • SQLWCHAR *
5	SQL_DIAG_NATIVE	Returns an ODBC driver-specific or data source-specific native error code. If there is no such error code, 0 is returned.	Returns 0 or the value of SQLCODE. For details about SQLCODE, see *Interpreting SQLCODEs* in the manual *HADB Messages*.	SQLINTEGER
6	SQL_DIAG_ROW_NUMBER	Returns a value indicating the column number in the row set or the parameter number in the parameter set. This applies only to statement handles.	Always returns 0.	SQLLEN
7	SQL_DIAG_SERVER_NAME	Returns the server name.	Always returns "".	• SQLCHAR * • SQLWCHAR *
8	SQL_DIAG_SQLSTATE	Returns an SQLSTATE diagnostic code consisting of 5 characters.	Returns a 5-character SQLSTATE diagnostic code stipulated by the ODBC conventions.	• SQLCHAR * • SQLWCHAR *
9	SQL_DIAG_SUBCLASS_ORIGIN	Returns a character string that contains a valid value in the same format as for SQL_DIAG_CLASS_ORIGIN. This value identifies the definition of the subclass portion of SQLSTATE.	Returns the following values: • "ODBC 3.0" • "ISO 9075"	• SQLCHAR * • SQLWCHAR *

17 Troubleshooting

This chapter explains how to troubleshoot use of the ODBC interfaces.

17.1 Information used for troubleshooting

This section describes the information used for troubleshooting when using the ODBC interface (such as BI tools and ODBC modules).

17.1.1 Messages output by BI tools and ODBC modules

If an error occurs when a BI tool or ODBC module (such as Microsoft(R) Excel) accesses the HADB server, the tool or module will output an error message. The output destination might be a dialog box, the status window or message area of the BI tool, or a log file.

The format in which the message is output depends on the specification of the BI tool or ODBC module. It is typical for a message from the error source to be output after the message output by the BI tool or ODBC module. If the output message contains a string like `Hitachi` or `KFAA`, HADB might be the cause of the error.

Important

When an error messages is output to a dialog box or to the message area of a BI tool, you might not be able to retrieve the message for later review. We recommend that you preserve it for troubleshooting purposes by taking a screenshot.

17.1.2 ODBC traces

The driver manager can output information about I/O activity between the application and driver manager to the trace log. This information is called an *ODBC trace*.

Information is written to an ODBC trace file each time the driver manager receives a request from an application. This makes the data in an ODBC trace useful when debugging applications and investigating the cause of errors.

- **Acquiring an ODBC trace**

  To acquire an ODBC trace, on the **Tracing** tab of the ODBC Data Source Administrator, click **Start Tracing Now**.

  The ODBC trace is output to the path specified in **Log File Path**. For details about how to acquire an ODBC trace, see *ODBC Data Source Administrator* in the *MSDN Library*.

Important

Because the I/O activity of trace log information to a file takes place for each request the driver manager receives from an application, processing performance will decrease when this functionality is in use. For this reason, we recommend that you do not leave ODBC trace output permanently enabled.

17.1.3 HADB ODBC driver trace information

HADB ODBC driver trace information is output by the trace output functionality (HADB ODBC driver trace) inherent to the HADB ODBC driver. This functionality outputs information about I/O activity between the driver manager and the HADB ODBC driver.

For details about how to acquire HADB ODBC driver trace information, see 17.3 Settings for outputting HADB ODBC driver trace information. For details about the information that is output, see 17.4 Information output as HADB ODBC driver trace information.

▪ **Output destination of HADB ODBC driver trace information**

HADB ODBC driver trace information is output to one of the following folders:

- The folder specified in **Trace Directory Path** in the **Trace Setup** dialog box of the ODBC Data Source Administrator
- The folder specified in the `ADBODBTRCPATH` environment variable

The file name of an HADB ODBC driver trace file is as follows:

- `adbodbtrace_PID_TID_01.log`
- `adbodbtrace_PID_TID_02.log`

`PID` is the process ID of the application or BI tool that uses the HADB ODBC driver expressed as a hexadecimal. `TID` is the thread ID of the thread started by the process of the application or BI tool that uses the HADB ODBC driver expressed as a hexadecimal.

An HADB ODBC driver trace file is output for each thread of each process that uses the HADB ODBC driver. This means that the number of output files increases in proportion to the number of processes and threads of the applications and BI tools that use the HADB ODBC driver.

The output destination for the trace information switches between `adbodbtrace_PID_TID_01.log` and `adbodbtrace_PID_TID_02.log` as each HADB ODBC driver trace file reaches its maximum size. When switching the output destination from one file to the other, if the new destination file has reached its maximum size, the entire contents of the file are deleted before the output of HADB ODBC driver trace information resumes.

Important

HADB opens and closes the HADB ODBC driver trace file each time it outputs HADB ODBC driver trace information. This is likely to significantly impact processing performance. For this reason, we recommend that you do not leave the output of HADB ODBC driver trace information permanently enabled.

17.1.4 Messages output by the HADB server and HADB client

The messages output by the HADB server and HADB client include information about connections from clients and information about executed SQL statements.

17.1.5 SQL trace information

SQL trace information is trace information for SQL statements the HADB server accepts from applications. For details about SQL trace information, see *Running SQL tracing* in the *HADB Setup and Operation Guide*.

In environments that use the ODBC interface, you can use this information to view the requests made from the HADB ODBC driver to the HADB client, and the information returned by the HADB client to the HADB ODBC driver.

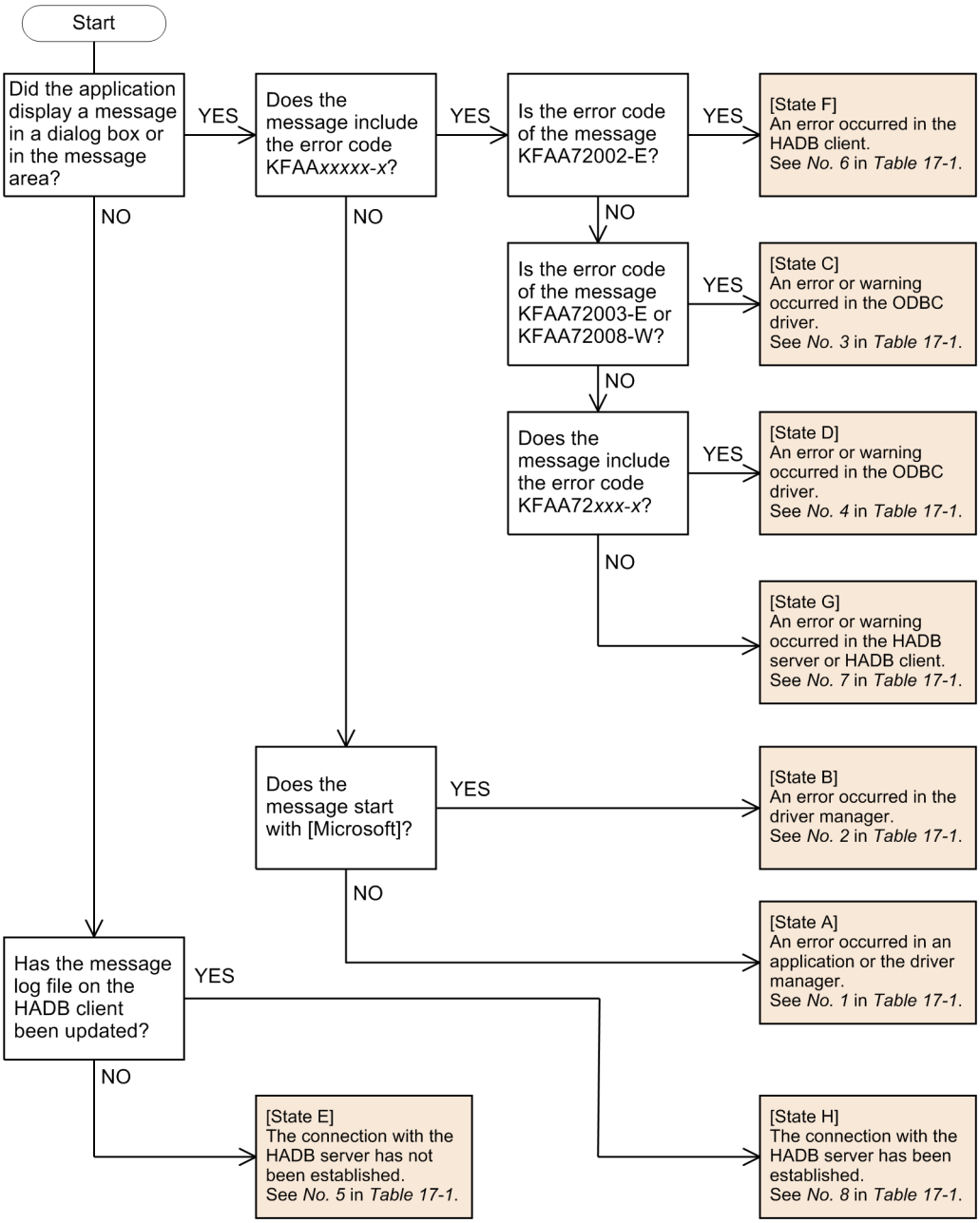
17.2 Troubleshooting procedure

This section describes how to troubleshoot issues that arise when using the ODBC interface.

17.2.1 Handling errors

In the first part of the troubleshooting process, you identify the source of the error. After identifying the source, you then take the appropriate action to resolve the error. The following figure explains the process of identifying the source of an error.

Figure 17-1: Identifying the source of an error



After identifying the source of the error based on this flow chart, use the following table to find out what action to take.

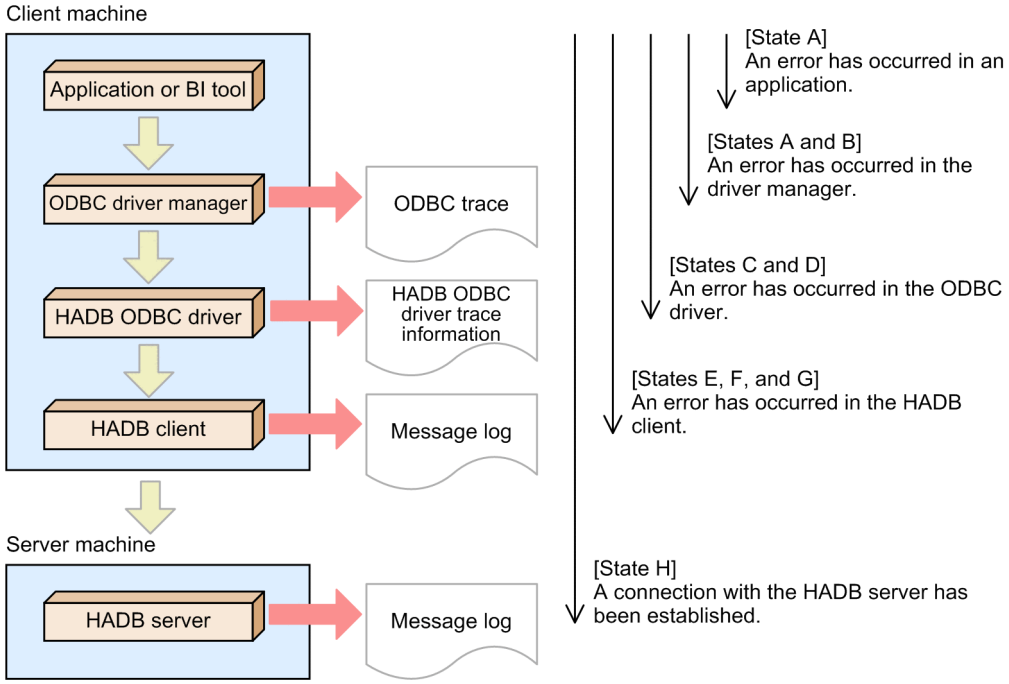
Table 17-1: Handling errors

No.	State	Handling procedure
1	State A An error occurred in an application or the driver manager.	Perform the following steps: • Check the configuration of the application or program. • Make sure that the HADB client was installed correctly. • Make sure that the data sources are registered correctly, and that the connection information is specified correctly. • Check the value output for `SQLSTATE`. Review `SQLSTATE` and message contents based on resources such as *ODBC Programmer's Reference* in the *MSDN Library*. • If the error is reproducible, acquire an ODBC trace and use it to identify the ODBC function that caused the error.
2	State B An error occurred in the driver manager.	Same as the handling procedure for No.1.
3	State C An error or warning occurred in the ODBC driver.	Perform the following steps: • Check the value output for `SQLSTATE`. For details about `SQLSTATE`, see the *ODBC Programmer's Reference* in the *MSDN Library*, and the description of *SQLSTATE* for each ODBC function in 16. ODBC Functions. This error is output according to the ODBC implementation conventions. • If the error is reproducible, acquire HADB ODBC driver trace information and use it to investigate the nature of the error. Your investigation might focus on the sequence in which ODBC functions were executed, or what kind of error occurred in which ODBC function.
4	State D An error or warning occurred in the ODBC driver.	Perform the following steps: • Check the output message, and investigate the cause of the error or warning with reference to the manual *HADB Messages*. • If the error or warning is reproducible, acquire HADB ODBC driver trace information and use it to identify the sequence in which ODBC functions were executed and the ODBC function that caused the error. • This state is often caused by a error related to data conversion or character code conversion. Review the configuration of the BI tool or ODBC module, the display method for retrieval data, and other potential causes. • The ODBC driver might be attempting to access attribute information it does not support. For the support status, see 16.13 Information types that can be specified for InfoType in SQLGetInfo and SQLGetInfoW to 16.18 Attributes that can be specified in DiagIdentifier of SQLGetDiagField and SQLGetDiagFieldW.
5	State E The connection with the HADB server has not been established.	The HADB server might not be accessible. You need to identify the part of the access route where access fails. First, perform the following steps: • Make sure that the access privileges for the output destination folder for message log files and for the message log files themselves are assigned correctly. • Make sure that environment variables are set correctly. • Make sure that the HADB client was installed correctly. If the error still occurs despite all these settings being correct, enable the output of ODBC traces, HADB ODBC driver trace information, and SQL trace information. Then, reproduce the error and analyze the information output in each trace. If there is no error message and nothing is output in the BI tool or ODBC module, the issue might be with the BI tool or ODBC module.
6	State F An error occurred in the HADB client.	Check the error cause code and investigate the cause of the error. For details about the error cause code, see 19.8 Return values of the CLI functions. Potential causes include a path being specified incorrectly in an environment variable or the client definition, or not having the required privileges.
7	State G	Check the message output by HADB, and investigate the cause of the error or warning.

No.	State	Handling procedure
	An error or warning occurred in the HADB server or HADB client.	
8	State H The connection with the HADB server has been established.	The HADB server is accessible. Perform troubleshooting by analyzing the SQL statements and the messages output to the message log file. If you want to acquire detailed information, enable the output of ODBC traces, HADB ODBC driver trace information, and SQL trace information. Then, reproduce the error and analyze the information output in each trace. If there is no error message and nothing is output in the BI tool or ODBC module, the issue might be with the BI tool or ODBC module.

The following figure shows the flow of processing when using the HADB ODBC driver to access the HADB server from a BI tool or ODBC module.

Figure 17-2: Flow of processing when using HADB ODBC driver to access HADB server



If you resolve the error and a different error then occurs, begin the troubleshooting process again starting from identification of the source.

17.2.2 Troubleshooting tips

(1) Tips about viewing ODBC trace information

By viewing ODBC trace information, you can see the requests the driver manager has received from sources such as applications, BI tools, and the ODBC module, and the data that was returned in response to those requests.

An application, BI tool, ODBC module, or other entity to which the driver manager returns an error typically issues the ODBC functions `SQLGetDiagField` and `SQLGetDiagRec` to acquire detailed information about the error. The `SQLSTATE`, `NativeError`, and `MessageText` obtained by `SQLGetDiagRec` is particularly useful information for troubleshooting.

> **Note**
>
> An ODBC trace does not provide information about requests from the HADB ODBC driver, or the responses to those requests.

(2) Tips for identifying the source of an error from message text

When message text is output in relation to an error, you can use the message text to identify the source of the error. This subsection provides tips about this process.

1. Check whether the message text contains a message ID starting with `KFAA`. Check whether the message text includes the tag `[Hitachi Advanced Data Binder] [ODBC Driver]`.

   If either of these conditions is met, the error or warning was generated by HADB (or the HADB ODBC driver).

   - If the message starting with `KFAA` is in the `KFAA72000` range, the error occurred in the HADB ODBC driver.
   - For all other message IDs, the error occurred in the HADB server or HADB client.
   - If the message ID or tag is only partially output, the HADB server or HADB client might have been involved in processing where the error occurred. In this case, acquire the full message from the relevant trace or log file.

2. Check whether the message text includes the tag `[Microsoft] [ODBC Driver Manager]`. If the message text contains this tag, the error or warning was generated by the driver manager.

   In this case, it is likely that HADB (including the HADB ODBC driver) was not processing where the error occurred. Errors with this tag are often critical errors such as sequence errors. If these errors occur, you need to take action such as debugging the application or reviewing the settings of the BI tool or ODBC module.

3. If the message text does not contain any of the keywords in 1. or 2., it is likely that the driver manager and HADB (including the HADB ODBC driver) were not involved in processing where the error occurred. Check the settings of the BI tool or ODBC module for errors in interface or DBMS specification, and review the settings as needed.

(3) Tips about SQLSTATE values

After identifying the source of the error, identify the cause by checking the value of `SQLSTATE`.

- If the driver manager is the error source

  The `SQLSTATE` value returned by the driver manager is based on the ODBC implementation conventions. A list of `SQLSTATE` values can be found in the table of *ODBC Error Codes* in the *ODBC Programmer's Reference* in the *MSDN Library*. There you can find a simple explanation of the cause of the error, and the ODBC function that returned the `SQLSTATE` value. For detailed information, look up the *ODBC API Reference* based on the `SQLSTATE` value and ODBC function you identified.

- If the HADB ODBC driver is the error source

  Identify the cause of the error by checking the `SQLSTATE` value returned by the ODBC function.

  > **Note**
  >
  > Like the `SQLSTATE` value returned by the driver manager, the `SQLSTATE` value returned by the HADB ODBC driver is based on the ODBC implementation conventions. If the cause of the error can be identified in greater detail, HADB returns its own `SQLSTATE` value.

- If the HADB server or HADB client is the error source

The SQLSTATE values returned by the HADB server and HADB client correspond to HADB messages. For details about the correspondence between SQLSTATE values and HADB messages, see *List of SQLSTATE values* in the manual *HADB Messages*.

Identify the cause of the error by checking the content of the corresponding message.

(4) Acquiring SQLSTATE values and message text

Use the following ODBC functions to acquire SQLSTATE values and message text:

- SQLGetDiagField
- SQLGetDiagFieldW
- SQLGetDiagRec
- SQLGetDiagRecW

17.3 Settings for outputting HADB ODBC driver trace information

There are two ways to configure HADB to output HADB ODBC driver trace information:

- Configuration in ODBC Data Source Administrator
- Configuration using environment variables

The following table describes guidelines for selecting the method that is appropriate for your situation.

Table 17-2: Guidelines for selecting configuration method

Usage conditions			Recommended configuration method
Is a data source used when connecting to the HADB server?	**What type of data source is used when connecting to the HADB server?**	**Do you need the ability to dynamically output or stop output of trace information?**	
Yes	User DSN	Yes	Configuration in ODBC Data Source Administrator
		No	Either method can be used
	System DSN	Yes	Configuration in ODBC Data Source Administrator
		No	Either method can be used
	File DSN	Yes	None
		No	Configuration using environment variables
No	Not applicable	Yes	None
		No	Configuration using environment variables

Each method is described in the following subsections.

17.3.1 Configuration in ODBC Data Source Administrator

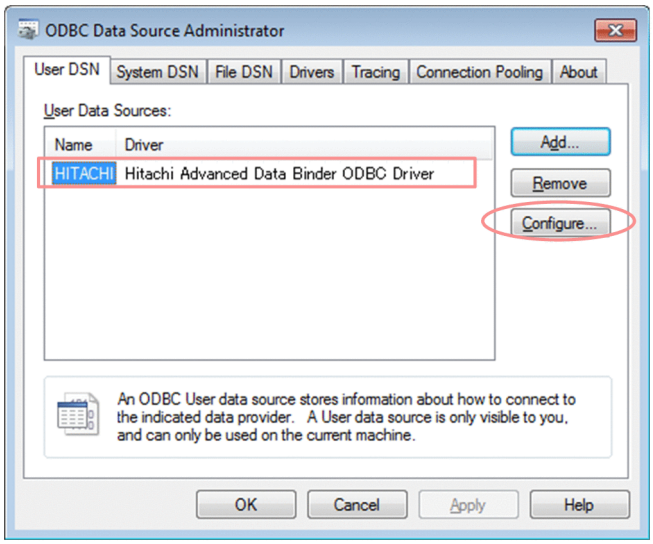
This subsection describes how to configure the output of HADB ODBC driver trace information using ODBC Data Source Administrator.

(1) When outputting HADB ODBC driver trace information

To configure output of HADB ODBC driver trace information:

Procedure

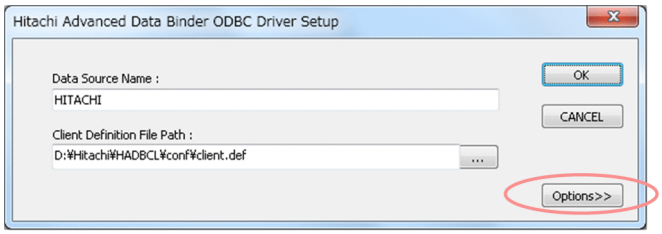
1. Display the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box.
   On the **User DSN** tab of ODBC Data Source Administrator, select the data source that uses the HADB ODBC driver, and then click **Configure...**.



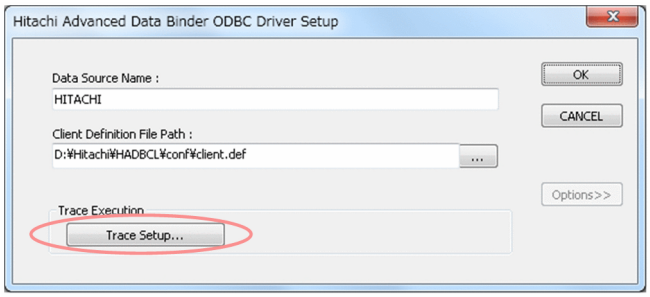
Note

You can also display the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box by performing the same operation on the **System DSN** tab of ODBC Data Source Administrator.

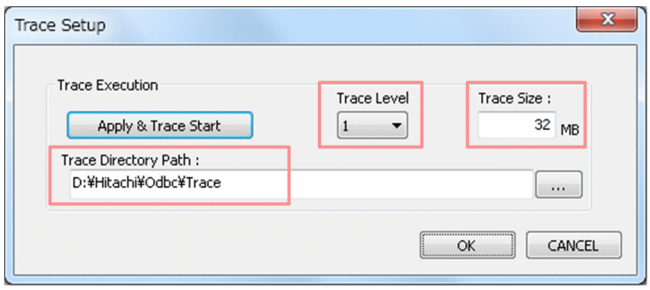
2. In the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box, click **Options>>**.



3. Click **Trace Setup...**.



4. In the **Trace Setup** dialog box, enter the settings that relate to output of HADB ODBC driver trace information.



Specify the following settings:

- `Trace Level`

  Select a trace level. For details about trace levels, see 17.4.1 About trace levels.

- `Trace Size`

  Specify the maximum size (in MB) of each HADB ODBC driver trace file. You can specify a value in the range from `32` to `1,024`.

  If you specify no value or an invalid value, 256 is assumed.

- `Trace Directory Path`

  Specify the absolute path of the folder in which to store HADB ODBC driver trace files.

  Click the **...** button and select a folder in the dialog box that appears.

  You must specify a folder for which the execution user has access privileges.

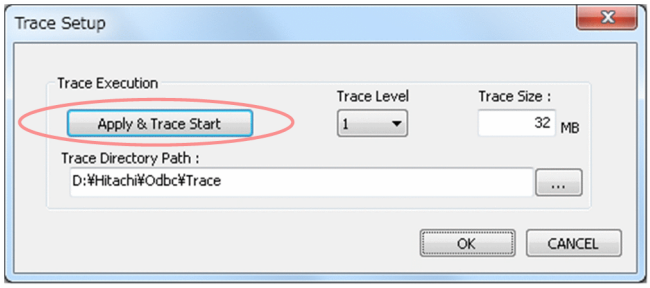
  The path name you specify cannot be more than 210 bytes in length.

  Important

  HADB ODBC driver trace information will not be output in the following circumstances:

  - No path is specified in **Trace Directory Path**.
  - An invalid path is specified in **Trace Directory Path**.
  - A path that is 211 bytes or longer is specified in **Trace Directory Path**.

5. Click **Apply & Trace Start**.

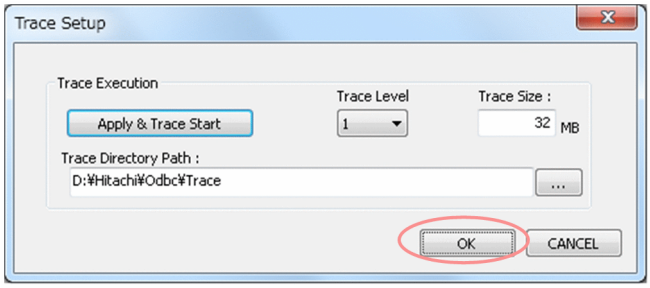


The output of HADB ODBC driver trace information starts.

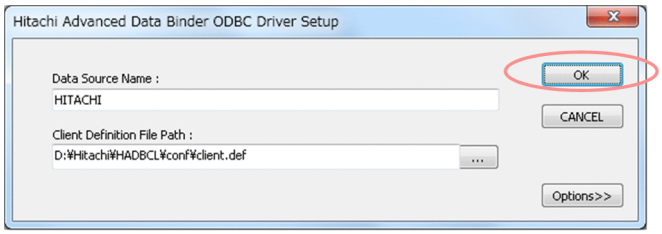
Note

- The **Apply & Trace Start** button becomes available when you complete step 4.
- If you decide not to apply these settings, click **CANCEL** instead of the **Apply & Trace Start** button.

6. Click **OK**.



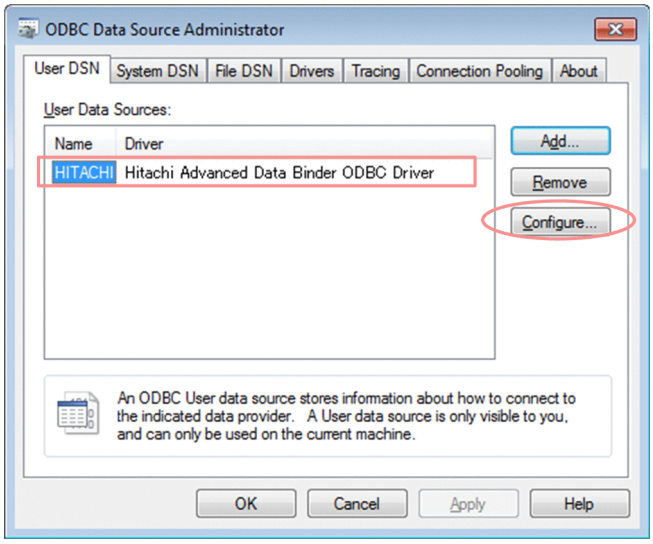
7. In the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box, click **OK**.



(2) When stopping output of HADB ODBC driver trace information

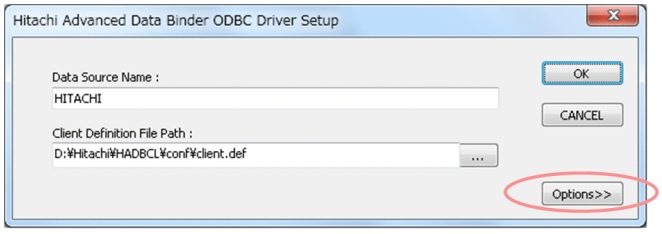
1. Display the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box.

   On the **User DSN** tab of ODBC Data Source Administrator, select the data source that uses the HADB ODBC driver, and then click **Configure...**.

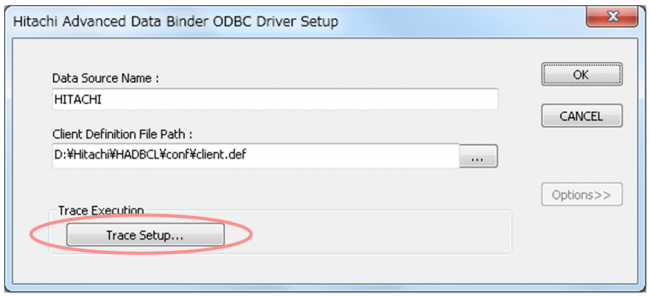


> **Note**
>
> You can also display the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box by performing the same operation on the **System DSN** tab of ODBC Data Source Administrator.

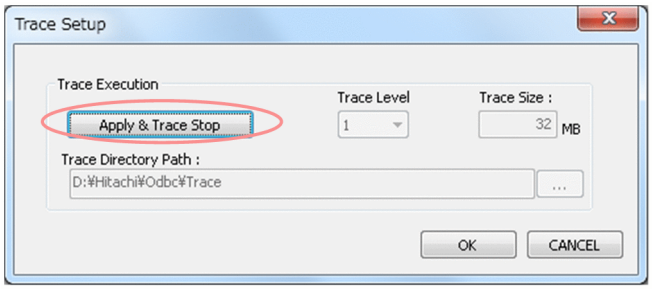
2. In the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box, click **Options>>**.



3. Click **Trace Setup...**.

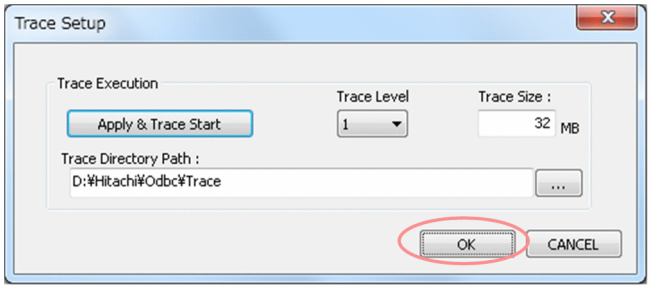


4. Click **Apply & Trace Stop**.

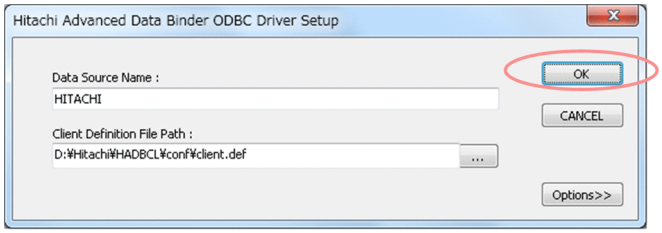


The output of HADB ODBC driver trace information stops.

5. Click **OK**.



6. In the **Hitachi Advanced Data Binder ODBC Driver Setup** dialog box, click **OK**.



17.3.2 Configuration using environment variables

In the following circumstances, configure the output of HADB ODBC driver trace information by using environment variables:

- You use the `SQLDriverConnect`, `SQLDriverConnectW`, `SQLBrowseConnect`, or `SQLBrowseConnectW` function to connect using an ODBC driver name (specifying `DRIVER` as a connection attribute) without using a data source.
- You use a file DSN in a connection function such as `SQLConnect` or `SQLConnectW`.

Specify the following environment variables:

- ADBODBTRC
  Specify whether to output HADB ODBC driver trace information.
- ADBODBTRCSIZE
  Specify the maximum size of each HADB ODBC driver trace file.
- ADBODBTRCPATH
  Specify the folder in which to store the HADB ODBC driver trace files.
  You must specify a folder for which the execution user has access privileges.
- ADBODBTRCLV
  Specify the trace level of the HADB ODBC driver trace.

For details about how to specify each environment variable, see 4.3.1 HADB client for Windows.

Note that the value of an environment variable will be invalid if you specify it in the following way. In this case, the default value of the environment variable applies.

```
export ADBODBTRCSIZE=
```

Important

If you change the value of these environment variables while an application is accessing the HADB server using the ODBC driver, the new value will not apply to that application process.

17.3.3 Relative priority of configuration in ODBC Data Source Administrator and environment variables

If you configure the output of HADB ODBC driver trace information in ODBC Data Source Administrator and in environment variables, the settings in ODBC Data Source Administrator generally have priority. However, in the following scenario, the settings in ODBC Data Source Administrator will not apply:

- With YES specified in the ADBODBTRC environment variable, you stop the output of HADB ODBC driver trace information in ODBC Data Source Administrator by clicking **Apply & Trace Stop**

In this case, the setting in the ADBODBTRC environment variable takes effect, and output of HADB ODBC driver trace information will continue.

17.4 Information output as HADB ODBC driver trace information

This section describes the information output as HADB ODBC driver trace information.

The trace level you select determines how much information is output.

17.4.1 About trace levels

You can select which information is output as HADB ODBC driver trace information by specifying a trace level. There are two trace levels. Trace level 1 outputs simple trace information, and trace level 2 outputs information that is more detailed. The following table lists the relationship between trace levels and the information output as HADB ODBC driver trace information.

Table 17-3: Relationship between trace levels and output information

Output information	Trace level	
	Trace level 1	Trace level 2
Access type	N	Y
Function name	Y	Y
Arguments	N	Y
Handle	Y	Y
Time	Y	Y
Execution result	Y	Y
SQLSTATE	Y	Y
Error message	Y	Y
Executed SQL	Y	Y
Additional information	Y	Y

Legend:

Y: Output.

N: Not output.

When the trace level is 1, a maximum of 256 bytes of information is output per line. When the trace level is 2, all the information is output. Selecting trace level 2 can impact performance due to the greater amount of information that is output.

The timing with which HADB ODBC driver trace information is output differs for each trace level, as follows:

- Trace level 1
  Information is output immediately before the ODBC function terminates.
- Trace level 2
  Information is output when the ODBC function starts and immediately before it terminates.

Note

To output detailed troubleshooting information, you must select trace level 2. However, be aware of the impact this will have on performance.

17.4.2 Information output when trace level is 1

The following is an example of the information output when 1 is selected as the trace level.

▪ Output example (trace level 1)

```
[Trace Start Time] 2015/04/07 14:29:26.000
[Process ID] 345
[Module Name] Sample01
[Platform] Win32
[ODBC environment variables]
  ADBCLTLANG(SJIS)
  ADBCODBTRC(YES)
  ADBCODBTRCSIZE(256)
  ADBCODBTRCPATH(C:\HADB_odb\ODBC35\project\HADBCL\spool\)
  ADBCODBTRCLV(1)
  ADBCODBAPMODE(ACCESS)

[RETURN Value]
  0 : SQL_SUCCESS
  1 : SQL_SUCCESS_WITH_INFO
  2 : SQL_STILL_EXECUTING
 99 : SQL_NEED_DATA
100 : SQL_NO_DATA
 -1 : SQL_ERROR
 -2 : SQL_INVALID_HANDLE

FUNCTION            HANDLE     START-TIME              END-TIME                RETURN SQL   CON_ID CON_NUM    OPTION
                                                                                   STATE

------------------- ---------- ----------------------- ----------------------- ------ ----- ------ --------- ------------------------------------------------
SQLAllocHandle      0x00000000 2015/06/11 18:20:15.838 2015/06/11 18:20:15.839      0 00000      *         * HandleType=SQL_HANDLE_ENV, Address=0x001a576c
SQLSetEnvAttr       0x017b5af0 2015/06/11 18:20:15.856 2015/06/11 18:20:15.857      0 00000      *         * Attribute=SQL_ATTR_ODBC_VERSION
SQLAllocHandle      0x017b5af0 2015/06/11 18:20:15.859 2015/06/11 18:20:15.860      0 00000      *         * HandleType=SQL_HANDLE_DBC, Address=0x001a575c
SQLGetInfoW         0x017b5de8 2015/06/11 18:20:15.862 2015/06/11 18:20:15.862      0 00000      *         * InfoType=SQL_DRIVER_ODBC_VER
SQLSetConnectAttrW  0x017b5de8 2015/06/11 18:20:15.863 2015/06/11 18:20:15.863      0 00000      *         * Attribute=SQL_ATTR_ANSI_APP
SQLConnectW         0x017b5de8 2015/06/11 18:20:15.865 2015/06/11 18:20:16.954      0 00000      1        22 ServerNameW="HADB_NT30", UserNameW="HADBUSER"
*** Connect Info ***********************************************************************
[DataSourceName] HADB_NT30
[Client Definition File Path] D:\work\HADB\ODB\0301\HADBCL\conf\client.def
[Hitachi Advanced Data Binder ODBC Driver Version] 03.01
[ProcessID] 1752
******************************************************************************
SQLPrepareW         0x017b6770 2015/06/11 18:20:17.82  2015/06/11 18:20:17.126      0 00000      1        22 tran_id=49, stmt_hdl=1, sql_serial_num=0
 <SQL>INSERT INTO T1 VALUES(100, 'ABCDE', ?)
SQLExecute          0x017b6770 2015/06/11 18:20:17.157 2015/06/11 18:20:17.158     -1 07002      1        22 tran_id=49, stmt_hdl=1, sql_serial_num=0
 <Message>[Hitachi Advanced Data Binder][ODBC Driver]KFAA72003-E An error occurred in the ODBC driver. (SQLSTATE = 07002)
```

When using trace level 1, each line of characters can contain a maximum of 256 bytes. This restriction does not apply to the SQL statement whose execution was requested and any diagnostic messages, which are always output in full.

The following table explains each item of information output at trace level 1.

Table 17-4: Information output as HADB ODBC driver trace information (trace level 1)

No.	Output item	Description
1	`[Trace Start Time]`	The date and time at which output of the HADB ODBC driver trace information started. If HADB could not acquire the time, 0 is output in all date and time fields.
2	`[Process ID]`	The process ID
3	`[Module Name]`	The name of the module that uses the HADB ODBC driver. If HADB cannot identify the module name, `unknown` is output.
4	`[Platform]`	The platform on which the HADB ODBC driver is operating
5	`[ODBC environment variables]`	The values of the environment variables used by the HADB ODBC driver. The values output here are not necessarily the values specified in the environment variables. They are the values actually being used by the HADB ODBC driver, which might be the default values or a value explicitly specified in ODBC Data Source Administrator.

No.	Output item	Description
		• ADBCLTLANG The character encoding used by the HADB client • ADBODBTRC Whether HADB ODBC driver trace information is to be output. • ADBODBTRCSIZE The maximum size of each HADB ODBC driver trace file • ADBODBTRCPATH The absolute path of the folder in which HADB ODBC driver trace files are stored • ADBODBTRCLV The specification of the trace level • ADBODBAPMODE The application mode of the HADB ODBC driver For details about the preceding environment variables, see 4.3.1 HADB client for Windows.
6	[RETURN Value]	An explanation of the return values of the ODBC function. This information provides definitions that correspond to the integer values of the SQLRETURN data type output for RETURN (No. 11).
7	FUNCTION	The name of the ODBC function
8	HANDLE	The value of the handle passed as an argument to the ODBC function
9	START-TIME	The time at which the ODBC function started executing. If HADB could not acquire the time, 0 is output in all date and time fields.
10	END-TIME	The time at which the ODBC function finished executing. If HADB could not acquire the time, 0 is output in all date and time fields.
11	RETURN	The SQLRETURN value that is the execution result of the ODBC function. For details about the meaning of each value, see the [RETURN Value] section that precedes it in the trace information.
12	SQLSTATE	The value of SQLSTATE. This information is only output if executing the ODBC function results in output of a SQLSTATE value.
13	CON_ID	The connection number. This is the same as the connection number output in SQL trace information. If the ODBC function had not yet connected to the HADB server, * is output.
14	CON_NUM	The connection sequence number. This is the same as the connection sequence number output in SQL trace information. If the ODBC function had not yet connected to the HADB server, * is output.
15	OPTION	Optional information. For details, see Table 17-5: Optional information output for OPTION. The information output as optional information differs between ODBC functions. If the ODBC function does not have optional information, * is output. If HADB cannot identify the value for information output as a symbol name, unknown is output instead of the symbol value.
16	Connect Info	The information output when the connection with the HADB server was established.
17	[DataSourceName]	The name of the data source to which the HADB ODBC is connected. If the driver is not using a data source or is using a file DSN, * is output.
18	[Client Definition File Path]	The path of the client definition file

No.	Output item	Description
19	[Hitachi Advanced Data Binder ODBC Driver Version]	The version of the HADB ODBC driver
20	[ProcessID]	The process ID of the HADB client
21	<SQL>	The SQL statement whose execution was requested. If the following ODBC functions were executed, <SQL> shows the SQL statement passed as an argument of the ODBC function. • SQLPrepare • SQLExecDirect
22	<Message>	A message that provides diagnostic information. A message is output when the execution result of the ODBC function is one of the following: • SQL_ERROR • SQL_SUCCESS_WITH_INFO • SQL_NEED_DATA (when the function is SQLBrowseConnect)

The following table lists the optional information output for OPTION in the preceding table.

Table 17-5: Optional information output for OPTION

No.	Type	ODBC function	Optional information output for OPTION
1	Connections with data sources	SQLAllocHandle	• HandleType (*symbol value*) • Address: The address value returned to the OutputHandlePtr parameter
2		SQLConnect(W)	• ServerName • UserName
3		SQLDriverConnect(W)	InConnectionString A single asterisk * is output in place of the password.
4		SQLBrowseConnect(W)	
5	Acquisition of driver and data source information	SQLDataSources(W)	None
6		SQLDrivers(W)	
7		SQLGetInfo(W)	InfoType
8		SQLGetFunctions	FunctionID
9		SQLGetTypeInfo(W)	DataType
10	Setting and acquiring driver options	SQLSetConnectAttr(W)	Attribute
11		SQLGetConnectAttr(W)	
12		SQLSetEnvAttr	
13		SQLGetEnvAttr	
14		SQLSetStmtAttr(W)	
15		SQLGetStmtAttr(W)	
16	Setting descriptor values	SQLGetDescField(W)	• RecNumber • FieldIdentifier (*symbol value*)

No.	Type	ODBC function	Optional information output for OPTION
17		SQLGetDescRec(W)	• RecNumber • *Name
18		SQLSetDescField(W)	• RecNumber • FieldIdentifier
19		SQLSetDescRec	• RecNumber • Type • SubType
20		SQLCopyDesc	• Source: Value of SourceDescHandle • Target: Value of TargetDescHandle
21	Creating SQL requests	SQLPrepare(W)	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
22		SQLBindParameter	• ParameterNumber • ValueType • ParameterType
23		SQLGetCursorName(W)	None
24		SQLSetCursorName(W)	
25		SQLDescribeParam	ParameterNumber
26		SQLNumParams	*ParameterCountPtr
27	SQL execution	SQLExecute	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
28		SQLExecDirect(W)	
29		SQLNativeSql(W)	OutStatementText
30		SQLParamData	None
31		SQLPutData	StrLen or Ind
32	Acquiring execution results and execution result information	SQLRowCount	*RowCountPtr
33		SQLNumResultCols	*ColumnCountPtr
34		SQLDescribeCol(W)	ColumnNumber
35		SQLColAttribute(W)	• ColumnNumber • FieldIdentifier (*symbol value*)
36		SQLBindCol	• ColumnNumber • TargetType (*symbol value*)
37		SQLFetch	None
38		SQLFetchScroll	
39		SQLGetData	• ColumnNumber • TargetType (*symbol value*)
40		SQLSetPos	None
41		SQLBulkOperations	

No.	Type	ODBC function	Optional information output for OPTION
42		SQLMoreResults	
43		SQLGetDiagField(W)	DiagIdentifier
44		SQLGetDiagRec(W)	None
45	Acquiring system information for data sources	SQLColumnPrivileges(W)	None
46		SQLColumns(W)	
47		SQLForeignKeys(W)	
48		SQLPrimaryKeys(W)	
49		SQLProcedureColumns(W)	
50		SQLProcedures(W)	
51		SQLSpecialColumns(W)	
52		SQLStatistics(W)	
53		SQLTablePrivileges(W)	
54		SQLTables(W)	
55	Terminating SQL execution	SQLFreeStmt	Option When Option=SQL_CLOSE is specified, the following information is also output: • tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
56		SQLCloseCursor	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
57		SQLCancel	None
58		SQLEndTran	• CompletionType • tran_id: Transaction ID
59	Disconnecting from data sources	SQLDisconnect	None
60		SQLFreeHandle	

Notes:

- For variables whose names are prefixed with *, HADB outputs the output value. For all other variables, the input value is output.
- (*symbol value*) means that the symbol value is output as a character string. In all other cases, a numeric value is output. If the value is unknown, unknown is output.
- In some circumstances, such as when the transaction has been settled, HADB might be unable to acquire tran_id, stmt_htdl, and sql_serial_num. In this case, * is output for each value.
- When outputting two or more pieces of optional information, each piece of information is separated by a comma.

17.4.3 Information output when trace level is 2

The following is an example of the information output when 2 is selected as the trace level.

▪ Output example (trace level 2)

```
[Trace Start Time] 2015/04/07 14:29:26.000
[Process ID] 345
[Module Name] Sample01
[Platform] Win32
[ODBC environment variables]
  ADBCLTLANG(SJIS)
  ADBODBTRC(YES)
  ADBODBTRCSIZE(256)
  ADBODBTRCPATH(C:\HADB_odb\ODBC35\project\HADBCL\spool\)
  ADBODBTRCLV(1)
  ADBODBAPMODE(ACCESS)

ACC FUNCTION             HANDLE     START-TIME             END-TIME               RETURN                 SQL   CON_ID CON_NUM
ESS                                                                                                     STATE

--- -------------------- ---------- ---------------------- ---------------------- ---------------------- ----- ------ ----------
[E] SQLAllocHandle       0x00000000 2015/06/11 19:23:04.865                       *                      *     *      *          *
 [Input]  HandleType(SQLSMALLINT)       = 1(SQL_HANDLE_ENV)

          InputHandle(SQLHANDLE)        = 0x00000000
          OutputHandlePtr(SQLHANDLE)    = 0x0020576c
[R] SQLAllocHandle       0x00000000 2015/06/11 19:23:04.865 2015/06/11 19:23:04.884             SQL_SUCCESS 00000      *          *
 [Output] OutputHandlePtr(SQLHANDLE)    = 0x0020576c(0x011b1290)

[E] SQLSetEnvAttr        0x011b1290 2015/06/11 19:23:04.885                       *                      *     *      *          *
 [Input]  EnvironmentHandle(SQLHENV)    = 0x011b1290
          Attribute(SQLINTEGER)         = 200 (SQL_ATTR_ODBC_VERSION)
          ValuePtr(SQLPOINTER)          = 3 (SQL_OV_ODBC3)
          StringLength(SQLINTEGER)      = 0
[R] SQLSetEnvAttr        0x011b1290 2015/06/11 19:23:04.885 2015/06/11 19:23:04.886             SQL_SUCCESS 00000      *          *
[E] SQLAllocHandle       0x011b1290 2015/06/11 19:23:04.886                       *                      *     *      *          *
 [Input]  HandleType(SQLSMALLINT)       = 2(SQL_HANDLE_DBC)
          InputHandle(SQLHANDLE)        = 0x011b1290
          OutputHandlePtr(SQLHANDLE)    = 0x0020575c
[R] SQLAllocHandle       0x011b1290 2015/06/11 19:23:04.886 2015/06/11 19:23:04.887             SQL_SUCCESS 00000      *          *
 [Output] OutputHandlePtr(SQLHANDLE)    = 0x0020575c(0x011b1588)
[E] SQLSetConnectAttrW   0x011b1588 2015/06/11 19:23:04.889                       *                      *     *      *          *
 [Input]  ConnectionHandle(SQLHDBC)     = 0x011b1588
          Attribute(SQLINTEGER)         = 115(SQL_ATTR_ANSI_APP)
          ValuePtr(SQLPOINTER)          = 0x00000001(1)
          StringLength(SQLINTEGER)      = -5
[R] SQLSetConnectAttrW   0x011b1588 2015/06/11 19:23:04.889 2015/06/11 19:23:04.889             SQL_SUCCESS 00000      *          *
[E] SQLConnectW          0x011b1588 2015/06/11 19:23:04.890                       *                      *     *      *          *
 [Input]  ConnectionHandle(SQLHDBC)     = 0x011b1588
          ServerNameW                   = 0x00205044("HADB_NT30")
          NameLength1(SQLSMALLINT)      = 9
          UserNameW                     = 0x00201300("HADBUSER")

          NameLength2(SQLSMALLINT)      = -3
          AuthenticationW               = 0x00201330("*")
          NameLength3(SQLSMALLINT)      = -3
[R] SQLConnectW          0x011b1588 2015/06/11 19:23:04.890 2015/06/11 19:23:05.117             SQL_SUCCESS 00000      1          25
*** Connect Info ********************************************************************************
[DataSourceName] HADB_NT30
[Client Definition File Path] D:\work\HADB\ODB\0301\HADBCL\conf\client.def
[Hitachi Advanced Data Binder ODBC Driver Version] 03.01.0000
[ProcessID] 8536
*****************************************************************************************
[E] SQLAllocHandle       0x011b1588 2015/06/11 19:23:05.176                       *                      *     *      1          25
 [Input]  HandleType(SQLSMALLINT)       = 3(SQL_HANDLE_STMT)
          InputHandle(SQLHANDLE)        = 0x011b1588
          OutputHandlePtr(SQLHANDLE)    = 0x00205fd8
[R] SQLAllocHandle       0x011b1588 2015/06/11 19:23:05.176 2015/06/11 19:23:05.181             SQL_SUCCESS 00000      1          25
 [Output] OutputHandlePtr(SQLHANDLE)    = 0x00205fd8(0x011b1e80)
[E] SQLPrepareW          0x011b1e80 2015/06/11 19:23:05.241                       *                      *     *      1          25
 [Input]  StatementHandle(SQLHSTMT)     = 0x011b1e80
          StatementTextW(SQLWCHAR *)    = 0x014236f8("INSERT INTO T1 VALUES(100, 'ABCDE', ?)")
          TextLength(SQLINTEGER)        = -3
[R] SQLPrepareW          0x011b1e80 2015/06/11 19:23:05.241 2015/06/11 19:23:05.245             SQL_SUCCESS 00000      1          25
 [SYSTEM] tran_id                       = 57

          stmt_hdl                      = 1
          sql_serial_num                = 0
[E] SQLExecute           0x011b1e80 2015/06/11 19:23:05.283                       *                      *     *      1          25
 [Input]  StatementHandle(SQLHSTMT)     = 0x011b1e80
[R] SQLExecute           0x011b1e80 2015/06/11 19:23:05.283 2015/06/11 19:23:05.284               SQL_ERROR 07002      1          25
 [SYSTEM] tran_id                       = 57
          stmt_hdl                      = 1
          sql_serial_num                = 0
 <Message>[Hitachi Advanced Data Binder][ODBC Driver]KFAA72003-E An error occurred in the ODBC driver. (SQLSTATE = 07002)
```

The following table explains each item of information output at trace level 2.

Table 17-6: Information output as HADB ODBC driver trace information (trace level 2)

No.	Output item	Description
1	`[Trace Start Time]`	The date and time at which output of the HADB ODBC driver trace information started. If HADB could not acquire the time, 0 is output in all date and time fields.
2	`[Process ID]`	The process ID

No.	Output item	Description
3	[Module Name]	The name of the module that uses the HADB ODBC driver. If HADB cannot identify the module name, unknown is output.
4	[Platform]	The platform on which the HADB ODBC driver is operating
5	[ODBC environment variables]	The values of the environment variables used by the HADB ODBC driver. The values output here are not necessarily the values specified in the environment variables. They are the values actually being used by the HADB ODBC driver, which might be the default values or a value explicitly specified in ODBC Data Source Administrator. • ADBCLTLANG The character encoding used by the HADB client • ADBODBTRC Whether HADB ODBC driver trace information is to be output. • ADBODBTRCSIZE The maximum size of each HADB ODBC driver trace file • ADBODBTRCPATH The absolute path of the folder in which HADB ODBC driver trace files are stored • ADBODBTRCLV The specification of the trace level • ADBODBAPMODE The application mode of the HADB ODBC driver For details about the preceding environment variables, see 4.3.1 HADB client for Windows.
6	ACCESS	The access type • [E]: Called by the function. • [R]: Returned from the function.
7	FUNCTION	The name of the ODBC function
8	HANDLE	The value of the handle passed as an argument to the ODBC function. In 32-bit environments, the handle will be output a string of eight characters prefixed with the two characters 0x. In 64-bit environments, the handle is a 16-character hexadecimal value.
9	START-TIME	The time at which the ODBC function started executing. If HADB could not acquire the time, 0 is output in all date and time fields.
10	END-TIME	The time at which the ODBC function finished executing. If HADB could not acquire the time, 0 is output in all date and time fields. When the access type is [E], * is output.
11	RETURN	The symbol name that represents the value of the SQLRETURN data type that is the execution result of the ODBC function. If the value is unknown, unknown is output instead of the symbol name. When the access type is [E], * is output.
12	SQLSTATE	The value of SQLSTATE. This information is only output if executing the ODBC function results in output of a SQLSTATE value. When the access type is [E], * is output.
13	CON_ID	The connection number. This is the same as the connection number output in SQL trace information. If the ODBC function had not yet connected to the HADB server, * is output.
14	CON_NUM	The connection sequence number. This is the same as the connection sequence number output in SQL trace information.

No.	Output item	Description
		If the ODBC function had not yet connected to the HADB server, * is output.
15	[Input]	The input parameters. Input parameters are output in the following format: • *variable-name (data-type-name) = value (symbol-name-or-referenced-data)* Parenthesized information is omitted in some cases. If the value is unknown, unknown is output instead of the symbol name.
16	[Output]	The output parameters. Output parameters are output in the following format: • *variable-name (data-type-name) = value (symbol-name-or-referenced-data)* Parenthesized information is omitted in some cases. If the value is unknown, unknown is output instead of the symbol name. This information is not output if the ODBC function does not have output parameters.
17	Connection user name and password	The user name and password used to establish the connection. A single asterisk * is output in place of the password.
18	Connect Info	The information output when the connection with the HADB server was established
19	[DataSourceName]	The name of the data source to which the HADB ODBC is connected. If the driver is not using a data source or is using a file DSN, * is output.
20	[Client Definition File Path]	The path of the client definition file
21	[Hitachi Advanced Data Binder ODBC Driver Version]	The version of the HADB ODBC driver.
22	[ProcessID]	The process ID of the HADB client
23	[SYSTEM]	Information specific to HADB. For details, see Table 17-7: Information output in [SYSTEM]. The information output for SYSTEM differs between ODBC functions.
24	<Message>	A message that provides diagnostic information. A message is output when the execution result of the ODBC function is one of the following: • SQL_ERROR • SQL_SUCCESS_WITH_INFO • SQL_NEED_DATA (when the function is SQLBrowseConnect)

The following table lists the information output for [SYSTEM] in the preceding table.

Table 17-7: Information output in [SYSTEM]

No.	Type	ODBC function name	Information output in [SYSTEM]
1	Connections with data sources	SQLAllocHandle	None
2		SQLConnect(W)	
3		SQLDriverConnect(W)	
4		SQLBrowseConnect(W)	
5	Acquisition of driver and data source information	SQLDataSources(W)	None
6		SQLDrivers(W)	
7		SQLGetInfo(W)	
8		SQLGetFunctions	

No.	Type	ODBC function name	Information output in [SYSTEM]
9		SQLGetTypeInfo(W)	
10	Setting and acquiring driver options	SQLSetConnectAttr(W)	None
11		SQLGetConnectAttr(W)	
12		SQLSetEnvAttr	
13		SQLGetEnvAttr	
14		SQLSetStmtAttr(W)	
15		SQLGetStmtAttr(W)	
16	Setting descriptor values	SQLGetDescField(W)	None
17		SQLGetDescRec(W)	
18		SQLSetDescField(W)	
19		SQLSetDescRec	
20		SQLCopyDesc	
21	Creating SQL requests	SQLPrepare(W)	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
22		SQLBindParameter	None
23		SQLGetCursorName(W)	
24		SQLSetCursorName(W)	
25		SQLDescribeParam	
26		SQLNumParams	
27	SQL execution	SQLExecute	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
28		SQLExecDirect(W)	
29		SQLNativeSql(W)	None
30		SQLParamData	
31		SQLPutData	
32	Acquiring execution results and execution result information	SQLRowCount	None
33		SQLNumResultCols	
34		SQLDescribeCol(W)	
35		SQLColAttribute(W)	
36		SQLBindCol	
37		SQLFetch	
38		SQLFetchScroll	
39		SQLGetData	
40		SQLSetPos	
41		SQLBulkOperations	

No.	Type	ODBC function name	Information output in [SYSTEM]
42		SQLMoreResults	
43		SQLGetDiagField(W)	
44		SQLGetDiagRec(W)	
45	Acquiring system information for data sources	SQLColumnPrivileges(W)	None
46		SQLColumns(W)	
47		SQLForeignKeys(W)	
48		SQLPrimaryKeys(W)	
49		SQLProcedureColumns(W)	
50		SQLProcedures(W)	
51		SQLSpecialColumns(W)	
52		SQLStatistics(W)	
53		SQLTablePrivileges(W)	
54		SQLTables(W)	
55	Terminating SQL execution	SQLFreeStmt	When Option=SQL_CLOSE, the following information is output: • tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
56		SQLCloseCursor	• tran_id: Transaction ID • stmt_hdl: Statement handle ID • sql_serial_num: SQL statement sequence number
57		SQLCancel	None
58		SQLEndTran	tran_id: Transaction ID
59	Disconnecting from data sources	SQLDisconnect	None
60		SQLFreeHandle	

Note:

In some circumstances, such as when the transaction has been settled, HADB might be unable to acquire tran_id, stmt_htdl, and sql_serial_num. In this case, * is output for each value.

17.5 Notes about HADB ODBC driver trace information

- If an error or warning occurs that is caused the HADB ODBC driver trace itself, the output of HADB ODBC driver trace information might stop. This will not affect the execution of ODBC functions.
- In environments where HADB ODBC driver trace information is being output, its output affects all applications that use the ODBC interface. Keep this in mind when deciding whether to output HADB ODBC driver trace information.
- The size of the HADB ODBC driver trace file increases each time HADB ODBC driver trace information is output. For this reason, insufficient space at the output destination might prevent further HADB ODBC driver trace information from being output.

18 Creating Application Programs

This chapter describes the basic considerations involved in designing and creating application programs in C and C++.

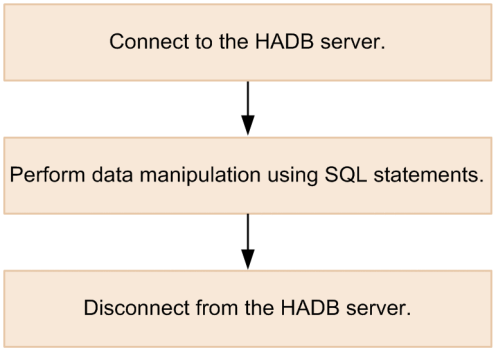
18.1 Designing application programs

This section describes the basic considerations involved in designing application programs.

18.1.1 Flow of application program processing

The following figure shows the flow of application program processing for manipulating a database.

Figure 18-1: Flow of application program processing for manipulating a database



Connecting to the HADB server

To manipulate a database from an application program, you must connect the application program to the HADB server. You do this by first allocating a connection handle for uniquely identifying the connection to the application program. Next, you use this connection handle to establish the connection. The application program is then connected to the HADB server.

You can also establish multiple connections from the same application program. The maximum number of connections that can be established concurrently is fixed.

Note that you must start an HADB server before you connect an application program to it.

Once an application program is connected to the HADB server, it can use SQL statements to manipulate the database.

Disconnecting from the HADB server

Before you terminate the application program, make sure that you disconnect it from the HADB server. First, close the connection and then release the connection handle. The application program is then disconnected from the HADB server, so that you can terminate the application program.

18.1.2 Transaction control

This subsection explains transaction start and termination and transaction control (commit and rollback processing).

(1) Relationship between a connection and a transaction

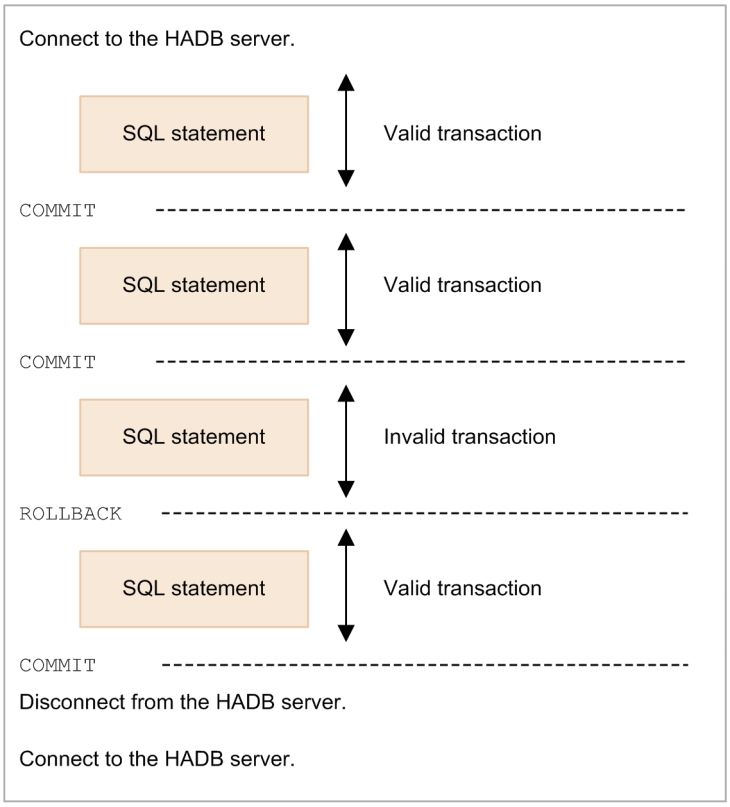
Before starting a transaction, make sure that you establish a connection with the HADB server. No transaction can be started if a connection has not been established. You use the `a_rdb_SQLConnect()` CLI function to establish a connection.

When a connection that has been established is closed, the transaction is committed (the transaction terminates normally when the connection is closed without having to explicitly issue `COMMIT` from within the application program). You use the `a_rdb_SQLDisconnect()` CLI function to close the connection.

(2) Starting and terminating a transaction

A transaction starts when the application program acquires a statement handle. The transaction is terminated when COMMIT or ROLLBACK is executed. The following figure shows an example of transaction start and termination.

Figure 18-2: Example of transaction start and termination



COMMIT and ROLLBACK for transactions are executed by using the a_rdb_SQLEndTran() CLI function.

When a definition SQL statement is executed, COMMIT processing is performed automatically.

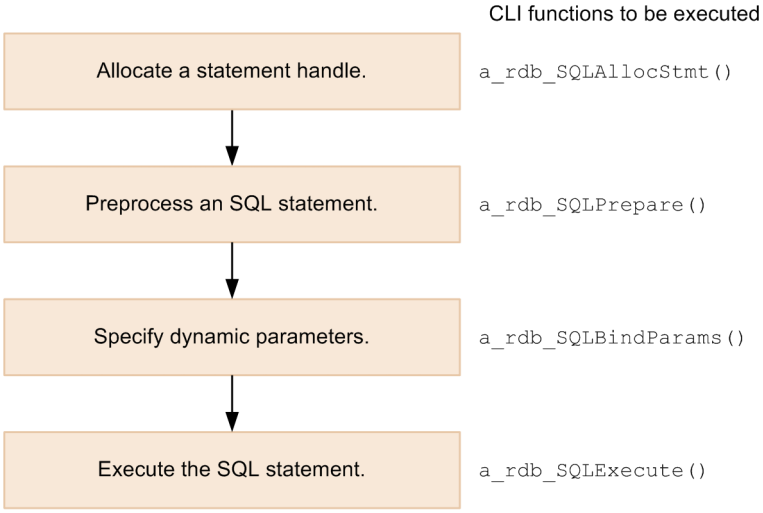
18.1.3 Flow of processing using dynamic parameters

When you execute a data manipulation SQL statement multiple times using different input values, the use of dynamic parameters can improve processing efficiency as compared to when you use the application program to create separate SQL statements, because repeated preprocessing of the SQL statements is eliminated.

You designate the use of a dynamic parameter by specifying ? at the location in the SQL statement where a value is to be passed from the application program. You use the a_rdb_SQLPrepare() CLI function to perform preprocessing. After that, you use the a_rdb_SQLBindParams() CLI function to set the dynamic parameter so that you can execute the SQL statement with the appropriate dynamic parameter value specified.

The following figure shows the flow of processing using dynamic parameters.

Figure 18-3: Flow of processing using dynamic parameters



For details about dynamic parameters, see *Variables (dynamic parameters)* in the manual *HADB SQL Reference*.

18.1.4 Effects of update operations on a retrieval using a cursor

When an update operation is performed during a retrieval using a cursor, the results of the update operation might be applied to the retrieval results, depending on the timing. To prevent the results of update operations from being applied to retrieval results, do the following:

- Close the cursor before adding or updating rows.
- Specify data and search conditions in such a manner that rows to be added or updated are not included in the retrieval results.

The following example performs updating while a retrieval processing using a cursor is underway:

```
char *selSql = "SELECT * FROM T1 WHERE C1 BETWEEN 10 AND 20";
char *updSql = "UPDATE T1 SET C1=30 WHERE C1=20";

/* preprocessing for the SELECT statement */
rtnc = a_rdb_SQLPrepare(cnctContext, hStmt1,selSql);
    :

/* retrieve rows */
rtnc = a_rdb_SQLFetch(cnctContext, hStmt1);                ...1

/* preprocessing for UPDATE */
rtnc = a_rdb_SQLPrepare(cnctContext, hStmt2, updSql);

/* update rows */
rtnc = a_rdb_SQLExecute(cnctContext, hStmt2);              ...2

/* retrieve rows */
rtnc = a_rdb_SQLFetch(cnctContext, hStmt1);                ...3
    :
```

Explanation:

If rows whose `C1` column is `20` are updated in 2, the HADB server is already performing retrieval processing asynchronously with the application program due to execution of the first `FETCH` in 1. Depending on the timing,

the application program might not be able at 3 to retrieve rows whose C1 column is 20. If retrieval of rows whose C1 column is 20 has already been completed, the application program can retrieve rows.

18.1.5 Evaluation and handling of SQL statement errors

You must determine whether the SQL statements executed by the application program have executed successfully. This subsection explains how to determine whether the SQL statements have been executed successfully and the error handling procedure.

(1) How to evaluate SQL statement errors

When an SQL statement is executed, SQLCODE is returned from the CLI function. You use the SQLCODE value to determine whether the SQL statement executed successfully. The following table lists and describes the SQLCODE values.

Table 18-1: SQLCODE values

No.	SQLCODE value	Meaning
1	100	There are no more rows to be retrieved. This is useful especially when the following is to be determined: • Whether all rows have been fetched by a FETCH statement • Whether there is any row that is to be updated by an INSERT, DELETE, or UPDATE statement
2	1	The SQL statement's processing terminated, but a warning occurred in an extension of the processing. Possible causes of warnings are as follows: • The disk at the HADB server that stores the server message log files has become full. • The disk that stores client message log files has become full. • A memory shortage was detected when outputting SQL trace information on the HADB server. • ROLLBACK processing was performed and then the connection was closed because COMMIT processing failed in an extension of the connection close processing.
3	Negative value	An SQL error occurred.
4	Other	The SQL statement executed successfully.

For details about the return values of the CLI function, see 19.8 Return values of the CLI functions.

(2) Error handling procedure

You use the procedure described below to handle errors that are detected. This subsection presents an example using a CLI function.

(a) Output of return value

The CLI function's return value is output or displayed.

(b) Error handling

If data manipulation by an SQL statement results in an error, take the action described below.

To handle the error:

1. Check the value of the isInConnect member in the SQL results information. The action to be taken depends on the value of isInConnect:

- **If the value of `isInConnect` is `a_rdb_SQL_IS_IN_CONNECT`**

  Go to step 2.

- **If the value of `isInConnect` is `a_rdb_SQL_IS_NOT_IN_CONNECT`**

  The application program has disconnected from the HADB server due to a fatal error. Check the message output to the client message log file. If the HADB server terminated abnormally, contact the HADB administrator.

2. Check the value of the `EndTran` member in the SQL results information. The action to be taken depends on the `EndTran` value, as shown in the following table:

No.	Value of EndTran	Action
1	`a_rdb_SQL_ROLLBACKED`	Because internal rollback was executed, disconnect the application program from the HADB server and terminate the application program. Then take appropriate action based on the return value.
2	`a_rdb_SQL_TRAN_NOT_ENDED`	Execute `ROLLBACK` to cancel the transaction, disconnect the application program from the HADB server, and terminate the application program. Then take appropriate action based on the return value.
3	Other	Disconnect the application program from the HADB server and terminate the application program. Then take appropriate action based on the return value.

18.2 How to use the CLI functions

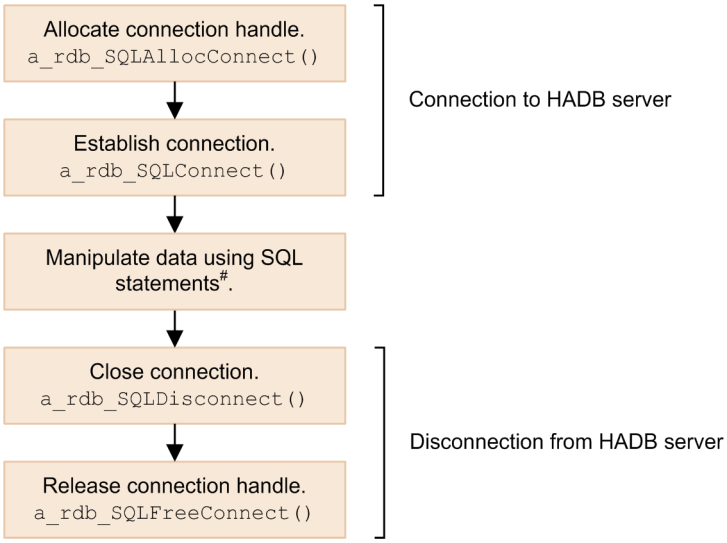
You use the CLI functions to create application programs in C or C++. This section explains the basic usage of CLI functions.

18.2.1 Connecting to and disconnecting from the HADB server

This subsection explains the procedure for using CLI functions to connect to and disconnect from the HADB server.

The following figure shows the procedure from connection through disconnection from the HADB server.

Figure 18-4: Procedure from connection through disconnection from the HADB server



#

For details about using SQL statements to manipulate data, see 18.2.2 Referencing data and its subsections.

(1) Connecting to the HADB server

You use the following CLI functions to connect to the HADB server:

- `a_rdb_SQLAllocConnect()` (allocate a connection handle)
- `a_rdb_SQLConnect()` (establish a connection)

(a) Allocating a connection handle

Before you establish a connection, you must allocate a connection handle. You use `a_rdb_SQLAllocConnect()` to allocate a connection handle. When a connection handle has been allocated successfully, the value `a_rdb_RC_SQL_SUCCESS` is returned. The following shows an example of calling `a_rdb_SQLAllocConnect()`.

Example of calling a_rdb_SQLAllocConnect()

```
signed short rtnc ;                 /* Return value */
void  *hCnct ;                      /* Connection handle address */

/* Allocate a connection handle */
```

```
rtnc = a_rdb_SQLAllocConnect(&hCnct,
                         "/hadb/client.def",    /* Client definition file path */
                         NULL) ;
```

You can use a different client definition for each connection by specifying the absolute path of the appropriate client definition file in the second argument of `a_rdb_SQLAllocConnect()`. For details about a client definition, see 4.4 Creating a client definition.

For details about `a_rdb_SQLAllocConnect()`, see 19.2.1 a_rdb_SQLAllocConnect() (allocate a connection handle).

(b) Establishing a connection

You use `a_rdb_SQLConnect()` to establish a connection. When a connection has been established, the value `a_rdb_RC_SQL_SUCCESS` is returned.

The following shows an example of calling `a_rdb_SQLConnect()`.

Example of calling a_rdb_SQLConnect()

```
a_rdb_SQLResultInfo_t  rsltInfo ;          /* SQL results information */

/* Establish a connection */
rtnc = a_rdb_SQLConnect(hCnct,             /* Connection handle */
                        "ADBUSER01",       /* Authorization identifier */
                        "password01",      /* Password */
                        &rsltInfo,
                        NULL) ;
```

You can acquire SQL results information for each CLI function call by specifying the corresponding address of the SQL results information in the fourth argument of `a_rdb_SQLConnect()`. For details about SQL results information, see 19.7.6 a_rdb_SQLResultInfo_t structure (SQL results information).

For details about `a_rdb_SQLConnect()`, see 19.2.2 a_rdb_SQLConnect() (establish a connection).

(2) Disconnecting from the HADB server

You use the following CLI functions to disconnect from the HADB server:

- `a_rdb_SQLDisconnect()` (close a connection)
- `a_rdb_SQLFreeConnect()` (release the connection handle)

(a) Closing the connection

You use `a_rdb_SQLDisconnect()` to close the connection. The following shows an example of calling `a_rdb_SQLDisconnect()`.

Example of calling a_rdb_SQLDisconnect()

```
/* Close the connection */
rtnc = a_rdb_SQLDisconnect(hCnct, NULL) ;
```

For details about `a_rdb_SQLDisconnect()`, see 19.2.4 a_rdb_SQLDisconnect() (close a connection).

(b) Releasing the connection handle

After you have closed the connection, you must release the connection handle. You use `a_rdb_SQLFreeConnect()` to release a connection handle. The following shows an example of calling `a_rdb_SQLFreeConnect()`.

Example of calling a_rdb_SQLFreeConnect()

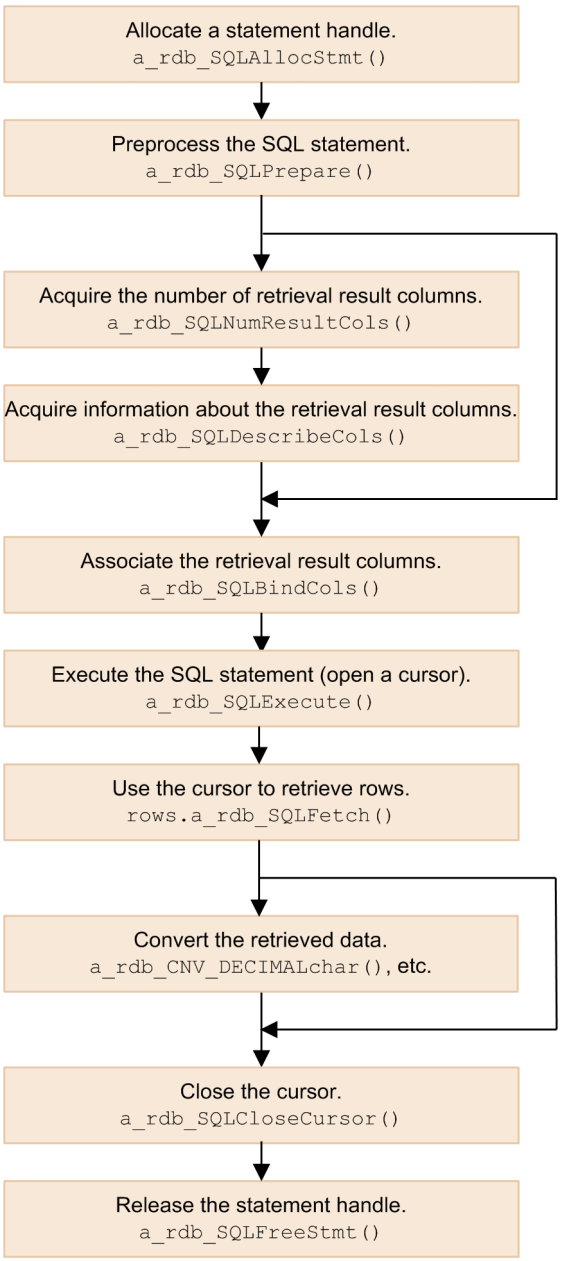
```
/* Release the connection handle  */
rtnc = a_rdb_SQLFreeConnect(hCnct, NULL) ;
```

For details about `a_rdb_SQLFreeConnect()`, see 19.2.5 a_rdb_SQLFreeConnect() (release a connection handle).

18.2.2 Referencing data

This subsection explains how to reference data by way of an example. The example provided here uses a cursor to retrieve rows. The following figure shows how to retrieve rows using a cursor.

Figure 18-5: How to retrieve rows using a cursor



The processing steps are explained below.

(1) Allocating a statement handle

Before you execute an SQL statement, you must use `a_rdb_SQLAllocStmt()` to allocate a statement handle. When a statement handle is allocated successfully, `a_rdb_RC_SQL_SUCCESS` is returned. The following shows an example of statement handle allocation.

Example of statement handle allocation

```
void  *hStmt ;                              /* Statement handle address */

/* Allocate a statement handle */
rtnc = a_rdb_SQLAllocStmt(hCnct,
                          &hStmt,
                          NULL) ;
```

For details about a_rdb_SQLAllocStmt(), see 19.4.1 a_rdb_SQLAllocStmt() (allocate a statement handle).

(2) Preprocessing an SQL statement

Next, you allocate the statement handle acquired in (1) Allocating a statement handle to an SQL statement. To do this, you use a_rdb_SQLPrepare() to preprocess the SQL statement. The following shows an example of preprocessing a SELECT statement.

Example of preprocessing a SELECT statement

```
/* Preprocess a SELECT statement */
rtnc = a_rdb_SQLPrepare(hCnct,
                        hStmt,
                        "SELECT C1,C2,C3 FROM T1",
                        NULL) ;
```

For details about a_rdb_SQLPrepare(), see 19.4.14 a_rdb_SQLPrepare() (preprocess an SQL statement).

(3) Acquiring the number of retrieval result columns

If the number of retrieval result columns (the number of columns that are output as the retrieval results) is not known at the time of application program creation, such as when SQL statements will be executed dynamically, you use a_rdb_SQLNumResultCols() to acquire the number of retrieval result columns. The following shows an example of acquiring the number of retrieval result columns.

Example of acquiring the number of retrieval result columns

```
/* Acquire the number of retrieval result columns */
rtnc = a_rdb_SQLNumResultCols(hCnct,
                              hStmt,
                              &colCount,    /* Number of columns */
                              NULL) ;
```

For details about a_rdb_SQLNumResultCols(), see 19.4.13 a_rdb_SQLNumResultCols() (acquire the number of retrieval result columns).

(4) Acquiring information about the retrieval result columns

If column information, such as the column names, data types, and data lengths, is not known at the time of application program creation, such as when SQL statements will be executed dynamically, you use a_rdb_SQLDescribeCols() to acquire information about the retrieval result columns. You can acquire the following information by using a_rdb_SQLDescribeCols():

- Column names of the retrieval result columns
- Data types of the retrieval result columns
- Maximum numbers of elements in the retrieval result columns
- Data lengths of the retrieval result columns

The following shows an example of acquiring information about the retrieval result columns.

Example of acquiring information about the retrieval result columns

```
/* Acquire information about the retrieval result columns */
rtnc = a_rdb_SQLDescribeCols(hCnct,
                             hStmt,
```

```
                              colCount,       /* Number of retrieval result columns
 */
                              &(colInf[0]),   /* Information return area for all re
trieval result columns */
                              NULL) ;
```

For details about a_rdb_SQLDescribeCols(), see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns).

(5) Associating the retrieval result columns

You must associate the retrieval result columns with the area for storing values retrieved from the retrieval result columns. When you use a_rdb_SQLFetch() to manipulate the cursor, the values in the retrieval result columns are stored automatically in the associated area.

You use a_rdb_SQLBindCols() to associate retrieval result columns. The following shows an example of associating retrieval result columns.

Example of associating retrieval result columns

```
/* Associate retrieval result columns */
rtnc = a_rdb_SQLBindCols(hCnct,
                         hStmt,
                         colCount,           /* Number of retrieval result columns
 */
                         &(colInf[0]),       /* Assigned storage area for all colu
mns */
                         NULL) ;
```

For details about a_rdb_SQLBindCols(), see 19.4.3 a_rdb_SQLBindCols() (associate retrieval result columns).

(6) Executing the SQL statement (opens a cursor)

You use a_rdb_SQLExecute() to execute the preprocessed SQL statements. Specify in the argument of a_rdb_SQLExecute() the statement handle for the SQL statement that is to be executed. If the SQL statement executes successfully, the value a_rdb_RC_SQL_SUCCESS is returned and a cursor opens. The following shows an example of executing an SQL statement.

Example of executing an SQL statement

```
/* Execute an SQL statement */
rtnc = a_rdb_SQLExecute(hCnct,
                        hStmt,
                        NULL) ;
```

For details about a_rdb_SQLExecute(), see 19.4.9 a_rdb_SQLExecute() (execute a preprocessed SQL statement).

(7) Using the cursor to retrieve rows

You use a_rdb_SQLFetch() to retrieve rows by using the cursor that has been opened by executing the SQL statement. If row retrieval is successful, the value a_rdb_RC_SQL_SUCCESS is returned. The following shows an example of using a cursor to retrieve rows.

Example of using a cursor to retrieve rows

```
/* Using cursor to retrieve rows */
rtnc = a_rdb_SQLFetch(hCnct,
```

```
                          hStmt,
                          NULL) ;
```

For details about a_rdb_SQLFetch(), see 19.4.10 a_rdb_SQLFetch() (fetch a row).

(8) Converting the retrieved data

If the fetched data's SQL data type is DECIMAL, BINARY, VARBINARY, DATE, TIME, or TIMESTAMP, you can use a CLI function to convert it to character string data supported by C or C++. The following shows an example of converting DECIMAL-type data.

Example of data conversion

```
#define PRECISION 6
#define SCALE 3

unsigned char data_decimal[4];
char data_char[PRECISION+4];

/* Convert DECIMAL-type data */
rtnc = a_rdb_CNV_DECIMALchar(data_decimal,   /* Start address of output data */
                             PRECISION,      /* Precision of output data */
                             SCALE,          /* Scaling of output data */
                             data_char,      /* Address of area for storing convert
ed data */
                             (PRECISION+4),  /* Length of area for storing converte
d data */
                             NULL);
```

- To convert DECIMAL-type data, you use a_rdb_CNV_DECIMALchar(). For details about a_rdb_CNV_DECIMALchar(), see 19.5.9 a_rdb_CNV_DECIMALchar() (convert DECIMAL-type data).
- To convert BINARY-type data, you use a_rdb_CNV_BINARYchar(). For details about a_rdb_CNV_BINARYchar(), see 19.5.7 a_rdb_CNV_BINARYchar() (convert BINARY-type data).
- To convert VARBINARY-type data, you use a_rdb_CNV_VARBINARYchar(). For details about a_rdb_CNV_VARBINARYchar(), see 19.5.12 a_rdb_CNV_VARBINARYchar() (convert VARBINARY-type data).
- To convert DATE-type data, you use a_rdb_CNV_DATEchar(). For details about a_rdb_CNV_DATEchar(), see 19.5.8 a_rdb_CNV_DATEchar() (convert DATE-type data).
- To convert TIME-type data, you use a_rdb_CNV_TIMEchar(). For details about a_rdb_CNV_TIMEchar(), see 19.5.10 a_rdb_CNV_TIMEchar() (convert TIME-type data).
- To convert TIMESTAMP-type data, you use a_rdb_CNV_TIMESTAMPchar(). For details about a_rdb_CNV_TIMESTAMPchar(), see 19.5.11 a_rdb_CNV_TIMESTAMPchar() (convert TIMESTAMP-type data).

(9) Closing the cursor

You use a_rdb_SQLCloseCursor() to close the cursor. The following shows an example of closing the cursor.

Example of closing the cursor

```
/* Close the cursor */
rtnc = a_rdb_SQLCloseCursor(hCnct,
                            hStmt,
                            NULL) ;
```

For details about a_rdb_SQLCloseCursor(), see 19.4.5 a_rdb_SQLCloseCursor() (close the cursor).

(10) Releasing the statement handle

You use a_rdb_SQLFreeStmt() to release the allocated statement handle. The following shows an example of releasing the statement handle.

Example of releasing the statement handle

```
/* Release the statement handle */
rtnc = a_rdb_SQLFreeStmt(hCnct,
                         hStmt,
                         NULL) ;
```

Important

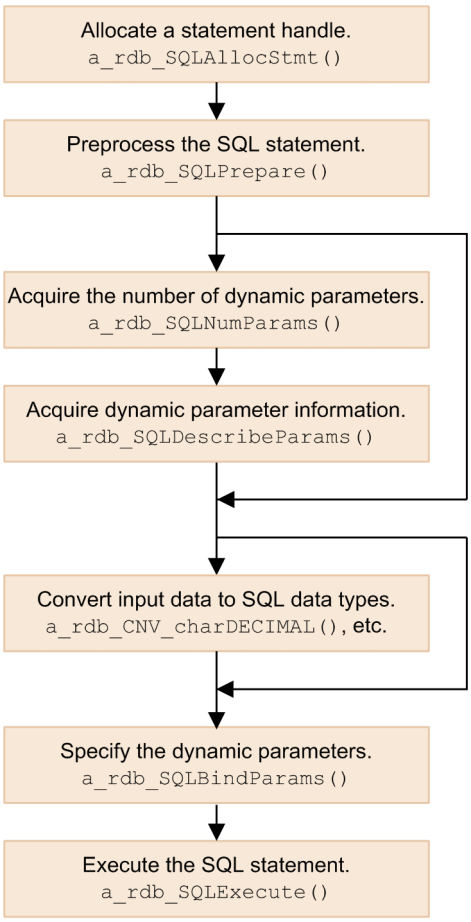
The statement handle is released any time COMMIT or ROLLBACK is executed. In such a case, do not execute a_rdb_SQLFreeStmt().

For details about a_rdb_SQLFreeStmt(), see 19.4.11 a_rdb_SQLFreeStmt() (release a statement handle).

18.2.3 Using dynamic parameters

Executing an SQL statement with dynamic parameters involves allocation of a statement handle, preprocessing of the SQL statement, specifying the dynamic parameters, and execution of the SQL statement. The following figure shows the procedure for executing an SQL statement with dynamic parameters specified.

Figure 18-6: Procedure for executing an SQL statement with dynamic parameters specified



The methods for allocating the statement handle and preprocessing and executing the SQL statement are the same as explained in 18.2.2 Referencing data. This subsection explains how to acquire the number of dynamic parameters and dynamic parameter information, convert input data to SQL data types, and specify dynamic parameters.

(1) Acquiring the number of dynamic parameters

If the number of dynamic parameters is not known at the time of application program creation, such as when SQL statements will be executed dynamically, you use `a_rdb_SQLNumParams()` to acquire the number of dynamic parameters. The following shows an example of acquiring the number of dynamic parameters.

Example of acquiring the number of dynamic parameters

```
/* Acquire the number of dynamic parameters */
rtnc = a_rdb_SQLNumParams(hCnct,
                          hStmt,
                          &paramCount,    /* Number of dynamic parameters */
                          NULL) ;
```

For details about `a_rdb_SQLNumParams()`, see 19.4.12 a_rdb_SQLNumParams() (acquire the number of dynamic parameters).

(2) Acquiring dynamic parameter information

If the dynamic parameter information, such as the data types and data lengths, is not known at the time of application program creation, such as when SQL statements will be executed dynamically, you use

a_rdb_SQLDescribeParams() to acquire the dynamic parameter information. You can acquire the following information by using a_rdb_SQLDescribeParams():

- Data types of the dynamic parameters
- Maximum numbers of elements of the dynamic parameters
- Data lengths of the dynamic parameters

The following shows an example of acquiring dynamic parameter information.

Example of acquiring dynamic parameter information

```
/* Acquire dynamic parameter information */
rtnc = a_rdb_SQLDescribeParams(hCnct,
                               hStmt,
                               paramCount,       /* Number of dynamic parameters *
/
                               &(paramInfo[0]),  /* Area for returning all dynamic
 parameter information */
                               NULL) ;
```

For details about a_rdb_SQLDescribeParams(), see 19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information).

(3) Converting input data to SQL data types

If the SQL data type of the input data for dynamic parameters is DECIMAL, BINARY, VARBINARY, DATE, TIME, or TIMESTAMP, you can use a CLI function to convert the character string data supported by C or C++ to the corresponding data type. The following shows an example of converting character string data in C or C++ to DECIMAL-type data.

Example of data conversion

```
#define PRECISION 6
#define SCALE 3

char data_char[]="-123.567";
unsigned char data_decimal[4];

/* Convert to DECIMAL-type data */
rtnc = a_rdb_CNV_charDECIMAL(data_char,          /* Start address of data to be co
nverted */
                             (unsigned short)strlen(data_char), /* Length of data t
o be converted */
                             PRECISION,          /* Precision of input data */
                             SCALE,              /* Scaling of input data */
                             data_decimal,       /* Address of input data storage
area */
                             4,                  /* Length of input data storage a
rea */
                             NULL);
```

- To convert input data to DECIMAL-type data, you use a_rdb_CNV_charDECIMAL(). For details about a_rdb_CNV_charDECIMAL(), see 19.5.3 a_rdb_CNV_charDECIMAL() (convert to DECIMAL-type data).
- To convert input data to BINARY-type data, you use a_rdb_CNV_charBINARY(). For details about a_rdb_CNV_charBINARY(), see 19.5.1 a_rdb_CNV_charBINARY() (convert to BINARY-type data).

- To convert input data to VARBINARY-type data, you use a_rdb_CNV_charVARBINARY(). For details about a_rdb_CNV_charVARBINARY(), see 19.5.6 a_rdb_CNV_charVARBINARY() (convert to VARBINARY-type data).
- To convert input data to DATE-type data, you use a_rdb_CNV_charDATE(). For details about a_rdb_CNV_charDATE(), see 19.5.2 a_rdb_CNV_charDATE() (convert to DATE-type data).
- To convert input data to TIME-type data, you use a_rdb_CNV_charTIME(). For details about a_rdb_CNV_charTIME(), see 19.5.4 a_rdb_CNV_charTIME() (convert to TIME-type data).
- To convert input data to TIMESTAMP-type data, you use a_rdb_CNV_charTIMESTAMP(). For details about a_rdb_CNV_charTIMESTAMP(), see 19.5.5 a_rdb_CNV_charTIMESTAMP() (convert to TIMESTAMP-type data).

(4) Specifying the dynamic parameters

You use a_rdb_SQLBindParams() to specify the dynamic parameters. When the dynamic parameters are specified successfully, the return value a_rdb_RC_SQL_SUCCESS is returned. The following shows an example of specifying dynamic parameters.

Example of dynamic parameter specification

```
/* Specify dynamic parameters */
rtnc = a_rdb_SQLBindParams(hCnct,
                           hStmt,
                           paramCount,       /* Number of dynamic parameters */
                           &(paramInfo[0]),  /* Area for specifying all dynamic pa
rameters */
                           NULL) ;
```

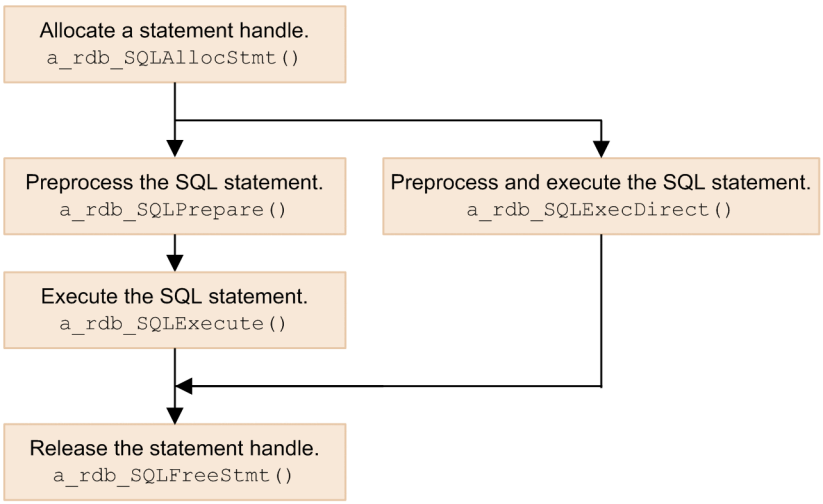
For details about a_rdb_SQLBindParams(), see 19.4.4 a_rdb_SQLBindParams() (associate dynamic parameters).

18.2.4 Adding, updating, or deleting data

To use an update SQL statement to add, update, or delete data, you allocate a statement handle, preprocess the SQL statement, and then use a_rdb_SQLExecute() to execute the SQL statement, in the same manner as for a SELECT statement.

The following figure shows the procedure for data update and deletion processing.

Figure 18-7: Procedure for data update and deletion processing



Note

You can also use `a_rdb_SQLExecDirect()` to both preprocess and execute an SQL statement without preprocessing the SQL statement in advance with `a_rdb_SQLPrepare()`.

Note that using dynamic parameters requires a process of binding (association), which can be performed by using `a_rdb_SQLBindParams()` or `a_rdb_SQLBindArrayParams()`. For details about `a_rdb_SQLBindParams()`, see 19.4.4 a_rdb_SQLBindParams() (associate dynamic parameters). For details about `a_rdb_SQLBindArrayParams()`, see 19.4.2 a_rdb_SQLBindArrayParams() (bind dynamic parameters in batch mode).

The following shows an example of updating and deleting data.

Example of updating and deleting data

```
/* Allocate a statement handle */
rtnc = a_rdb_SQLAllocStmt(hCnct, &hStmt, NULL) ;
/* Prepare the INSERT statement */
rtnc = a_rdb_SQLPrepare(hCnct,
                        hStmt,
                        "INSERT INTO TABLE1 VALUES ('A','B','C')",
                        NULL) ;
/* Execute the SQL statement */
rtnc = a_rdb_SQLExecute(hCnct, hStmt, NULL) ;
/* Execute DELETE */
rtnc = a_rdb_SQLExecDirect(hCnct,
                           hStmt,
                           "DELETE FROM TABLE1",
                           NULL) ;
```

For details about `a_rdb_SQLExecDirect()`, see 19.4.8 a_rdb_SQLExecDirect() (preprocess and execute an SQL statement).

18.2.5 Canceling SQL processing that is executing

You can use `a_rdb_SQLCancel()` to cancel SQL processing that is executing. The following SQL statements (CLI functions) can be canceled:

- a_rdb_SQLCloseCursor()
- a_rdb_SQLExecDirect()
- a_rdb_SQLExecute()
- a_rdb_SQLFetch()
- a_rdb_SQLPrepare()

You must execute a_rdb_SQLCancel() in a separate thread from the thread being used for the SQL processing. The following shows an example of canceling SQL processing.

SQL execution thread

```
rtnc = a_rdb_SQLAllocStmt(hCnct, &hStmt, NULL) ;
/* Execute the DELETE statement */
rtnc = a_rdb_SQLExecDirect(hCnct,
                           hStmt,
                           "DELETE FROM TABLE1",
                           NULL) ;
/* Check whether SQL processing was canceled */
if ( rtnc == -955 )
{
  /* Processing after cancellation */
}
else ;
```

Example of canceling an SQL statement (executing in a separate thread)

```
/* Execute the SQL cancel function */
rtnc = a_rdb_SQLCancel(hCnct, NULL) ;
                  /* Specify the connection handle being used for the   */
                  /* SQL processing that is to be canceled */
```

When cancellation of SQL processing is successful, the SQL processing is canceled, the transaction is rolled back, and SQLCODE is returned.

For details about a_rdb_SQLCancel(), see 19.3.1 a_rdb_SQLCancel() (cancel SQL processing).

Note that normal termination of a_rdb_SQLCancel() does not mean that the cancellation processing was successful because the cancellation processing is performed asynchronously with a_rdb_SQLCancel().

18.2.6 Notes about using the CLI functions

This section describes the valid address period and boundary alignment that are specified in the arguments of CLI functions.

(1) Valid period of area

Operations cannot be guaranteed if an address specified in a CLI function argument is no longer valid because the area indicated by the address has been released.

You must ensure that the arguments listed below that specify an address to a CLI function are valid throughout the processing (from the beginning to the end of the processing):

- Arguments that must be valid from the time the application program is connected to the HADB server until it is disconnected from the HADB server:

- `ConnectionHandle` of each CLI function
- `ResultInfo` of `a_rdb_SQLConnect()` (if SQL results information is acquired)

- Arguments that must be valid from the time a cursor is opened until it is closed:
  - `TargetValue` and `StrLen_or_Ind` members of the `ColumnInfo` structure of `a_rdb_SQLBindCols()`
  - `ParameterValue` and `Ind` members of the `ParameterInfo` structure of `a_rdb_SQLBindParams()` and `a_rdb_SQLBindArrayParams()`

    For the arguments of `a_rdb_SQLBindParams()` and `a_rdb_SQLBindArrayParams()`, you must ensure that the area is valid until `a_rdb_SQLExecute()` is executed, even when an SQL statement that does not use a cursor is preprocessed.

There are no arguments that are required to be valid from transaction startup to transaction termination.

(2) Boundary alignment

For each SQL data type, you must apply boundary alignment to the start address of the area for storing information, such as data for retrieval results and input values of dynamic parameters, according to the following table.

Table 18-2: Boundary alignment

No.	SQL data type	Boundary alignment
1	`VARCHAR` type `VARBINARY` type	2-byte boundary
2	`SMALLINT` type	4-byte boundary
3	`INTEGER` type	8-byte boundary
4	`DOUBLE PRECISION` type	

18.3 Compiling and linking application programs

You must use a compiler designed for the programming language in which the application was coded to compile and link the source program. For details about how to compile and link, see the compiler documentation for the applicable language. Also see the *Readme* file for the applicable OS.

When you compile and link an application program, specify the library provided by the HADB client that is appropriate to your development environment, as shown in the following table.

Table 18-3: Library to be specified when application programs are compiled and linked

No.	Development environment (OS)	Library to be specified
1	Windows	• `adbclt.lib` (for 64-bit edition of Windows) • `adbclt32.lib` (for 32-bit edition of Windows)
2	Linux	`libadbclt.so`

19 CLI Functions

This chapter explains the capabilities and syntax of the CLI functions provided by HADB.

19.1 List of CLI functions and common rules

This section presents a list of the CLI functions and explains their common rules.

19.1.1 List of CLI functions

HADB provides the CLI functions listed in the table below.

Table 19-1: List of CLI functions

No.	Classification		CLI function	Function
1	CLI functions for connecting to and disconnecting from the HADB server		a_rdb_SQLAllocConnect()	Allocates a connection handle.
2			a_rdb_SQLConnect()	Establishes a connection with the HADB server.
3			a_rdb_SQLSetConnectAttr()	Sets connection attributes.
4			a_rdb_SQLDisconnect()	Closes a connection.
5			a_rdb_SQLFreeConnect()	Releases a connection handle.
6	CLI functions for controlling transactions		a_rdb_SQLCancel()	Cancels the current SQL processing.
7			a_rdb_SQLEndTran()	Terminates the transaction.
8	CLI functions for execution of SQL statements	Allocating and releasing a statement handle	a_rdb_SQLAllocStmt()	Allocates a statement handle.
9			a_rdb_SQLFreeStmt()	Releases a statement handle.
10		Preprocessing and executing an SQL statement, manipulating a cursor, and fetching rows	a_rdb_SQLPrepare()	Preprocesses an SQL statement.
11			a_rdb_SQLExecute()	Executes a preprocessed SQL statement.
12			a_rdb_SQLExecDirect()	Preprocesses and executes an SQL statement.
13			a_rdb_SQLFetch()	Advances to the next row the cursor that points to the next row to be fetched, and then reads the column values in that row into the fetch target specified in the fetch target list.
14			a_rdb_SQLCloseCursor()	Closes the cursor.
15		Retrieval result columns	a_rdb_SQLNumResultCols()	Acquires the number of retrieval result columns.
16			a_rdb_SQLDescribeCols()	Acquires information about the retrieval result columns.
17			a_rdb_SQLBindCols()	Binds (associates) the retrieval result columns with an area for storing the values fetched from those columns.
18		Dynamic parameters	a_rdb_SQLNumParams()	Acquires the number of dynamic parameters in an SQL statement.

No.	Classification		CLI function	Function
19			a_rdb_SQLDescribeParams()	Acquires an SQL statement's dynamic parameter information.
20			a_rdb_SQLBindParams()	Binds (associates) the dynamic parameters in an SQL statement with an area for specifying their values.
21			a_rdb_SQLBindArrayParams()	Binds (associates) the dynamic parameters in an SQL statement with an area for specifying their values. This CLI function binds the values of multiple sets of dynamic parameters in an SQL statement in batch mode.
22	CLI functions for data type conversion	Converting character string data in C or C++ to SQL data types	a_rdb_CNV_charBINARY()	Converts character string data in C or C++ (binary or hexadecimal) to SQL BINARY type data.
23			a_rdb_CNV_charDATE()	Converts character string data in C or C++ to SQL DATE type data.
24			a_rdb_CNV_charDECIMAL()	Converts character string data in C or C++ to SQL DECIMAL type data.
25			a_rdb_CNV_charTIME()	Converts character string data in C or C++ to SQL TIME type data.
26			a_rdb_CNV_charTIMESTAMP()	Converts character string data in C or C++ to SQL TIMESTAMP type data.
27			a_rdb_CNV_charVARBINARY()	Converts character string data in C or C++ (binary or hexadecimal) to SQL VARBINARY type data.
28		Converting SQL data types to character string data in C or C++	a_rdb_CNV_BINARYchar()	Converts SQL BINARY-type data to character string data in C or C++.
29			a_rdb_CNV_DATEchar()	Converts SQL DATE-type data to character string data in C or C++.
30			a_rdb_CNV_DECIMALchar()	Converts SQL DECIMAL-type data to character string data in C or C++.
31			a_rdb_CNV_TIMEchar()	Converts SQL TIME-type data to character string data in C or C++.
32			a_rdb_CNV_TIMESTAMPchar()	Converts SQL TIMESTAMP-type data to character string data in C or C++.
33			a_rdb_CNV_VARBINARYchar()	Converts SQL VARBINARY-type data to character string data in C or C++.

To use these CLI functions, you must load the header files and the client library provided by the HADB client into a source program coded in C or C++. The following table lists the header files and the client library provided by the HADB client.

Table 19-2: Header files and client library provided by the HADB client

No.	CLI function to be used	Header file and client library to be used	
		Header file	Client library

1	CLI functions for connecting to and disconnecting from the HADB server	• `$ADBDIR/include/adbcli.h`
		• `$ADBDIR/include/adbtypes.h`
			• 64-bit edition of Windows
			`%ADBCLTDIR%\client\lib\adbclt.lib`
			• 32-bit edition of Windows
			`%ADBCLTDIR%\client\lib\adbclt32.lib`
			• Linux
			`$ADBDIR/client/lib/libadbclt.so`

2	CLI functions for controlling transactions		
3	CLI functions for execution of SQL statements		
4	CLI functions for data type conversion	`$ADBDIR/include/adbcnv.h`	

Note: The header file that is required depends on the CLI function being used.

19.1.2 Common rules

This subsection describes the rules common to all CLI functions.

(1) Connection handle

A connection handle is used to uniquely identify each connection to the HADB server. A connection handle must remain valid from the time the connection to the HADB server is established until the connection is closed. If the connection handle does not remain valid for this entire period, operation is not guaranteed. To keep a connection handle valid, you must specify the connection handle allocated by `a_rdb_SQLAllocConnect()` in the argument of every CLI function until the connection handle is released by `a_rdb_SQLFreeConnect()`.

(2) SQL results information

You can acquire various types of SQL results information for each CLI function call. If you wish to acquire SQL results information, you must not release the area that stores the SQL results information until the connection to the HADB server is closed.

(3) Statement handle

A statement handle is used to provide a unique identification for each allocated SQL statement. When `COMMIT` or `ROLLBACK` occurs, the statement handle is released.

If you attempt to use multiple statement handles allocated by the same connection handle to execute SQL statements concurrently, but the number of processing real threads available does not match the maximum number of SQL processing real threads, SQL statement execution requests will result in an error (the requests will not be placed in wait status).

The maximum number of SQL processing real threads is specified in the following operands:

- `adb_sql_exe_max_rthd_num` in the server definition
- `adb_sql_exe_max_rthd_num` in the client definition

(4) Implicit cursor

Typically, table retrieval results consist of multiple rows. In order for an application program to retrieve one row of retrieval results at a time, a pointer called a cursor is used to retain the most recent retrieval location. The cursor is used for data retrieval.

A cursor is allocated when a `SELECT` statement is preprocessed and is opened when the `SELECT` statement is executed. When `COMMIT` or `ROLLBACK` occurs, all open cursors are closed.

19.2 CLI functions for connecting to and disconnecting from the HADB server

This section explains the CLI functions for connecting to and disconnecting from the HADB server.

19.2.1 a_rdb_SQLAllocConnect() (allocate a connection handle)

(1) Function

This CLI function allocates a connection handle to uniquely identify a connection with the HADB server.

(2) Format

```
signed short a_rdb_SQLAllocConnect
(
  void                          **ConnectionHandle,    /* Out */
  char                           *ClientDefPath,       /* In  */
  void                           *Option               /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies the address at which the connection handle is to be set.

`ClientDefPath`

Specifies the path of the client definition file, as a character string in C or C++.

When `NULL` is specified, the function assumes that `%ADBCLTDIR%\conf\client.def` is specified.#

#: The function assumes `$ADBCLTDIR/conf/client.def` for HADB clients using Linux.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLAllocConnect()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If a connection handle is allocated successfully, but the disk containing the client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

`a_rdb_SQLAllocConnect()` cannot be executed in the following cases:

- The path specified for the client definition file is invalid.
- The specified client definition is invalid.
- Allocation of a connection handle is not possible.

19.2.2 a_rdb_SQLConnect() (establish a connection)

(1) Function

This CLI function establishes a connection with the HADB server.

(2) Format

```
signed short a_rdb_SQLConnect
(
  void                          *ConnectionHandle,    /* In */
  char                          *UserID,              /* In */
  char                          *Password,            /* In */
  a_rdb_SQLResultInfo_t         *ResultInfo,          /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`UserID`

Specifies the authorization identifier to be used to connect to the HADB server, as a character string in C or C++.

`Password`

Specifies the password for the authorization identifier that was specified for `UserID`, in the representation of a character string in C or C++ language.

For details about the rules for passwords, see *Specification format and rules for the CREATE USER statement* in the manual *HADB SQL Reference*.

`ResultInfo`

Specifies for each CLI function the address where the SQL results information is to be stored. SQL results information is returned to the specified area until `a_rdb_SQLDisconnect()` is issued.

If the specified address is `NULL`, SQL results information is not returned.

For details about the `a_rdb_SQLResultInfo_t` structure, see 19.7.6 a_rdb_SQLResultInfo_t structure (SQL results information).

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLConnect()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the connection is established successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

`a_rdb_SQLConnect()` cannot be executed in the following cases:

- An HADB user who does not have the CONNECT privilege executed this CLI function.
- Connection has already been established.
- An invalid authorization identifier is specified.
- A character encoding that differs from that for the HADB server is specified in ADBCLTLANG.
- The number of connections to the HADB server has reached the maximum number of concurrent connections.

19.2.3 a_rdb_SQLSetConnectAttr() (set connection attributes)

(1) Function

This CLI function sets the connection attributes of a specified connection handle.

(2) Format

```
signed short a_rdb_SQLSetConnectAttr
(
  void                          *ConnectionHandle,    /* In */
  signed short                   Attribute,           /* In */
  void                          *Value,               /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

Attribute

Specifies one of the following connection attributes:

- When the transaction isolation level is set:
  a_rdb_SQL_ATTR_TXN_ISOLATION
- When a sort order is set for character string data in a SELECT statement in which the ORDER BY clause is specified:
  a_rdb_SQL_ATTR_ORDER_MODE
- When the transaction access mode is set:
  a_rdb_SQL_ATTR_ACCESS_MODE

Value

Specifies the address at which the attribute information is stored.

• When the transaction isolation level is set

To set the transaction isolation level, specify the address of an unsigned short area that contains one of the following values:

- For READ COMMITTED:
  a_rdb_SQL_TXN_READ_COMMITTED
- For REPEATABLE READ:
  a_rdb_SQL_TXN_REPEATABLE_READ

- If the specified value is changed into unspecified status (in which the transaction isolation level has never been set by using this CLI function):
  a_rdb_SQL_TXN_UNSPECIFIED

• When a sort order is set for character string data in a SELECT statement in which the ORDER BY clause is specified

Specify the address of the unsigned short area containing one of the following values:

- Sorting of character string data by bytecode:
  a_rdb_SQL_ORDER_MODE_BYTE
- Sorting of character string data by sort code (ISO/IEC 14651:2011 compliance):
  a_rdb_SQL_ORDER_MODE_ISO
- If the specified value is changed into unspecified status (in which the sort order for character string data has never been set by using this CLI function):
  a_rdb_SQL_ORDER_MODE_UNSPECIFIED

• When the transaction access mode is set:

Specify the address of an unsigned short area in which one of the following values is set:

- Read/write mode:
  a_rdb_SQL_ACCESS_MODE_READ_WRITE
- Read-only mode:
  a_rdb_SQL_ACCESS_MODE_READ_ONLY
- If the specified value is changed into unspecified status (in which the transaction access mode has never been set by using this CLI function):
  a_rdb_SQL_ACCESS_MODE_UNSPECIFIED

Option

Specifies NULL.

(4) Return value

1. If a_rdb_SQLSetConnectAttr() terminates normally, a_rdb_RC_SQL_SUCCESS is returned.
2. If the connection attributes have been set successfully but the disk containing client message log files has become full, a_rdb_RC_SQL_WARNING is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. a_rdb_SQLSetConnectAttr() cannot be executed in the following cases:
   - Connection has not been established.
   - The transaction has not been settled.
   - An invalid connection attribute is specified.
   - An invalid transaction isolation level is specified.
   - An invalid sort order for character string data is specified.
   - An invalid transaction access mode is specified.
2. The transaction isolation level is determined in the priority order described below. A smaller number represents a higher priority (for example, 1 has a higher priority than 2).

1. Transaction isolation level specified by a_rdb_SQLSetConnectAttr()
2. Transaction isolation level specified in the adb_clt_trn_iso_lv client definition operand
3. Transaction isolation level specified in the adb_sys_trn_iso_lv server definition operand

3. The sort order for character string data is determined in the priority order shown below. A smaller number represents a higher the priority (for example, 1 has a higher priority than 2):
   1. Sort order specified by a_rdb_SQLSetConnectAttr()
   2. Sort order specified in the adb_clt_sql_order_mode client definition operand
   3. Sort order specified in the adb_sql_order_mode server definition operand
4. The transaction access mode is determined in the priority order described below. A smaller number represents a higher priority (for example, 1 has a higher priority than 2).
   1. Transaction access mode specified with a_rdb_SQLSetConnectAttr()
   2. Transaction access mode specified in the adb_clt_trn_access_mode operand in the client definition

19.2.4 a_rdb_SQLDisconnect() (close a connection)

(1) Function

This CLI function closes a connection by terminating the current transaction normally, setting a synchronization point, and generating a commitment unit. The function then disconnects the application program from the HADB server.

(2) Format

```
signed short a_rdb_SQLDisconnect
(
  void                          *ConnectionHandle,   /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

Option

Specifies NULL.

(4) Return value

1. If a_rdb_SQLDisconnect() terminates normally, a_rdb_RC_SQL_SUCCESS is returned.
2. If the connection was closed successfully, but the disk containing server message log files or client message log files has become full, a_rdb_RC_SQL_WARNING is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. If the transaction has not terminated when `a_rdb_SQLDisconnect()` is executed, the HADB server automatically executes `COMMIT`, and then the connection with the HADB server is closed.
2. If an internally executed `COMMIT` has failed, the HADB server executes `ROLLBACK` and cancels the transaction, and then the connection with the HADB server is closed.
3. If the application program terminates without `a_rdb_SQLDisconnect()` being executed, the HADB server executes `ROLLBACK`, and then the connection with the HADB server is closed.
4. `a_rdb_SQLDisconnect()` cannot be executed if a connection has not been established.

19.2.5 a_rdb_SQLFreeConnect() (release a connection handle)

(1) Function

This CLI function releases the connection handle for a closed connection.

(2) Format

```
signed short a_rdb_SQLFreeConnect
(
  void                          *ConnectionHandle,    /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLFreeConnect()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the connection handle is released successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

If `a_rdb_SQLFreeConnect()` is executed without `a_rdb_SQLDisconnect()` having been executed, processing equivalent to `a_rdb_SQLDisconnect()` is performed as an extension of `a_rdb_SQLFreeConnect()`, and then the connection handle is released.

19.3 CLI functions for controlling transactions

This section explains the CLI functions used for controlling transactions.

19.3.1 a_rdb_SQLCancel() (cancel SQL processing)

(1) Function

This CLI function cancels the current SQL processing. The following SQL processing (CLI functions) can be canceled by `a_rdb_SQLCancel()`:

- `a_rdb_SQLCloseCursor()`
- `a_rdb_SQLExecDirect()`
- `a_rdb_SQLExecute()`
- `a_rdb_SQLFetch()`
- `a_rdb_SQLPrepare()`

If you execute `a_rdb_SQLCancel()` during execution of a CLI function other than the preceding ones, `a_rdb_SQLCancel()` terminates normally.

(2) Format

```
signed short a_rdb_SQLCancel
(
  void                          *ConnectionHandle,   /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`Option`

Specifies `NULL`.

(4) Return value

If `a_rdb_SQLCancel()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.

(5) Notes

1. You must execute `a_rdb_SQLCancel()` in a different thread from the one used for the SQL processing.
2. Execution of `a_rdb_SQLCancel()` does not set any SQL results information.
3. If `a_rdb_SQLCancel()` results in an error, no message is output to the client message log files.

4. When SQL processing is canceled successfully, the canceled SQL processing is rolled back and SQLCODE is returned.
5. Normal termination of a_rdb_SQLCancel() does not mean that cancellation has been successful, because the cancellation processing is performed asynchronously with a_rdb_SQLCancel().
6. a_rdb_SQLCancel() cannot be executed if connection has not been established.

19.3.2 a_rdb_SQLEndTran() (terminate the transaction)

(1) Function

This CLI function terminates the transaction. When the transaction is completed, all statement handles in effect at that point are released.

(2) Format

```
signed short a_rdb_SQLEndTran
(
  void                          *ConnectionHandle,    /* In */
  unsigned short                 CompletionType,      /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

CompletionType

Specifies one of the following values:

- a_rdb_SQL_COMMIT: Specifies that COMMIT (normal termination of transaction) is to be executed.
- a_rdb_SQL_ROLLBACK: Specifies that ROLLBACK (cancellation of transaction) is to be executed.

Option

Specifies NULL.

(4) Return value

1. If a_rdb_SQLEndTran() terminates normally, a_rdb_RC_SQL_SUCCESS is returned.
2. If the transaction is terminated successfully, but the disk containing server message log files or client message log files has become full, a_rdb_RC_SQL_WARNING is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. If a row fetching error is detected during the cursor close processing that is executed as an extension of the COMMIT processing, ROLLBACK processing is performed automatically, in which case the transaction does not terminate normally. If COMMIT fails for any other reason, the HADB server terminates abnormally.

2. If ROLLBACK fails, the HADB server terminates abnormally.
3. a_rdb_SQLEndTran() cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid value is specified for CompletionType.
   - A row fetching error is detected during the cursor close processing that is executed as an extension of the COMMIT processing.
4. If COMMIT or ROLLBACK processing is performed on the transaction as a result of executing a_rdb_SQLEndTran(), the following occurs:
   - All open cursors are closed.
   - Any preprocessed SQL statement is ignored.
   - Statement handles are all released.

19.4 CLI functions for execution of SQL statements

This section explains the CLI functions used for execution of SQL statements.

19.4.1 a_rdb_SQLAllocStmt() (allocate a statement handle)

(1) Function

This CLI function allocates a statement handle (handle for an SQL statement).

(2) Format

```
signed short a_rdb_SQLAllocStmt
(
  void                          *ConnectionHandle,   /* In  */
  void                         **StatementHandle,    /* Out */
  void                          *Option              /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies the address at which the statement handle is to be set.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLAllocStmt()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If a statement handle is allocated successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. `a_rdb_SQLAllocStmt()` cannot be executed if connection has not been established.
2. If the transaction fails to start, the HADB server terminates abnormally.

19.4.2 a_rdb_SQLBindArrayParams() (bind dynamic parameters in batch mode)

(1) Function

This CLI function binds (associates) the dynamic parameters in an SQL statement with an area for specifying their values.

This CLI function binds multiple sets of dynamic parameters in any of the following SQL statements in the batch mode:

- `DELETE` statement
- `INSERT` statement
- `UPDATE` statement

You use this CLI function to perform batch transfer of dynamic parameter values. For details about batch transfer of dynamic parameter values, see 5.15 Batch transfer of dynamic parameter values.

(2) Format

```
signed short a_rdb_SQLBindArrayParams
(
  void                          *ConnectionHandle,  /* In  */
  void                          *StatementHandle,   /* In  */
  int                            ArrayCount,        /* In  */
  unsigned short                 ParameterCount,    /* In  */
  a_rdb_SQLParameterInfo_t      *ParameterInfo[],   /* In  */
  unsigned long long             RowCount[],        /* Out */
  void                          *Option             /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ArrayCount`

Specifies the number of `ParameterInfo` and `RowCount` arrays.

`ParameterCount`

Specifies the number of dynamic parameters to which values are to be assigned.

Specify the number of dynamic parameters acquired by `a_rdb_SQLNumParams()`.

`ParameterInfo[]`

Specifies an array of addresses for the parameter information area in which the addresses of the value storage areas are to be set. Provide an area in which as many `a_rdb_SQLParameterInfo_t` structures as is specified for `ParameterCount` can be placed consecutively for each array, and then specify the start addresses of those areas as the values for the array.

For details about the `a_rdb_SQLParameterInfo_t` structure, see 19.7.5 a_rdb_SQLParameterInfo_t structure (parameter information).

RowCount[]

Specifies the start addresses of the array in which the numbers of rows resulting from SQL statement processing are stored. If NULL is specified, no numbers of results rows are acquired.

After an SQL statement has been executed by a_rdb_SQLExecute(), the number of results rows obtained by using each set of dynamic parameter values is stored in each array.

If an error occurs during SQL statement execution, the numbers of results rows obtained up to the point of the error are stored. However, if the processing is rolled back because of an SQL statement error, zero is set in all arrays.

Option

Specifies NULL.

(4) Return value

1. If a_rdb_SQLBindArrayParams() terminates normally, a_rdb_RC_SQL_SUCCESS is returned.
2. If the dynamic parameters have been bound successfully but the disk containing client message log files has become full, a_rdb_RC_SQL_WARNING is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must apply boundary alignment to the start addresses of the data storage areas of the a_rdb_SQLParameterInfo_t structure. For details about boundary alignment, see (2) Boundary alignment in 18.2.6 Notes about using the CLI functions.
2. When a_rdb_SQLBindArrayParams() is used to associate dynamic parameters and then a_rdb_SQLExecute() is executed, the processing continues unless the SQL statement results in an error. Even if a specified set of dynamic parameter values results in zero as the number of results rows during processing, the processing still continues.
3. The total number of results rows that have been stored in the RowCount[] array is stored in RowCount of the a_rdb_SQLResultInfo_t structure.
4. a_rdb_SQLBindArrayParams() cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - A value smaller than zero is specified as the number of arrays.
   - The number of dynamic parameters specified does not match the number of dynamic parameters acquired by a_rdb_SQLNumParams().
   - An attempt was made to execute a_rdb_SQLBindArrayParams() on an SQL statement that is not DELETE, INSERT, or UPDATE.

19.4.3 a_rdb_SQLBindCols() (associate retrieval result columns)

(1) Function

This CLI function binds (associates) the retrieval result columns with an area for storing the values fetched from the retrieval result columns in the batch mode. Once a retrieval result column has been associated, the column's value is stored when the cursor is manipulated by execution of `a_rdb_SQLFetch()`.

You can also execute `a_rdb_SQLBindCols()` for retrieval results that are not columns.

(2) Format

```
signed short a_rdb_SQLBindCols
(
  void                          *ConnectionHandle,   /* In  */
  void                          *StatementHandle,    /* In  */
  unsigned short                 ColumnCount,        /* In  */
  a_rdb_SQLColumnInfo_t         *ColumnInfo,         /* In  */
  void                          *Option              /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ColumnCount`

Specifies the number of columns whose values are to be fetched.

Specify the number of retrieval result columns that was acquired by `a_rdb_SQLNumResultCols()`.

`ColumnInfo`

Specifies the start address of the column information area in which the addresses of the value storage areas are to be set.

Provide an area in which as many `a_rdb_SQLColumnInfo_t` structures as is specified for `ColumnCount` can be placed consecutively.

For details about the `a_rdb_SQLColumnInfo_t` structure, see 19.7.1 a_rdb_SQLColumnInfo_t structure (column information).

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLBindCols()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the retrieval result columns are associated successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. If the data type of obtained data is `CHAR`, the data storage area for the `a_rdb_SQLColumnInfo_t` structure stores the obtained data followed by a null character. The null character here refers to `0x00`.
2. You must apply boundary alignment to the start addresses of the data storage areas of the `a_rdb_SQLColumnInfo_t` structure. For details about boundary alignment, see (2) Boundary alignment in 18.2.6 Notes about using the CLI functions.
3. The value for the area for storing either the data storage area length or the indicator value for the `a_rdb_SQLColumnInfo_t` structure is set according to the acquired data, as shown in the table below. In the table, *BL* means the length of the specified data storage area and *DL* means the size of the area required for storing the acquired data (in bytes).

   Table 19-3: Value for the area for storing the data storage area length or the indicator value

No.	Stored data	Data structure	Data type	Relationship between BL and DL	Data storage area length or indicator value
1	Null value	Any	Any	Any	`a_rdb_SQL_NULL_DATA`
2	Non-null value	Any	Any	BL = DL	`a_rdb_SQL_NOT_NULL_DATA`
3				Other	-- (SQL error)#

   #

   This means that an SQL error occurred and the processing for setting a value in the indicator was not performed, resulting in an undefined value.

4. `a_rdb_SQLBindCols()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - The value specified for the number of retrieval result columns is not the value acquired by `a_rdb_SQLNumResultCols()`.
   - The data storage area length in the `a_rdb_SQLColumnInfo_t` structure is not the length required for storing the acquired data.

19.4.4 a_rdb_SQLBindParams() (associate dynamic parameters)

(1) Function

This CLI function binds (associates) the dynamic parameters in an SQL statement with an area for specifying their values.

(2) Format

```
signed short a_rdb_SQLBindParams
(
  void                          *ConnectionHandle,    /* In  */
  void                          *StatementHandle,     /* In  */
  unsigned short                 ParameterCount,      /* In  */
  a_rdb_SQLParameterInfo_t      *ParameterInfo,       /* In  */
  void                          *Option               /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ParameterCount`

Specifies the number of dynamic parameters to which values are to be assigned.

Specify the number of dynamic parameters acquired by `a_rdb_SQLNumParams()`.

`ParameterInfo`

Specifies the start address of the parameter information area in which the addresses of the value storage areas are to be set.

Provide an area in which as many `a_rdb_SQLParameterInfo_t` structures as is specified for `ParameterCount` can be placed consecutively.

For details about the `a_rdb_SQLParameterInfo_t` structure, see 19.7.5 a_rdb_SQLParameterInfo_t structure (parameter information).

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLBindParams()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the dynamic parameters are associated successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must apply boundary alignment to the start addresses of the data storage areas of the `a_rdb_SQLParameterInfo_t` structure. For details about boundary alignment, see (2) Boundary alignment in 18.2.6 Notes about using the CLI functions.
2. `a_rdb_SQLBindParams()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - The specified number of dynamic parameters is not the value acquired by `a_rdb_SQLNumParams()`.

19.4.5 a_rdb_SQLCloseCursor() (close the cursor)

(1) Function

This CLI function closes the cursor and terminates row fetching.

If there is a processing real thread that is fetching rows, this function terminates that processing real thread.

(2) Format

```
signed short a_rdb_SQLCloseCursor
(
  void                          *ConnectionHandle,    /* In */
  void                          *StatementHandle,     /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLCloseCursor()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the cursor is closed successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. `a_rdb_SQLCloseCursor()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - A row fetching error is detected during the cursor close processing.
2. If the transaction is terminated, all cursors that are open at that point are closed. If database update processing is implicitly invalidated, the cursors are also all closed, in which case the `KFAA51001-E` or `KFAA51002-E` message is output to the server message log file and client message log file.

19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns)

(1) Function

This CLI function acquires information about the retrieval result columns. The information that is acquired includes the following:

- Column names of the retrieval result columns
- Data types of the retrieval result columns

- Maximum numbers of elements in the retrieval result columns
- Data lengths of the retrieval result columns

(2) Format

```
signed short a_rdb_SQLDescribeCols
(
  void                          *ConnectionHandle,    /* In  */
  void                          *StatementHandle,     /* In  */
  unsigned short                 ColumnCount,         /* In  */
  a_rdb_SQLColumnInfo_t         *ColumnInfo,          /* Out */
  void                          *Option               /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ColumnCount`

Specifies the number of retrieval result columns for which information is to be acquired.

Specify the number of retrieval result columns that was acquired by `a_rdb_SQLNumResultCols()`.

`ColumnInfo`

Specifies the start address of the column information area in which the addresses of the areas for acquiring retrieval result column information is to be set.

Provide an area in which as many `a_rdb_SQLColumnInfo_t` structures as is specified for `ColumnCount` can be placed consecutively.

For details about the `a_rdb_SQLColumnInfo_t` structure, see 19.7.1 a_rdb_SQLColumnInfo_t structure (column information).

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLDescribeCols()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If information about the retrieval result columns is acquired successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. The names of the retrieval result columns are acquired in the name storage area of the `a_rdb_SQLNameInfo_t` structure. For details about the names of retrieval result columns, see *Rules* in *Specification format and rules for the SELECT statement* in the manual *HADB SQL Reference*.

2. Specify for the length of the name storage area of the a_rdb_SQLNameInfo_t structure a value that is greater by the byte length of one null character than the actual length (null character means 0x00).
3. If the length of the name storage area of the a_rdb_SQLNameInfo_t structure is equal to or greater than the length of the retrieval result column name (in bytes) plus the byte length of one null character, a character string to which one null character is appended is acquired as the retrieval result column name.
4. If the length of the name storage area of the a_rdb_SQLNameInfo_t structure is less than the length of the retrieval result column name (in bytes) plus the byte length of one null character, the trailing part of the retrieval result column name (the portion that does not fit in the name storage area) is truncated. As much of the leading part of the retrieval result column name as fits in the name storage area is stored with one null character appended.
5. The length of a retrieval result column name (in bytes) is acquired in the column name length storage area of the a_rdb_SQLNameInfo_t structure.
6. A code indicating the data type (data type code) of a retrieval result column is stored in the data type code storage area of the a_rdb_SQLDataType_t structure. For details about the data type code for each data type, see *List of data types* in the manual *HADB SQL Reference*.
7. The maximum number of elements in a retrieval result column is stored in the maximum elements count storage area of the a_rdb_SQLDataType_t structure. The maximum number of elements in a retrieval result column is always 1.
8. Information about the data length of a retrieval result column is stored in the column length storage area and column length attribute storage area of the a_rdb_SQLDataType_t structure. The value to be acquired in each area depends on the data type of the retrieval result column, as shown in the following table.

Table 19-4: Column length and column length attribute

No.	Data type	Column length	Column length attribute
1	INTEGER	8	0
2	SMALLINT	4	0
3	DECIMAL(*m*,*n*) #1	*m*	*n*
4	DOUBLE PRECISION	8	0
5	CHAR(*n*)	*n*	0
6	VARCHAR(*n*)	*n*	0
7	DATE	4	0
8	TIME(*p*) #2	$3 + \uparrow p \div 2 \uparrow$	*p*
9	TIMESTAMP(*p*) #2	$7 + \uparrow p \div 2 \uparrow$	*p*
10	BINARY(*n*)	*n*	0
11	VARBINARY(*n*)	*n*	0
12	ROW	*row-length*	0

Legend:

m, *n*, *p*: Positive integer

#1

If the data type is DECIMAL, the precision is set as the column length and the scaling is set as the column length attribute.

#2

If the data type is TIME or TIMESTAMP, the data length is set as the column length and the length of the fractional seconds is set as the column length attribute.

9. a_rdb_SQLDescribeCols() cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - The value specified as the number of retrieval result columns is not the value acquired by a_rdb_SQLNumResultCols().

19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information)

(1) Function

This CLI function acquires an SQL statement's dynamic parameter information. The information that is acquired includes the following:

- Data types of the dynamic parameters
- Maximum numbers of elements in the dynamic parameters
- Data lengths of the dynamic parameters

(2) Format

```
signed short a_rdb_SQLDescribeParams
(
  void                          *ConnectionHandle,   /* In  */
  void                          *StatementHandle,    /* In  */
  unsigned short                 ParameterCount,     /* In  */
  a_rdb_SQLParameterInfo_t      *ParameterInfo,      /* Out */
  void                          *Option              /* In  */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

StatementHandle

Specifies a statement handle.

ParameterCount

Specifies the number of dynamic parameters for which information is to be acquired.

Specify the number of dynamic parameters acquired by a_rdb_SQLNumParams().

ParameterInfo

Specifies the start address of the parameter information area in which the addresses of the various parameter information acquisition areas are to be set.

Provide an area in which as many `a_rdb_SQLParameterInfo_t` structures as is specified for `ParameterCount` can be placed consecutively.

For details about the `a_rdb_SQLParameterInfo_t` structure, see 19.7.5 a_rdb_SQLParameterInfo_t structure (parameter information).

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLDescribeParams()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If dynamic parameter information is acquired successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. Codes assumed by HADB (data type codes) indicating the data types of the dynamic parameters are acquired in the data type code storage area of the `a_rdb_SQLDataType_t` structure. For details about the data type code for each data type, see *List of data types* in the manual *HADB SQL Reference*.
2. The maximum number of elements in a dynamic parameter is acquired in the maximum elements count storage area of the `a_rdb_SQLDataType_t` structure. The maximum number of elements in a dynamic parameter is always 1.
3. Information about the data length of a dynamic parameter is stored in the parameter length storage area and the parameter attribute storage area of the `a_rdb_SQLDataType_t` structure. The value to be acquired in each area depends on the data type of the dynamic parameter assumed by HADB, as shown in the following table.

Table 19-5: Parameter length and parameter length attribute

No.	Data type	Parameter length	Parameter length attribute
1	`INTEGER`	8	0
2	`SMALLINT`	4	0
3	`DECIMAL(`*m*`,`*n*`)` #1	*m*	*n*
4	`DOUBLE PRECISION`	8	0
5	`CHAR(`*n*`)`	*n*	0
6	`VARCHAR(`*n*`)`	*n*	0
7	`DATE`	4	0
8	`TIME(`*p*`)` #2	$3 + \uparrow p \div 2 \uparrow$	*p*
9	`TIMESTAMP(`*p*`)` #2	$7 + \uparrow p \div 2 \uparrow$	*p*
10	`BINARY(`*n*`)`	*n*	0
11	`VARBINARY(`*n*`)`	*n*	0
12	`ROW`	*row-length*	0

Legend:

m, *n*, *p*: Positive integer

#1

If the data type is DECIMAL, the precision is set as the parameter length and the scaling is set as the parameter length attribute.

#2

If the data type is TIME or TIMESTAMP, the data length is set as the column length and the length of the fractional seconds is set as the column length attribute.

4. a_rdb_SQLDescribeParams() cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
   - The specified number of dynamic parameters is not the value acquired by a_rdb_SQLNumParams().

19.4.8 a_rdb_SQLExecDirect() (preprocess and execute an SQL statement)

(1) Function

This CLI function preprocesses and executes an SQL statement. The function can execute the following SQL statements:

- DELETE statement#
- INSERT statement#
- PURGE CHUNK statement#
- TRUNCATE TABLE statement
- UPDATE statement#
- Definition SQL statements

#

a_rdb_SQLExecDirect() cannot be executed if a dynamic parameter is specified.

(2) Format

```
signed short a_rdb_SQLExecDirect
(
  void                          *ConnectionHandle,   /* In */
  void                          *StatementHandle,    /* In */
  char                          *StatementText,      /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

ConnectionHandle

Specifies a connection handle.

StatementHandle

Specifies a statement handle.

StatementText

Specifies the text of the SQL statement that is to be preprocessed and executed, expressed as a character string in C or C++.

Option

Specifies NULL.

(4) Return value

1. If a_rdb_SQLExecDirect() terminates normally, a_rdb_RC_SQL_SUCCESS or a_rdb_RC_SQL_NO_DATA is returned.
2. If the SQL statement is preprocessed and executed successfully, but the disk containing server message log files or client message log files has become full, a_rdb_RC_SQL_WARNING is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

a_rdb_SQLExecDirect() cannot be executed in the following cases:

- Connection has not been established.
- An invalid statement handle is specified.

19.4.9 a_rdb_SQLExecute() (execute a preprocessed SQL statement)

(1) Function

This CLI function executes a preprocessed SQL statement. The function can execute the following SQL statements:

- DELETE statement
- INSERT statement
- PURGE CHUNK statement
- TRUNCATE TABLE statement
- SELECT statement
- UPDATE statement
- Definition SQL statements

When a_rdb_SQLExecute() is executed on a SELECT statement, a cursor is opened.

(2) Format

```
signed short a_rdb_SQLExecute
(
  void                          *ConnectionHandle,   /* In */
  void                          *StatementHandle,    /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLExecute()` terminates normally, `a_rdb_RC_SQL_SUCCESS` or `a_rdb_RC_SQL_NO_DATA` is returned.
2. If execution of the SQL statement is completed successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. `a_rdb_SQLExecute()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
2. A preprocessed SQL statement must be available for the specified statement handle.
3. To open a cursor that was opened before, you must first close it, and then re-open it.
4. To use `a_rdb_SQLFetch()` to a fetch row, you must first open the cursor, and then fetch the row.
5. If `COMMIT` or `ROLLBACK` (including implicit rollback) is executed, all cursors that are open at that point are closed.

19.4.10 a_rdb_SQLFetch() (fetch a row)

(1) Function

This CLI function advances the cursor to the next row. If columns are bound, the function reads the column values on the row pointed to by the cursor into the fetch target specified in the fetch target list.

(2) Format

```
signed short a_rdb_SQLFetch
(
  void                          *ConnectionHandle,   /* In */
  void                          *StatementHandle,    /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLFetch()` terminates normally, `a_rdb_RC_SQL_SUCCESS` or `a_rdb_RC_SQL_NO_DATA` is returned.
2. If a row is fetched successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. `a_rdb_SQLFetch()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
2. You must first use `a_rdb_SQLExecute()` to open the cursor that is to be used.

19.4.11 a_rdb_SQLFreeStmt() (release a statement handle)

(1) Function

This CLI function releases a statement handle allocated by `a_rdb_SQLAllocStmt()`.

When a statement handle is released, any open cursor is also closed.

(2) Format

```
signed short a_rdb_SQLFreeStmt
(
  void                          *ConnectionHandle,   /* In */
  void                          *StatementHandle,    /* In */
  void                          *Option              /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLFreeStmt()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the statement handle is released successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

`a_rdb_SQLFreeStmt()` cannot be executed in the following cases:

- Connection has not been established.
- An invalid statement handle is specified.
- A row fetching error is detected during the cursor close processing that is executed as an extension of the statement handle release processing.

19.4.12 a_rdb_SQLNumParams() (acquire the number of dynamic parameters)

(1) Function

CLI function acquires the number of dynamic parameters in an SQL statement.

(2) Format

```
signed short a_rdb_SQLNumParams
(
  void                          *ConnectionHandle,    /* In  */
  void                          *StatementHandle,     /* In  */
  unsigned short                *ParameterCount,      /* Out */
  void                          *Option               /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ParameterCount`

Specifies the address at which the number of dynamic parameters is to be stored.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLNumParams()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the number of dynamic parameters is acquired successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

`a_rdb_SQLNumParams()` cannot be executed in the following cases:

- Connection has not been established.
- An invalid statement handle is specified.

19.4.13 a_rdb_SQLNumResultCols() (acquire the number of retrieval result columns)

(1) Function

This CLI function acquires the number of retrieval result columns.

(2) Format

```
signed short a_rdb_SQLNumResultCols
(
  void                          *ConnectionHandle,   /* In  */
  void                          *StatementHandle,    /* In  */
  unsigned short                *ColumnCount,        /* Out */
  void                          *Option              /* In  */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`ColumnCount`

Specifies the address at which the number of retrieval result columns is to be stored.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLNumResultCols()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the number of retrieval result columns is acquired successfully, but the disk containing client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

1. `a_rdb_SQLNumResultCols()` cannot be executed in the following cases:
   - Connection has not been established.
   - An invalid statement handle is specified.
2. If the SQL statement allocated to the statement handle is not a `SELECT` statement, `0` is stored in the area for storing the number of retrieval result columns.

19.4.14 a_rdb_SQLPrepare() (preprocess an SQL statement)

(1) Function

This CLI function preprocesses an SQL statement. The function can preprocess the following SQL statements:

- `DELETE` statement
- `INSERT` statement
- `PURGE CHUNK` statement
- `TRUNCATE TABLE` statement
- `SELECT` statement
- `UPDATE` statement
- Definition SQL statements

When `a_rdb_SQLPrepare()` is executed, the specified SQL statement is preprocessed so that is becomes executable and then the SQL statement is allocated to the specified statement handle.

(2) Format

```
signed short a_rdb_SQLPrepare
(
  void                          *ConnectionHandle,    /* In */
  void                          *StatementHandle,     /* In */
  char                          *StatementText,       /* In */
  void                          *Option               /* In */
)
```

(3) Arguments

`ConnectionHandle`

Specifies a connection handle.

`StatementHandle`

Specifies a statement handle.

`StatementText`

Specifies the text of the SQL statement that is to be preprocessed, expressed as a character string in C or C++.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_SQLPrepare()` terminates normally, `a_rdb_RC_SQL_SUCCESS` is returned.
2. If the SQL statement is preprocessed successfully, but the disk containing server message log files or client message log files has become full, `a_rdb_RC_SQL_WARNING` is returned.
3. If an error occurs while messages cannot be output to the client message log file, the error cause code is returned. For details about the error cause code, see 19.8 Return values of the CLI functions.

(5) Notes

`a_rdb_SQLPrepare()` cannot be executed in the following cases:

- Connection has not been established.
- An invalid statement handle is specified.

19.5 CLI functions for data type conversion

This section explains the CLI functions that are used for data type conversion.

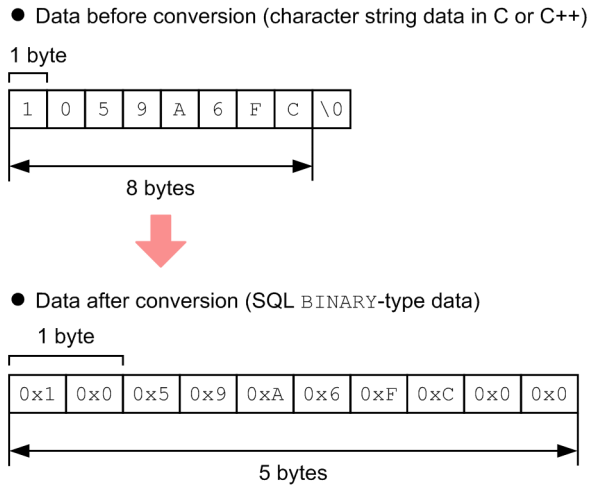
19.5.1 a_rdb_CNV_charBINARY() (convert to BINARY-type data)

(1) Function

This CLI function converts character string data in C or C++ (binary or hexadecimal) to SQL `BINARY` type data. The following figure shows an example of conversion from character string data to `BINARY`-type data.

Figure 19-1: Example of conversion from character string data to BINARY-type data

Example: Convert an 8-byte hexadecimal character string to `BINARY`-type data.



Important

The character string data before conversion must consist of the following values:

- When the character string data before conversion is in binary
  `0` and `1`
- When the character string data before conversion is in hexadecimal
  `0` to `9` and `A` to `F` (or `a` to `f`)

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charBINARY
(
  char                          *char_Data,          /* In  */
  unsigned int                   char_Length,        /* In  */
  unsigned short                 char_Type,          /* In  */
  unsigned short                 BINARY_Length,      /* In  */
  unsigned char                 *BINARY_Data,        /* Out */
  unsigned short                 BufferLength,       /* In  */
  void                          *Option              /* In  */
)
```

(3) Arguments

`char_Data`

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

`char_Length`

Specifies the length (in bytes) of the character string data before conversion. Specify the following value:

- When the character string data before conversion is in binary
  1 to *value of BINARY_Length* × 8
- When the character string data before conversion is in hexadecimal
  1 to *value of BINARY_Length* × 2

`char_Type`

Specifies the format of the character string data before conversion. Specify the following value:

- When the character string data before conversion is in binary
  `a_rdb_CNV_CHAR_TYPE_BINARY`
- When the character string data before conversion is in hexadecimal
  `a_rdb_CNV_CHAR_TYPE_HEX`

`BINARY_Length`

Specifies the length (in bytes) of the `BINARY`-type data after conversion. The permitted values are from 1 to 32,000.

`BINARY_Data`

Specifies the start address of the area where the `BINARY`-type data after conversion is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area for storing the `BINARY`-type data after conversion. Specify the same value as for `BINARY_Length`.

`Option`

Specifies `NULL`. Any value that is specified is ignored.

(4) Return value

1. If `a_rdb_CNV_charBINARY()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must ensure that there is no overlap between the area for storing the data before conversion (`char_Data`) and the area for storing the data after conversion (`BINARY_Data`).
2. The length of the character string data before conversion must satisfy one of the following conditions:
   - When the character string data before conversion is in binary: Multiple of 8
   - When the character string data before conversion is in hexadecimal: Multiple of 2
3. The character string data before conversion is converted to `BINARY`-type data, and then stored from the beginning of the storage area. If the character string data to be converted is smaller than the following value, `0x00` is padded to the right of the data:

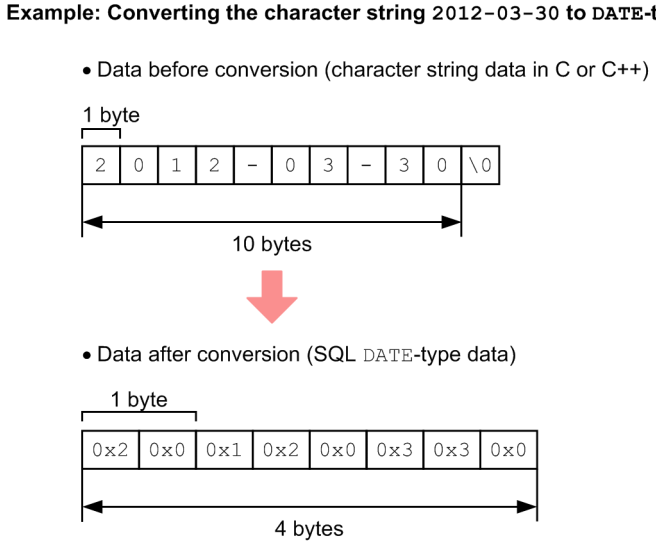
- When the character string data before conversion is in binary: *Value of BINARY_Length* × 8
- When the character string data before conversion is in hexadecimal: *Value of BINARY_Length* × 2

19.5.2 a_rdb_CNV_charDATE() (convert to DATE-type data)

(1) Function

This CLI function converts character string data in C or C++ to SQL `DATE`-type data. The following figure shows an example of conversion from character string data to `DATE`-type data.

Figure 19-2: Example of conversion from character string data to DATE-type data



Important

The character string data before conversion must be in a predefined input representation that represents a date. For details about the predefined input representations for dates, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.

Example:

The character string data that represents September 30, 2014, must be in one of the following formats:

```
2014-09-30
2014/09/30
```

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charDATE
(
  char              *char_Data,                 /* In  */
  unsigned char     *DATE_Data,                 /* Out */
  unsigned short     BufferLength,              /* In  */
  void              *Option                     /* In  */
)
```

(3) Arguments

`char_Data`

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

`DATE_Data`

Specifies the start address of the area where the `DATE`-type data after conversion is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area where the `DATE`-type data after conversion is to be stored. Specify a value of `4`.

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_CNV_charDATE()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

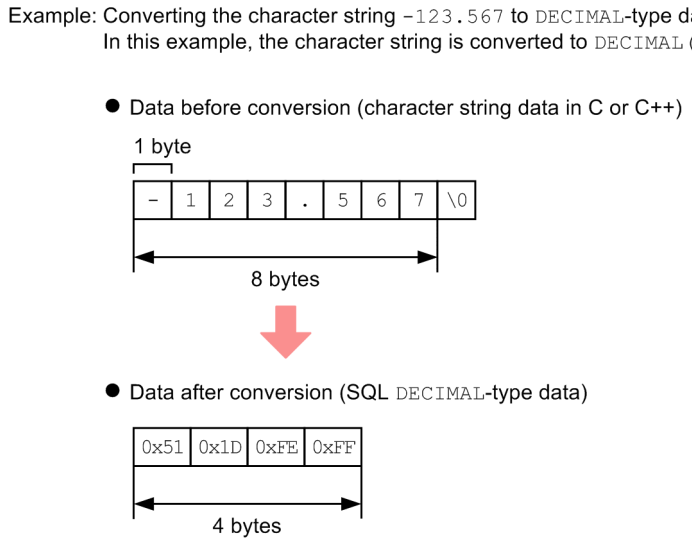
You must ensure that there is no overlap between the area for storing the data before conversion (`char_Data`) and the area for storing the data after conversion (`DATE_Data`).

19.5.3 a_rdb_CNV_charDECIMAL() (convert to DECIMAL-type data)

(1) Function

This CLI function converts character string data in C or C++ to SQL `DECIMAL`-type data. The following figure shows an example of conversion from character string data to `DECIMAL`-type data.

Figure 19-3: Example of conversion from character string data to DECIMAL-type data



Important

The character string data before conversion must be in the following format:

'[ΔΔ...Δ][+|-][*aa....aa*][.][*bb....bb*]'

[ΔΔ...Δ]: Zero or more spaces

[+|-]: Plus or minus sign (if the sign is omitted, a plus sign is assumed)

[*aa....aa*]: Integer part

[.]: Decimal point (if there is no fractional part, the decimal point can be omitted)

[*bb....bb*]: Fractional part

Please note the following:

- You can specify only an integer part [*aa....aa*]; if you specify a fractional part, [*bb....bb*], you must also specify the decimal point [.].
- The length of the integer part [*aa....aa*] and the fractional part [*bb....bb*] must not exceed 38 places.
- If the decimal point (period) is omitted when you specify an integer part only, a decimal point is assumed at the end of the data obtained after conversion.

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charDECIMAL
(
  char                 *char_Data,                /* In  */
  unsigned short        char_Length,              /* In  */
  unsigned short        DECIMAL_Precision,        /* In  */
  unsigned short        DECIMAL_Scale,            /* In  */
  unsigned char        *DECIMAL_Data,             /* Out */
  unsigned short        BufferLength,             /* In  */
  void                 *Option                    /* In  */
)
```

(3) Arguments

char_Data

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

char_Length

Specifies the length (in bytes) of the character string data before conversion. The permitted value range is from1 to 41.

DECIMAL_Precision

Specifies the precision of the DECIMAL-type data (total number of digits) after conversion. The permitted value range is from 1 to 38.

Specify the same precision as for the dynamic parameter that is to be associated.

DECIMAL_Scale

Specifies the scaling of the DECIMAL-type data (number of decimal places) after conversion. The permitted value range is from 0 to 38.

Specify the same scaling as for the dynamic parameter that is to be associated.

DECIMAL_Data

Specifies the start address of the area where the DECIMAL-type data after conversion is to be stored.

BufferLength

Specifies the length (in bytes) of the area where the DECIMAL-type data after conversion is to be stored. Specify the following value:

Table 19-6: Value of BufferLength

No.	Value of DECIMAL_Precision	Value of BufferLength
1	1 ≤ DECIMAL_Precision ≤ 4	2
2	5 ≤ DECIMAL_Precision ≤ 8	4
3	9 ≤ DECIMAL_Precision ≤ 16	8
4	17 ≤ DECIMAL_Precision ≤ 38	16

Option

Specifies NULL.

(4) Return value

1. If a_rdb_CNV_charDECIMAL() terminates normally, a_rdb_RC_CNV_SUCCESS is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must ensure that there is no overlap between the area for storing the data before conversion (char_Data) and the area for storing the data after conversion (DECIMAL_Data).
2. If the data before conversion is equivalent to "-0.", "+0." consisting of the same number of characters is assumed. For example, if the data to be converted is "-0.0", "+0.0" is assumed; if the data before conversion is "-0.0000", "+0.0000" is assumed.
3. The character string data before conversion is converted to DECIMAL-type data, and then stored from the beginning of the storage area. If the integer part cannot be stored, an error results. If the fractional part cannot be stored, the excess places are discarded.
4. You can specify character string data in which a zero has been added to the integer part only if the precision and scaling match. In this case, for DECIMAL_Precision, specify the number of digits without the zero in the integer part.

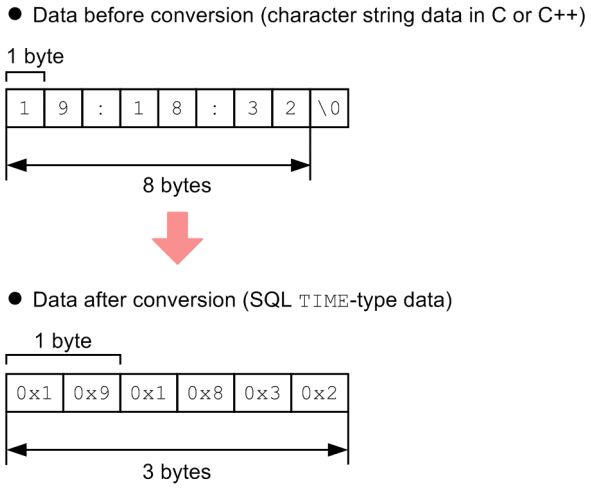
19.5.4 a_rdb_CNV_charTIME() (convert to TIME-type data)

(1) Function

This CLI function converts character string data in C or C++ to SQL TIME-type data. The following figure shows an example of conversion from character string data to TIME-type data.

Figure 19-4: Example of conversion from character string data to TIME-type data

Example: Convert character string data `19:18:32` to `TIME`-type data.



Important

The character string data before conversion must be in the predefined input representation that represents time. For details about the predefined input representation for time, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.

Example:

The character string data that represents 11:03:58.123 must be in the following format:

```
11:03:58.123
```

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charTIME
(
  char                  *char_Data,                 /* In  */
  unsigned short         char_Length,               /* In  */
  unsigned short         TIME_Scale,                /* In  */
  unsigned char         *TIME_Data,                 /* Out */
  unsigned short         BufferLength,              /* In  */
  void                  *Option                     /* In  */
)
```

(3) Arguments

`char_Data`

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

`char_Length`

Specifies the length (in bytes) of the character string data before conversion. Specify a value in the range from 8 to 8 + `TIME_Scale` value + 1.

`TIME_Scale`

Specifies the fractional seconds precision for the `TIME`-type data after conversion. Specify `0`, `3`, `6`, `9`, or `12`.

TIME_Data

Specifies the start address of the area where the TIME-type data after conversion is to be stored.

BufferLength

Specifies the length (in bytes) of the area for storing the TIME-type data after conversion. Specify 3 + ↑*value of TIME_Scale* ÷ 2↑.

Option

Specifies NULL. Any value that is specified is ignored.

(4) Return value

1. If a_rdb_CNV_charTIME() terminates normally, a_rdb_RC_CNV_SUCCESS is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must ensure that there is no overlap between the area for storing the data before conversion (char_Data) and the area for storing the data after conversion (TIME_Data).
2. The character string data before conversion is converted to TIME-type data, and then stored from the beginning of the storage area. If the fractional seconds precision for the character string data to be converted is not 3, 6, 9, or 12, zeros are padded to the right of the data until the fractional seconds precision specified for TIME_Scale is reached.

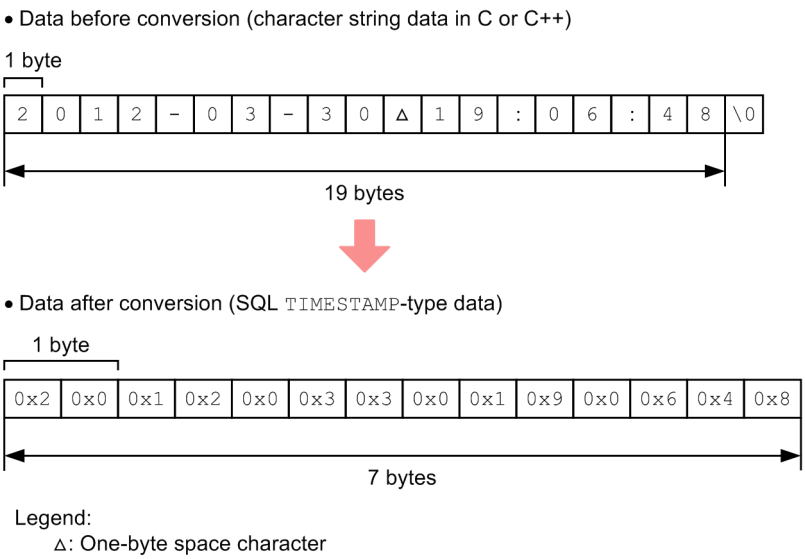
19.5.5 a_rdb_CNV_charTIMESTAMP() (convert to TIMESTAMP-type data)

(1) Function

This CLI function converts character string data in C or C++ to SQL TIMESTAMP−type data. The following figure shows an example of conversion from character string data to TIMESTAMP-type data.

Figure 19-5: Example of conversion from character string data to TIMESTAMP-type data

Example: Converting the character string 2012-03-30 19:06:48 to TIMESTAMP-type data



Important

The character string data before conversion must be in a predefined input representation that represents a time stamp. For details about the predefined input representations for time stamps, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.

Example:

The character string data that represents 19:06:48 on September 30, 2014, must be in one of the following formats:

```
2014-09-30 19:06:48
2014/09/30 19:06:48
```

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charTIMESTAMP
(
  char                  *char_Data,                 /* In  */
  unsigned short         char_Length,               /* In  */
  unsigned short         TIMESTAMP_Scale,           /* In  */
  unsigned char         *TIMESTAMP_Data,            /* Out */
  unsigned short         BufferLength,              /* In  */
  void                  *Option                     /* In  */
)
```

(3) Arguments

`char_Data`

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

`char_Length`

Specifies the length (in bytes) of the character string data before conversion. Specify a value in the range from 19 to 19 + *value of TIMESTAMP_Scale* + 1.

TIMESTAMP_Scale

Specifies the fractional seconds precision for the TIMESTAMP-type data after conversion. Specify 0, 3, 6, 9, or 12.

TIMESTAMP_Data

Specifies the start address of the area where the TIMESTAMP-type data after conversion is to be stored.

BufferLength

Specifies the length (in bytes) of the area where the TIMESTAMP-type data after conversion is to be stored. Specify 7 + ↑*value of TIMESTAMP_Scale* ÷ 2↑.

Option

Specifies NULL.

(4) Return value

1. If a_rdb_CNV_charTIMESTAMP() terminates normally, a_rdb_RC_CNV_SUCCESS is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

1. You must ensure that there is no overlap between the area for storing the data before conversion (char_Data) and the area for storing the data after conversion (TIMESTAMP_Data).
2. The character string data before conversion is converted to TIMESTAMP-type data, and then stored from the beginning of the storage area. If the fractional seconds precision for the character string data to be converted is not 3, 6, 9, or 12, zeros are padded to the right of the data until the fractional seconds precision specified for TIMESTAMP_Scale is reached.

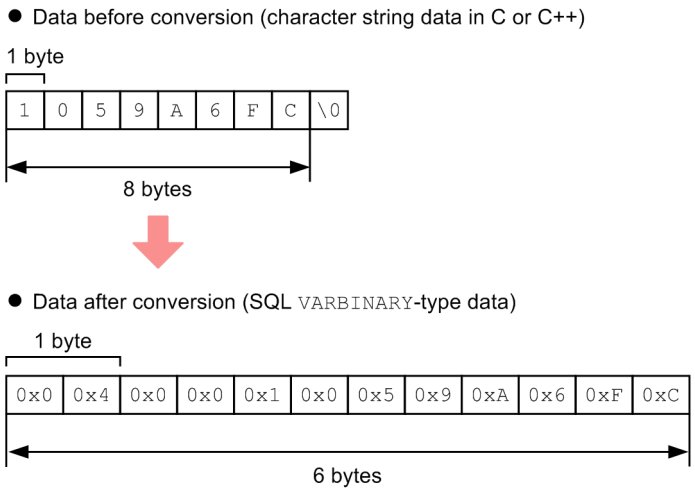
19.5.6 a_rdb_CNV_charVARBINARY() (convert to VARBINARY-type data)

(1) Function

This CLI function converts character string data in C or C++ to SQL VARBINARY-type data. The following figure shows an example of conversion from character string data to VARBINARY-type data.

Figure 19-6: Example of conversion from character string data to VARBINARY-type data

Example: Convert an 8-byte hexadecimal character string to VARBINARY-type data.



Important

The character string data before conversion must consist of one of the following values:

- When the character string data before conversion is in binary
  0 and 1
- When the character string data before conversion is in hexadecimal
  0 to 9 and A to F (or a to f)

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_charVARBINARY
(
  char                          *char_Data,           /* In  */
  unsigned int                   char_Length,         /* In  */
  unsigned short                 char_Type,           /* In  */
  unsigned short                 VARBINARY_Length,    /* In  */
  a_rdb_VARBINARY_t             *VARBINARY_Data,      /* Out */
  unsigned short                 BufferLength,        /* In  */
  void                          *Option               /* In  */
)
```

(3) Arguments

char_Data

Specifies the start address of the area where the character string data before conversion (character string data in C or C++) is stored.

char_Length

Specifies the length (in bytes) of the character string data before conversion. Specify one of the following values:

- When the character string data before conversion is in binary
  1 to *value of VARBINARY_Length* × 8
- When the character string data before conversion is in hexadecimal

1 to *value of VARBINARY_Length* × 2

`char_Type`

Specifies the format of the character string data before conversion. Specify one of the following values:

- When the character string data before conversion is in binary
  `a_rdb_CNV_CHAR_TYPE_BINARY`
- When the character string data before conversion is in hexadecimal
  `a_rdb_CNV_CHAR_TYPE_HEX`

`VARBINARY_Length`

Specifies the length (in bytes) of the `VARBINARY`-type data after conversion. The permitted values are from 1 to 32,000.

`VARBINARY_Data`

Specifies the start address of the area where the `VARBINARY`-type data after conversion is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area for storing the `VARBINARY`-type data after conversion. Specify the *value of VARBINARY_Length* + 2.

`Option`

Specifies `NULL`. Any value that is specified is ignored.

(4) Return value

1. If `a_rdb_CNV_charVARBINARY()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

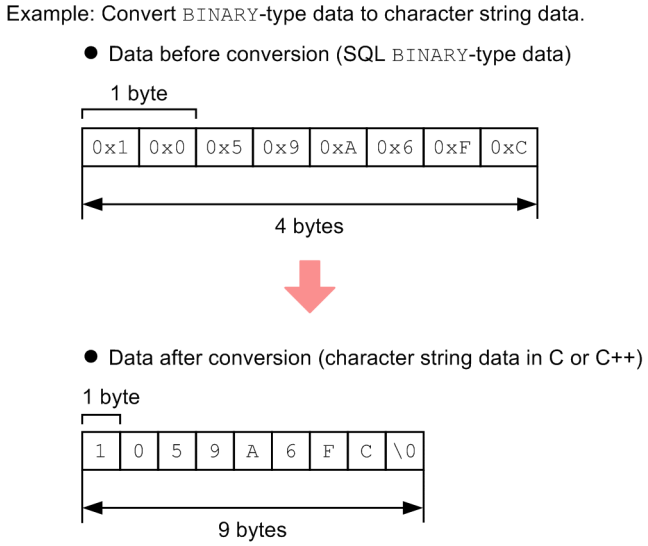
1. You must ensure that there is no overlap between the area for storing the data before conversion (`char_Data`) and the area for storing the data after conversion (`VARBINARY_Data`).
2. The length of the character string data before conversion must satisfy one of the following conditions:
   - When the character string data before conversion is in binary: Multiple of 8
   - When the character string data before conversion is in hexadecimal: Multiple of 2
3. The character string data before conversion is converted to `VARBINARY`-type data, and then stored from the beginning of the storage area.

19.5.7 a_rdb_CNV_BINARYchar() (convert BINARY-type data)

(1) Function

This CLI function converts SQL `BINARY`−type data to character string data in C or C++. The following figure shows an example of conversion from `BINARY`−type data to character string data.

Figure 19-7: Example of conversion from BINARY-type data to character string data



Explanation:

This example converts BINARY-type data to the character string literal described below and then stores the result from the beginning of the storage area. Null characters (0x00) are added at the end.

- When the format of the character string after conversion is in binary: Binary literal in binary format
- When the format of the character string after conversion is in hexadecimal: Binary literal in hexadecimal format

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_BINARYchar
(
  unsigned char                 *BINARY_Data,      /* In  */
  unsigned short                 BINARY_Length,    /* In  */
  unsigned short                 char_Type,        /* In  */
  char                          *char_Data,        /* Out */
  unsigned int                   BufferLength,     /* In  */
  void                          *Option            /* In  */
)
```

(3) Arguments

BINARY_Data

Specifies the start address of the area where the BINARY-type data before conversion is stored.

BINARY_Length

Specifies the length (in bytes) of the BINARY-type data before conversion. The permitted values are from 1 to 32,000.

char_Type

Specifies the format of the character string data after conversion. Specify one of the following values:

- When the format of the character string data after conversion is in binary
  a_rdb_CNV_CHAR_TYPE_BINARY
- When the format of the character string data after conversion is in hexadecimal

a_rdb_CNV_CHAR_TYPE_HEX

char_Data

Specifies the start address of area where the character string data after conversion (character string data in C or C++) is to be stored.

BufferLength

Specifies the length (in bytes) of the area where the character string after conversion is to be stored. Specify following value:

- When the format of the character string data after conversion is in binary
  value of BINARY_Length × 8 + 1
- When the format of the character string data after conversion is in hexadecimal
  value of BINARY_Length × 2 + 1

Option

Specifies NULL. Any value that is specified is ignored.

(4) Return value

1. If a_rdb_CNV_BINARYchar() terminates normally, a_rdb_RC_CNV_SUCCESS is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

You must ensure that there is no overlap between the area for storing the data before conversion (BINARY_Data) and the area for storing the data after conversion (char_Data).

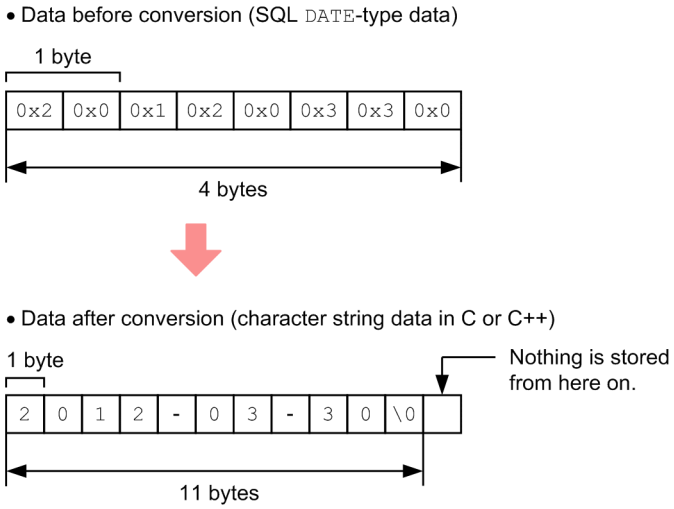
19.5.8 a_rdb_CNV_DATEchar() (convert DATE-type data)

(1) Function

This CLI function converts SQL DATE-type data to character string data in C or C++. The following figure shows an example of conversion from DATE-type data to character string data.

Figure 19-8: Example of conversion from DATE-type data to character string data

Example: Converting DATE-type data (March 30, 2012) to character string data



Explanation:

- This example converts DATE-type data to character string data according to the predefined output representation that represents a date. For details about the predefined output representation for dates, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.
- Null characters (0x00) are added at the end.

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_DATEchar
(
  unsigned char       *DATE_Data,              /* In  */
  char                *char_Data,              /* Out */
  unsigned short       BufferLength,           /* In  */
  void                *Option                  /* In  */
)
```

(3) Arguments

DATE_Data

Specifies the start address of the area where the DATE-type data before conversion is stored.

char_Data

Specifies the start address of area where the character string data after conversion (character string data in C or C++) is to be stored.

BufferLength

Specifies the length (in bytes) of the area where the character string data after conversion is to be stored. Specify a value of 11.

Option

Specifies NULL.

(4) Return value

1. If `a_rdb_CNV_DATEchar()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

You must ensure that there is no overlap between the area for storing the data before conversion (`DATE_Data`) and the area for storing the data after conversion (`char_Data`).

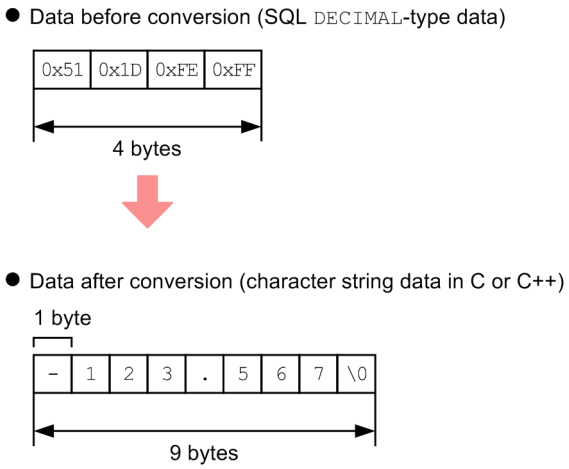
19.5.9 a_rdb_CNV_DECIMALchar() (convert DECIMAL-type data)

(1) Function

This CLI function converts SQL `DECIMAL`-type data to character string data in C or C++. The following figure shows an example of conversion from `DECIMAL`-type data to character string data.

Figure 19-9: Example of conversion from DECIMAL-type data to character string data

Example: Converting `DECIMAL(6,3)` data (`-123.567`) to character string data



Explanation:

- This example converts `DECIMAL`-type data to a character string representing an unsigned decimal literal.
- If the same values are specified for `DECIMAL_Precision` and `DECIMAL_Scale`, a zero is added before the decimal point, and then the data is converted to a character string.
- If the value of `DECIMAL_Scale` is zero, the decimal point is not added.
- If the data is a negative number, a minus sign (−) is added at the beginning.
- The value is padded with trailing null characters (`0x00`), and then the data is stored right-aligned in the storage area.
- If the stored character string is shorter than the value of `BufferLength`, the remaining area is padded with space characters.

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_DECIMALchar
(
  unsigned char          *DECIMAL_Data,             /* In  */
  unsigned short          DECIMAL_Precision,        /* In  */
  unsigned short          DECIMAL_Scale,            /* In  */
  char                   *char_Data,                /* Out */
  unsigned short          BufferLength,             /* In  */
  void                   *Option                    /* In  */
)
```

(3) Arguments

`DECIMAL_Data`

Specifies the start address of the area where the `DECIMAL`-type data before conversion is stored.

`DECIMAL_Precision`

Specifies the precision of the `DECIMAL`-type data (total number of digits) before conversion. The permitted value range is from 1 to 38.

Specify the same precision as for the data type of the fetched column's value.

`DECIMAL_Scale`

Specifies the scaling of the `DECIMAL`-type data (number of decimal places) before conversion. The permitted value range is from 0 to 38.

Specify the same scaling as for the data type of the fetched column's value.

`char_Data`

Specifies the start address of the area where the character string data after conversion (character string data in C or C++) is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area where the character string data after conversion is to be stored. Specify the following value:

value of DECIMAL_Precision + 4

`Option`

Specifies `NULL`.

(4) Return value

1. If `a_rdb_CNV_DECIMALchar()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

You must ensure that there is no overlap between the area for storing the data before conversion (`DECIMAL_Data`) and the area for storing the data after conversion (`char_Data`).

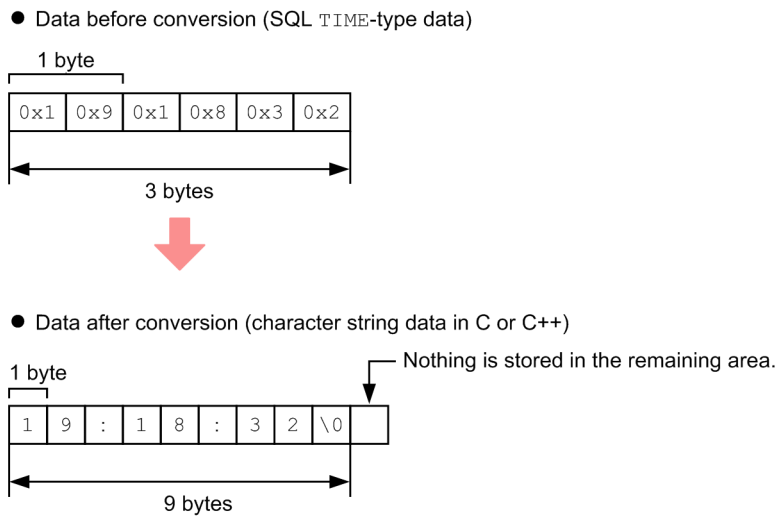
19.5.10 a_rdb_CNV_TIMEchar() (convert TIME-type data)

(1) Function

CLI function converts SQL TIME-type data to character string data in C or C++. The following figure shows an example of conversion from TIME-type data to character string data.

Figure 19-10: Example of conversion from TIME-type data to character string data

Example: Convert TIME-type data (19:18:32) to character string data.



Explanation:

- This example converts TIME-type data to character string data according to the predefined output representation that represents time. For details about the predefined output representation for time, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.
- Null characters (0x00) are added at the end.

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_TIMEchar
(
  unsigned char          *TIME_Data,              /* In  */
  unsigned short          TIME_Scale,             /* In  */
  char                   *char_Data,              /* Out */
  unsigned short          BufferLength,           /* In  */
  void                   *Option                  /* In  */
)
```

(3) Arguments

TIME_Data

Specifies the start address of the area where the TIME-type data before conversion is stored.

TIME_Scale

Specifies the fractional seconds precision for the TIME-type data before conversion. Specify 0, 3, 6, 9, or 12.

`char_Data`

Specifies the start address of the area where the character string data after conversion (character string data in C or C++) is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area where the character string after conversion is to be stored. Specify one of the following values:

- If the fractional seconds precision for the `TIME`-type data before conversion is `0`: 9
- If the fractional seconds precision for the `TIME`-type data before conversion is `3`, `6`, `9`, or `12`: 9 + `TIME_Scale` value + 1

`Option`

Specifies `NULL`. Any value that is specified is ignored.

(4) Return value

1. If `a_rdb_CNV_TIMEchar()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

You must ensure that there is no overlap between the area for storing the data before conversion (`TIME_Data`) and the area for storing the data after conversion (`char_Data`).

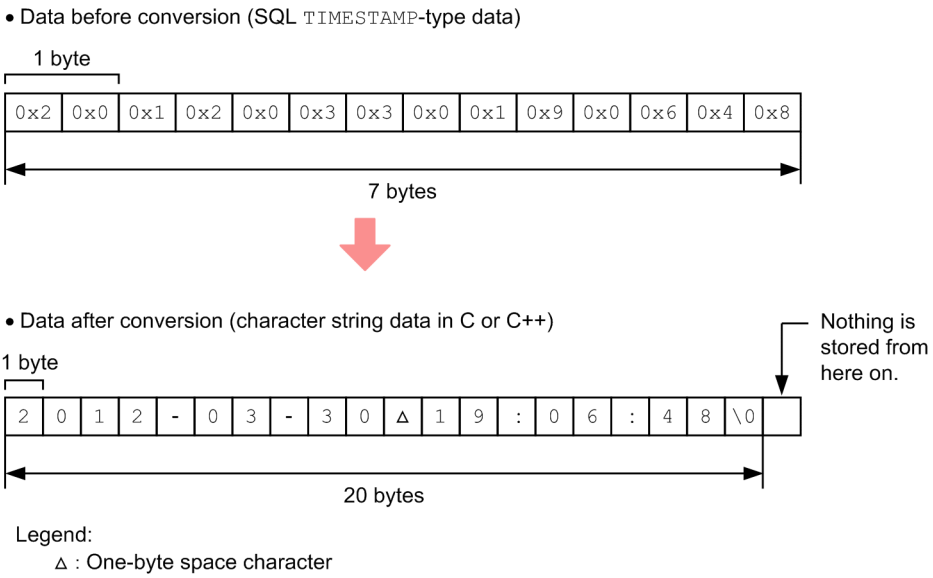
19.5.11 a_rdb_CNV_TIMESTAMPchar() (convert TIMESTAMP-type data)

(1) Function

This CLI function converts SQL `TIMESTAMP`-type data to character string data in C or C++. The following figure shows an example of conversion from `TIMESTAMP`−type data to character string data.

Figure 19-11: Example of conversion from TIMESTAMP-type data to character string data

Example: Converting TIMESTAMP-type data (19:06:48 on March 30, 2012) to character string data



Explanation:

- This example converts TIMESTAMP-type data to character string data according to the predefined output representation that represents a time stamp. For details about the predefined output representation for time stamps, see the topic *Predefined character-string representations* in the manual *HADB SQL Reference*.
- Null characters (0x00) are added at the end.

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_TIMESTAMPchar
(
  unsigned char          *TIMESTAMP_Data,          /* In  */
  unsigned short          TIMESTAMP_Scale,         /* In  */
  char                   *char_Data,               /* Out */
  unsigned short          BufferLength,            /* In  */
  void                   *Option                   /* In  */
)
```

(3) Arguments

TIMESTAMP_Data

Specifies the start address of the area where the TIMESTAMP-type data before conversion is stored.

TIMESTAMP_Scale

Specifies the fractional seconds precision for the TIMESTAMP-type data before conversion. Specify 0, 3, 6, 9, or 12.

char_Data

Specifies the start address of the area where the character string data after conversion (character string data in C or C++) is to be stored.

BufferLength

Specifies the length (in bytes) of the area where the character string after conversion is to be stored. Specify one of the following values:

- If the fractional seconds precision for the TIMESTAMP-type data to be converted is 0: 20
- If the fractional seconds precision for the TIMESTAMP-type data to be converted is 3, 6, 9, or 12: 20 + TIMESTAMP_Scale value + 1

Option

Specifies NULL.

(4) Return value

1. If a_rdb_CNV_TIMESTAMPchar() terminates normally, a_rdb_RC_CNV_SUCCESS is returned.
2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

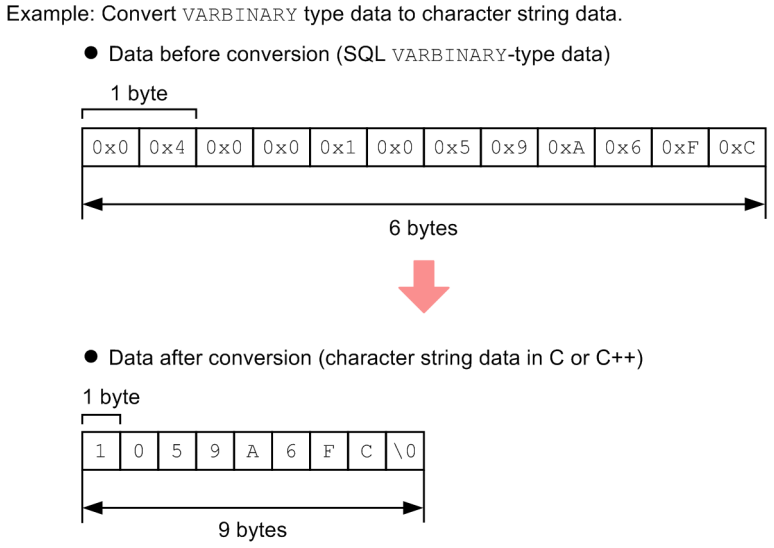
You must ensure that there is no overlap between the area for storing the data before conversion (TIMESTAMP_Data) and the area for storing the data after conversion (char_Data).

19.5.12 a_rdb_CNV_VARBINARYchar() (convert VARBINARY-type data)

(1) Function

This CLI function converts SQL VARBINARY-type data to character string data in C or C++. The following figure shows an example of conversion from VARBINARY−type data to character string data.

Figure 19-12: Example of conversion from VARBINARY-type data to character string data



Explanation:

This example converts VARBINARY-type data to the character string literal described below and then stores the result from the beginning of the storage area. Null characters (0x00) are added at the end of the data.

- When the format of the character string data after conversion is in binary: Binary literal in binary format
- When the format of the character string data after conversion is in hexadecimal: Binary literal in hexadecimal format

(2) Format

```
#include <adbcnv.h>

signed short a_rdb_CNV_VARBINARYchar
(
  a_rdb_VARBINARY_t           *VARBINARY_Data,       /* In  */
  unsigned short               VARBINARY_Length,     /* In  */
  unsigned short               char_Type,            /* In  */
  char                        *char_Data,            /* Out */
  unsigned int                 BufferLength,         /* In  */
  void                        *Option                /* In  */
)
```

(3) Arguments

`VARBINARY_Data`

Specifies the start address of the area where the `VARBINARY`-type data before conversion is stored.

`VARBINARY_Length`

Specifies the length (in bytes) of the `VARBINARY`-type data before conversion. The permitted values are from 1 to 32,000.

`char_Type`

Specifies the format of the character string data after conversion. Specify one of the following values:

- When the format of the character string data after conversion is in binary
  `a_rdb_CNV_CHAR_TYPE_BINARY`
- When the format of the character string data after conversion is in hexadecimal
  `a_rdb_CNV_CHAR_TYPE_HEX`

`char_Data`

Specifies the start address of the area where the character string data after conversion (character string data in C or C++) is to be stored.

`BufferLength`

Specifies the length (in bytes) of the area where the character string after conversion is to be stored. Specify one of the following values:

- When the format of the character string data after conversion is in binary
  value of VARBINARY_Length × 8 + 1
- When the format of the character string data after conversion is in hexadecimal
  value of VARBINARY_Length × 2 + 1

`Option`

Specifies `NULL`. Any value that is specified is ignored.

(4) Return value

1. If `a_rdb_CNV_VARBINARYchar()` terminates normally, `a_rdb_RC_CNV_SUCCESS` is returned.

2. If an error occurs, the error code is returned as the return value. For details about the error code, see 19.8 Return values of the CLI functions.

(5) Notes

You must ensure that there is no overlap between the area for storing the data before conversion (`VARBINARY_Data`) and the area for storing the data after conversion (`char_Data`).

19.6 Correspondence to the SQL data types

This section explains the SQL data types and the corresponding symbolic literals and data types.

19.6.1 Correspondences among SQL data types, symbolic literals, and values

The following table shows the correspondences among the SQL data types, the symbolic literals, and the values.

Table 19-7: Correspondences among the SQL data types, the symbolic literals, and the values

No.	SQL data type	Symbolic literal	Value
1	CHAR	a_rdb_SQL_DT_CHAR	0xC5
2	VARCHAR	a_rdb_SQL_DT_VARCHAR	0xC1
3	INTEGER	a_rdb_SQL_DT_INT	0xF1
4	SMALLINT	a_rdb_SQL_DT_SMALLINT	0xF5
5	DECIMAL	a_rdb_SQL_DT_DEC	0xE5
6	DOUBLE PRECISION	a_rdb_SQL_DT_DOUBLE	0xE1
7	DATE	a_rdb_SQL_DT_DATE	0x71
8	TIME	a_rdb_SQL_DT_TIME	0x79
9	TIMESTAMP	a_rdb_SQL_DT_TIMESTAMP	0x7D
10	BINARY	a_rdb_SQL_DT_BINARY	0x95
11	VARBINARY	a_rdb_SQL_DT_VARBINARY	0x91
12	ROW	a_rdb_SQL_DT_ROW	0x45

19.6.2 Correspondences between SQL data types and data descriptions

The following table shows the correspondences between the SQL data types and the data descriptions in C or C++.

Table 19-8: Correspondences between the SQL data types and the data descriptions in C or C++.

No.	SQL data type	Data description in C or C++	Area length (bytes)
1	CHAR(*n*)	char *variable-name* [*n* + 1];	*n* + 1
2	VARCHAR(*n*)	a_rdb_M_VARCHAR(*n*) *variable-name*;	*n* + 4
3	INTEGER	long long *variable-name*;	8
4	SMALLINT	int *variable-name*;	4
5	DECIMAL(*m*,*n*)	unsigned char *variable-name*[*p*#1];	*p*#1
6	DOUBLE PRECISION	double *variable-name*;	8
7	DATE	unsigned char *variable-name*[4];	4
8	TIME(*p*) #2	unsigned char *variable-name*[3 + (*p* + 1) ÷ 2];	3 + (*p* + 1) ÷ 2

No.	SQL data type	Data description in C or C++	Area length (bytes)
9	TIMESTAMP(*p*) #2	unsigned char *variable-name*[7 + (*p* + 1) ÷ 2];	7 + (*p* + 1) ÷ 2
10	BINARY(*n*)	unsigned char *variable-name*[*n*];	*n*
11	VARBINARY(*n*)	a_rdb_M_VARBINARY(*n*) *variable-name*;	*n* + 2
12	ROW	unsigned char *variable-name*[*row-length*#3];	*row-length*#3

Legend:

m, *n*: Positive integer

#1

The value of *p* depends on the value of *m* (precision).

No.	Value of m	Value of p
1	$1 \leq m \leq 4$	2
2	$5 \leq m \leq 8$	4
3	$9 \leq m \leq 16$	8
4	$17 \leq m \leq 38$	16

#2

p indicates the fractional seconds precision and its value is 0, 3, 6, 9, or 12.

#3

The row length is the sum of the data lengths of all the columns. For details about determining the data length of a column, see *Length of data storage* in the topic *List of data types* in the manual *HADB SQL Reference*.

Macros used in the data descriptions are expanded as shown below.

■ a_rdb_M_VARCHAR(*n*) *variable-name*;

```
struct
{
  unsigned int Length ;           /* Data length */
  char Data[n] ;                  /* Character string data */
}
```

■ a_rdb_M_VARBINARY(*n*) *variable-name*;

```
struct
{
  unsigned short Length ;         /* Data length */
  unsigned char Data[n] ;         /* Binary data */
}
```

19.6.3 Correspondence to the VARCHAR type

(1) Function

a_rdb_VARCHAR_t corresponds to the SQL data type VARCHAR.

(2) Format

```
typedef struct a_rdb_TG_VARCHAR {
  unsigned int     Length ;
  char             Data[1] ;
} a_rdb_VARCHAR_t ;
```

(3) Members

`Length`

Specifies the length (in bytes) of the variable-length character string.

`Data`

Specifies the variable-length character string.

19.6.4 VARBINARY type

(1) Function

`a_rdb_VARBINARY_t` corresponds to the SQL data type `VARBINARY`.

(2) Format

```
typedef struct a_rdb_TG_VARBINARY {
  unsigned short   Length ;
  unsigned char   Data[1] ;
} a_rdb_VARBINARY_t ;
```

(3) Members

`Length`

Specifies the length (in bytes) of the variable-length binary data.

`Data`

Specifies the variable-length binary data.

19.7 Data types used in the CLI functions

This section explains the data types used in the CLI functions.

19.7.1 a_rdb_SQLColumnInfo_t structure (column information)

(1) Function

Information about the retrieval result columns acquired by `a_rdb_SQLDescribeCols()` is stored in `a_rdb_SQLColumnInfo_t` structures.

These structures are also used when `a_rdb_SQLBindCols()` is used to associate the retrieval result columns.

For details about `a_rdb_SQLDescribeCols()`, see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns). For details about `a_rdb_SQLBindCols()`, see 19.4.3 a_rdb_SQLBindCols() (associate retrieval result columns).

(2) Format

```
typedef struct a_rdb_TG_ColumnInfo {
  struct a_rdb_TG_NameInfo    *NameInfo ;              /* Out */
  struct a_rdb_TG_DataType    *TypeInfo ;              /* Out */
  unsigned short               Nullable ;              /* Out */
  char                         _ColumnInfo_rsv01[2] ;
  signed int                   StrLen_or_Ind ;         /* Out */
  void                        *TargetValue ;           /* Out */
  unsigned int                 BufferLength ;          /* In */
  char                         _ColumnInfo_rsv02[28] ;
} a_rdb_SQLColumnInfo_t ;
```

(3) Members

`NameInfo`

Specifies the address at which information about a column name of a retrieval result column is to be stored.

Specify this member if you execute a column description. If you execute a column bind, this member is ignored, if specified.

If a value of `0` is specified, this information is not acquired.

`TypeInfo`

Specifies the address at which information about the data type of a retrieval result column is to be stored.

Specify this member if you execute a column description. If you execute a column bind, this member is ignored, if specified.

If a value of `0` is specified, this information is not acquired.

`Nullable`

Returns a value indicating whether the null value can be returned in the retrieval result column.

- The null value cannot be returned: `a_rdb_SQL_IS_NOT_NULLABLE`
- The null value can be returned: `a_rdb_SQL_IS_NULLABLE`

This value is set when a column description is executed.

`StrLen_or_Ind`

Acquires either a value length or an indicator value.

This value is set when a column description is executed.

`TargetValue`

Specifies the address at which the value is to be stored. This value must be specified as a data description in C or C ++ that corresponds to the SQL data type.

For details about the data descriptions in C or C++ that correspond to the SQL data types, see 19.6.2 Correspondences between SQL data types and data descriptions.

Specify this member if you execute a column bind. If you execute a column description, this member is ignored, if specified.

`BufferLength`

Specifies the length of the area in which the value is to be stored.

For details about the data area length, see 19.6.2 Correspondences between SQL data types and data descriptions.

Specify this member if you execute a column bind. If you execute a column description, this member is ignored, if specified.

(4) Notes

When you use a `a_rdb_SQLColumnInfo_t` structure, make sure that you first initialize the area to all zeros, and then specify a value for each member to be used.

19.7.2 a_rdb_SQLNameInfo_t structure (name information)

(1) Function

Name information, such as the column names of the retrieval result columns acquired by `a_rdb_SQLDescribeCols()`, is stored in `a_rdb_SQLNameInfo_t` structures. For details about `a_rdb_SQLDescribeCols()`, see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns).

(2) Format

```
typedef struct a_rdb_TG_NameInfo {
  unsigned short   NameLength ;                  /* Out */
  unsigned short   BufferLength ;                /* In * /
  char             _NameInfo_rsv01[4] ;
  char            *Name ;                        /* Out */
} a_rdb_SQLNameInfo_t ;
```

(3) Members

`NameLength`

Acquires the length of a name (in bytes).

`BufferLength`

Specifies the size (in bytes) of the area in which the name is to be acquired.

Name

Specifies the address of the area in which the name is to be stored.

(4) Notes

When you use a a_rdb_SQLNameInfo_t structure, make sure that you first initialize the area to zeros and then specify a value for each member to be used.

19.7.3 a_rdb_SQLDataType_t structure (data type information)

(1) Function

The following information is stored in a_rdb_SQLDataType_t structures:

- Information about the data types of the retrieval result columns acquired by a_rdb_SQLDescribeCols()
- Information about the data types of dynamic parameters acquired by a_rdb_SQLDescribeParams()

For details about a_rdb_SQLDescribeCols(), see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns). For details about a_rdb_SQLDescribeParams(), see 19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information).

(2) Format

```
typedef struct a_rdb_TG_DataType {
  signed int            DataType ;                /* Out */
  unsigned short        ElementCount ;            /* Out */
  char                  _DataType_rsv01[2] ;
  unsigned int          DataLength1 ;             /* Out */
  unsigned int          DataLength2 ;             /* Out */
  char                  _DataType_rsv02[16] ;
} a_rdb_SQLDataType_t ;
```

(3) Members

DataType

Stores a data type code.

ElementCount

Stores information about the data structure.

A value of 1 is stored.

DataLength1

If a_rdb_SQLDescribeCols() is executed, the column length is acquired. If a_rdb_SQLDescribeParams() is executed, the parameter length is acquired.

For details about the value to be specified, see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns) or 19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information).

DataLength2

If a_rdb_SQLDescribeCols() is executed, the column length attribute is acquired. If a_rdb_SQLDescribeParams() is executed, the parameter length attribute is acquired.

For details, see 19.4.6 a_rdb_SQLDescribeCols() (acquire information about the retrieval result columns) and 19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information).

(4) Notes

When you use an `a_rdb_SQLDataType_t` structure, make sure that you first initialize the area to all zeros, and then specify a value for each member to be used.

19.7.4 a_rdb_SQLInd_t (indicator)

(1) Function

`a_rdb_SQLInd_t` (indicator) indicates whether a value is the null value.

(2) Format

```
typedef signed char a_rdb_SQLInd_t ;
```

(3) Indicator value

The following table shows the symbolic literals and values for the indicator.

Table 19-9: Symbolic literals and values of the indicator

No.	Description	Symbolic literal	Value
1	Null value	`a_rdb_SQL_NULL_DATA`	`-1`
2	Non-null value data	`a_rdb_SQL_NOT_NULL_DATA`	`0`

19.7.5 a_rdb_SQLParameterInfo_t structure (parameter information)

(1) Function

The dynamic parameter information acquired by `a_rdb_SQLDescribeParams()` is stored in `a_rdb_SQLParameterInfo_t` structures.

These structures are also used when an area for specifying dynamic parameter values is associated by `a_rdb_SQLBindParams()` or `a_rdb_SQLBindArrayParams()`.

For details about `a_rdb_SQLDescribeParams()`, see 19.4.7 a_rdb_SQLDescribeParams() (acquire dynamic parameter information). For details about `a_rdb_SQLBindParams()`, see 19.4.4 a_rdb_SQLBindParams() (associate dynamic parameters). For details about `a_rdb_SQLBindArrayParams()`, see 19.4.2 a_rdb_SQLBindArrayParams() (bind dynamic parameters in batch mode).

(2) Format

```
typedef struct a_rdb_TG_ParameterInfo {
  struct a_rdb_TG_DataType     *TypeInfo ;        /* Out */
```

```
  unsigned short                Nullable ;       /* Out */
  char                          _ParameterInfo_rsv01[2] ;
  a_rdb_SQLInd_t                Ind ;            /* In  */
  char                          _ParameterInfo_rsv02[3] ;
  void                         *ParameterValue;  /* In  */
  char                          _ParameterInfo_rsv03[24] ;
} a_rdb_SQLParameterInfo_t ;
```

(3) Members

`TypeInfo`

Specifies the address at which information about the data type of a dynamic parameter is to be stored.

Specify this member if you execute a parameter description. If you execute a parameter join, there is no need to specify this member; if specified, this member is ignored.

`Nullable`

Returns a value indicating whether the null value can be specified for the dynamic parameter.

- The null value cannot be specified: `a_rdb_SQL_IS_NOT_NULLABLE`
- The null value can be specified: `a_rdb_SQL_IS_NULLABLE`

This value is set for a parameter description.

`Ind`

Specifies an indicator value.

Specify this member if you execute a parameter join. If you execute a parameter description, there is no need to specify this member. If specified, this member is ignored.

`ParameterValue`

Specifies the address at which the value is to be stored.

This value must be specified as a data description in C or C++ that corresponds to the SQL data type. For details about the data descriptions in C or C++ that correspond to the SQL data types, see 19.6.2 Correspondences between SQL data types and data descriptions.

Specify this member if you execute a parameter join. If you execute a parameter description, there is no need to specify this member. If specified, this member is ignored.

(4) Notes

When you use a `a_rdb_SQLParameterInfo_t` structure, make sure that you first initialize the area to all zeros, and then specify a value for each member to be used.

19.7.6 a_rdb_SQLResultInfo_t structure (SQL results information)

(1) Function

SQL results information is stored in an `a_rdb_SQLResultInfo_t` structure for each CLI function call.

(2) Format

```
typedef struct a_rdb_TG_ResultInfo {
  unsigned long long  RowCount ;                /* Out */
  unsigned char       isInConnect ;             /* Out */
```

```
  unsigned char       EndTran ;                   /* Out */
  unsigned char       RowCountOverFlowed ;        /* Out */
  unsigned char       isMessageLogFull ;          /* Out */
  unsigned char       isClientLogFull ;           /* Out */
  char                _ResultInfo_rsv01[3] ;
  char                SQLState[5] ;               /* Out */
  char                _ResultInfo_rsv02[11] ;
} a_rdb_SQLResultInfo_t ;
```

(3) Members

`RowCount`

Returns the number of results rows when the CLI function's return value is `a_rdb_RC_SQL_SUCCESS` or `a_rdb_RC_SQL_NO_DATA`. Otherwise, `RowCount` returns a value `0`.

The number of rows that have been processed is returned, in the range from 0 to 18,446,744,073,709,551,615.

If overflow occurs (if the returned value exceeds 18,446,744,073,709,551,615), 18,446,744,073,709,551,615 is returned and `a_rdb_SQL_OVERFLOWED` is set in `RowCountOverFlowed`.

`isInConnect`

Returns a value indicating whether a connection has been established (currently connected):

- Connection has not been established: `a_rdb_SQL_IS_NOT_IN_CONNECT`
- Connection has been established: `a_rdb_SQL_IS_IN_CONNECT`

`EndTran`

Returns the transaction execution result:

- Transaction terminated normally: `a_rdb_SQL_COMMITTED`
- Transaction was canceled: `a_rdb_SQL_ROLLBACKED`
- Transaction is executing: `a_rdb_SQL_TRAN_NOT_ENDED`
- Transaction has not started yet: `a_rdb_SQL_TRAN_NOT_STARTED`
- Transaction was canceled because an SQL error occurred when the transaction terminated normally: `a_rdb_SQL_DETECTED_ERROR_IN_COMMIT`

The following table shows the correspondences between transaction execution results and the transaction statuses.

Table 19-10: Correspondences between transaction execution results and transaction statuses

No.	Transaction execution result	Transaction status
1	`a_rdb_SQL_COMMITTED`	Transaction startable#
2	`a_rdb_SQL_ROLLBACKED`	Transaction startable#
3	`a_rdb_SQL_TRAN_NOT_ENDED`	Transaction executing
4	`a_rdb_SQL_TRAN_NOT_STARTED`	Transaction startable
5	`a_rdb_SQL_DETECTED_ERROR_IN_COMMIT`	Transaction startable#

#

The transaction was completed because `COMMIT` or `ROLLBACK` was successful, and the next transaction can now be started.

`RowCountOverFlowed`

Returns a value indicating whether overflow occurred on `RowCount`:

- Overflow has not occurred: a_rdb_SQL_NOT_OVERFLOWED
- Overflow has occurred: a_rdb_SQL_OVERFLOWED

isMessageLogFull

Returns the mode of the server message log file.

- Normal mode: a_rdb_SQL_IS_NOT_LOG_FULL
- Fall-back mode: a_rdb_SQL_IS_LOG_FULL

If the server message log file is placed in fall-back mode, contact the HADB administrator.

isClientLogFull

Returns the mode of the client message log file.

- Normal mode: a_rdb_SQL_IS_NOT_LOG_FULL
- Fall-back mode: a_rdb_SQL_IS_LOG_FULL

If the client message log file is placed in fall-back mode, secure sufficient free space on the disk that contains the client message log file. At least twice as much free space as the size specified for the ADBMSGLOGSIZE environment variable is required.

SQLState

Returns SQLSTATE of the CLI function execution results.

For details about SQLSTATE, see *List of SQLSTATE values* in the manual *HADB Messages*.

19.8 Return values of the CLI functions

The table below shows the return values of the following types of CLI functions:

- CLI functions for connecting to and disconnecting from the HADB server
- CLI functions for controlling transactions
- CLI functions for execution of SQL statements

Table 19-11: Return values of the CLI functions

No.	Event	Symbolic literal	Value
1	CLI function terminated normally.	a_rdb_RC_SQL_SUCCESS	0
2	CLI function terminated with a warning.	a_rdb_RC_SQL_WARNING	+1
3	There is no data.	a_rdb_RC_SQL_NO_DATA	+100
4	An SQL error occurred.	--	SQLCODE
5	An error occurred that disabled output of messages to the client message log file.	--	*error-cause-code*#
6	Other error occurred.	a_rdb_RC_SQL_ERROR	-1

Legend:

--: Not applicable.

#

The following table lists and describes the error cause codes.

Table 19-12: List of error cause codes

No.	Error cause code	Cause of the error
1	-10000	The specified ConnectionHandle argument is invalid (NULL was specified).
2	-11000	Acquisition of the absolute path of the message catalog file of the client message log file failed.
3	-12*XXX*	open processing on the message catalog file of the client message log file failed. errno (error number) is set in *XXX*.
4	-13000	Acquisition of the ADBMSGLOGSIZE environment variable (size of the client message log file) failed.
5	-14000	Acquisition of the absolute path of the client message log file failed.
6	-15*XXX*	open processing on the client message log file failed. errno (error number) is set in *XXX*.
7	-16*XXX*	fstat processing on the client message log file failed. errno (error number) is set in *XXX*.
8	-17*XXX*	read processing on the client message log file failed. errno (error number) is set in *XXX*.
9	-18*XXX*	lseek processing on the client message log file failed. errno (error number) is set in *XXX*.
10	-19*XXX*	write processing on the client message log file failed. errno (error number) is set in *XXX*.
11	-21000	The versions of the HADB client and ODBC driver do not match.

The following table lists and describes the return values of the CLI functions for data type conversion.

Table 19-13: Return values of CLI functions for data type conversion

No.	Event	Symbolic literal	Value	Action
1	The CLI function terminated normally.	a_rdb_RC_CNV_SUCCESS	0	None
2	The address of the area storing the data before conversion is not valid.	a_rdb_RC_CNV_INVALID_SRC_ADDRESS	1	Check the address of the area storing the data before conversion.
3	The length of the data to be converted is not valid.	a_rdb_RC_CNV_INVALID_SRC_LENGTH	2	Check the length of the data to be converted for any error.
4	The data before conversion contains data that cannot be converted.	a_rdb_RC_CNV_INVALID_FORMAT	3	Check the data to be converted.
5	The data type after conversion is not valid.	a_rdb_RC_CNV_INVALID_DEST_LENGTH	4	For details about the SQL data types, see the topic *Data types* in the manual *HADB SQL Reference*. Then check the following specification for any error: • DECIMAL_Precision or DECIMAL_Scale argument in a_rdb_CNV_charDECIMAL() • TIME_Scale argument in a_rdb_CNV_charTIME() • TIMESTAMP_Scale argument in a_rdb_CNV_charTIMESTAMP() • BINARY_Length argument in a_rdb_CNV_charBINARY() • VARBINARY_Length argument in a_rdb_CNV_charVARBINARY()
6	The address of the area for storing the data after conversion is not valid.	a_rdb_RC_CNV_INVALID_DEST_ADDRESS	5	Check the address of the area for storing the data after conversion.
7	The length of the area for storing the data after conversion is not valid.	a_rdb_RC_CNV_INVALID_BUF_LENGTH	6	Check the value of the argument (BufferLength) that specifies the length of the area for storing the converted data.
8	The area storing the data before conversion overlaps the area for storing the data after conversion.	a_rdb_RC_CNV_OVERRAPPING_AREA	7	Check the argument values to make sure that the specified areas do not overlap.
9	The specified character string type is invalid.	a_rdb_RC_CNV_INVALID_CHAR_TYPE	8	Specify the correct character string type.

Appendixes

A. Sample Application Program

HADB provides a sample application program that connects to and disconnects from the HADB server, and performs row retrieval, addition, and deletion processing.

A.1 Overview of sample application program

The following table describes the sample application program provided by HADB.

Table A-1: Sample application program provided by HADB

No.	Name of sample application program	Programming language of sample application program	File name of sample application program	Operations performed by sample application program
1	sample1	Java	Sample1.java	• Table retrieval (SELECT) • Row addition (INSERT) • Row deletion (DELETE)
2		ODBC functions	odbc_sample1.c	
3		C or C++	cli_sample1.c	

Note

The sample application program is located under the following directory on the HADB server:

- $ADBDIR/sample

A.2 Preparations before executing the sample application program

Before you execute the sample application program, you must perform the following preparations:

- Create a make environment for the sample application program. Specify the include files and libraries provided by HADB and then execute make.
- An HADB client environment setup must be completed. For details about HADB client environment setup, see 4. Setting Up an Environment for an HADB Client (If the ODBC Driver and CLI Functions Are Used).
- Define the SAMPLE table that is accessed by the sample application program and then import data into the SAMPLE table. For details, see A.3 How to create the SAMPLE table.

A.3 How to create the SAMPLE table

Before you create the SAMPLE table, you must create the data DB areas (ADBUTBL01 and ADBUIDX01) where the SAMPLE table will be stored. The SAMPLE table cannot be created using the method explained here until you have created ADBUTBL01 and ADBUIDX01.

To create the SAMPLE table:

Procedure

1. Create a user named ADBUSER02. Set the password for this ADBUSER02 user to #HelloHADB_02.
   SQL statement to be executed:

```
CREATE USER "ADBUSER02" IDENTIFIED BY '#HelloHADB_02'
```

2. Grant the CONNECT privilege and the schema definition privilege to the ADBUSER02 user.
   SQL statement to be executed:

```
GRANT CONNECT, SCHEMA TO "ADBUSER02"
```

3. To create the SAMPLE table, execute $ADBDIR/sample/create_sampledb.sh, which is a shell script that creates the SAMPLE table.
   When you execute this shell script, the SAMPLE table is defined, and then data is imported into the SAMPLE table.

The following shows the schema, table, and B-tree index definitions for the SAMPLE table that is created:

```
CREATE SCHEMA "ADBUSER02"

CREATE TABLE "SAMPLE"(
    "STATECODE"       SMALLINT,
    "STATENAME"       VARCHAR(15),
    "ZIPCODE"         CHAR(15),
    "ADDRESS"         VARCHAR(100),
    "AREA"            DECIMAL(19))
  IN ADBUTBL01

CREATE INDEX "CODE_IDX" ON "SAMPLE"("STATECODE" ASC)
  IN ADBUIDX01 EMPTY
```

■ SAMPLE table format

STATECODE	STATENAME	ZIPCODE	ADDRESS	AREA
1	Alabama	36130-2751	State Capitol N-104 600 Dexter Avenue Montgomery	135,765,000,000
2	Alaska	99811	State Capitol Juneau	1,717,854,000,000
3	Arizona	85007	State Capitol West Wing 1700 W. Washington, 9th Fl. Phoenix	295,254,000,000
:	:	:	:	:
49	Wisconsin	53707-7863	State Capitol P.O. Box 7863 Madison	169,639,000,000
50	Wyoming	82002-0010	State Capitol Cheyenne	253,336,000,000

Explanation:

STATECODE: State code

STATENAME: State name

ZIPCODE: Zip code

ADDRESS: Address of state capitol

AREA: Area of state in square meters

The files related to the sample application program are as follows:

- $ADBDIR/sample/Sample1.java: Sample application program that uses a JDBC driver

- $ADBDIR/sample/odbc_sample1.c: Sample application program that uses an ODBC driver
- $ADBDIR/sample/cli_sample1.c: Sample application program that uses CLI functions
- $ADBDIR/sample/create_sampledb.sh: Shell script for creating the SAMPLE table
- $ADBDIR/sample/SAMPLE.txt: Data to be stored in the SAMPLE table
- $ADBDIR/sample/SAMPLE_table.sql: File containing the definition SQL statements for defining the SAMPLE table

A.4 Sample application program execution procedure

The following explains the procedure for executing the sample application program (sample1).

To execute the sample program:

1. Start sample1's executable file.
2. The message shown below is displayed. Choose the SQL statement to be executed (1, 2, or 3). The subsections below explain the operating method for each of these SQL statements.

```
***** HADB CLI Function Sample Program *****
1. Search (specify search range)
2. Add
3. Delete
4. Exit
Please specify a menu item (1 - 4):
```

(1) SELECT statement is selected (1 is selected)

Retrieval processing specifying a US state code (STATECODE) is performed. The following SQL statement is executed:

```
SELECT "STATECODE","STATENAME","ZIPCODE","ADDRESS","AREA"
    FROM "SAMPLE"
    WHERE "STATECODE" BETWEEN ? AND ?
```

When the SELECT statement is selected, the following messages are output to the standard output:

```
• Please specify the minimum number of search conditions (1 - 50): (waits for an entr
y)
    ← Specify a value for the first dynamic parameter (minimum value).
• Please specify the maximum number of search conditions (1 - 50): (waits for an entr
y)
    ← Specify a value for the second dynamic parameter (maximum value).
```

The SELECT statement is executed based on the values you have entered.

> **Note**
>
> If the above two entries are both null (only the **Enter** key is pressed), the SELECT statement is executed with 1 assumed for the first dynamic parameter and 50 for the second dynamic parameter.

(2) INSERT statement is selected (2 is selected)

Rows are added to the SAMPLE table. The following SQL statement is executed:

```
INSERT INTO "SAMPLE" VALUES (?,?,?,?,?)
```

When the INSERT statement is selected, the following messages are output to the standard output:

```
• Please specify a state code : (waits for an entry)
    ← Specify a value for the first dynamic parameter (state code).
• Please specify a state name : (waits for an entry)
    ← Specify a value for the second dynamic parameter (state name).
• Please specify a zip code : (waits for an entry)
    ← Specify a value for the third dynamic parameter (zip code).
• Please specify an address : (waits for an entry)
    ← Specify a value for the fourth dynamic parameter (state capitol).
• Please specify an area : (waits for an entry)
    ← Specify a value for the fifth dynamic parameter (area).
```

The INSERT statement is executed based on the values you have entered.

If an invalid value is specified, an SQL error occurs.

(3) DELETE statement is selected (3 is selected)

Rows are deleted from the SAMPLE table. The following SQL statement is executed:

```
DELETE FROM "SAMPLE" WHERE "STATECODE" = ?
```

When the DELETE statement is selected, the following message is output to the standard output:

```
• Please specify the state code of the row to be deleted : (waits for an entry)
    ← Specify a value for the dynamic parameter (state code of the row to be deleted)
.
```

The DELETE statement is executed based on the value you have entered.

If an invalid value is specified, an SQL error occurs.

B. Structure of HADB Client Directories

This appendix describes the structures of the client directories for HADB clients (at installation and during operation).

B.1 HADB clients for Windows

(1) Structure of the client directory (at installation)

The tables below describe the structure of the client directory (at installation) when a Windows edition of HADB client is installed.

In the tables, `%INSTCLTDIR%` indicates the client directory (at installation).

Table B-1: Structure of the client directory (at installation) (64-bit edition of Windows)

No.	Folder and file names	Description
1	`%INSTCLTDIR%\client`	Folder for storing the commands, libraries, ODBC driver, and ODBC data source setup program used on the HADB client
2	`%INSTCLTDIR%\client\bin`	Folder for storing the commands, ODBC driver, and ODBC data source setup program used on the HADB client
3	`%INSTCLTDIR%\client\bin\adbclt.dll`	HADB client
4	`%INSTCLTDIR%\client\bin\adbodbc.dll`	ODBC driver
5	`%INSTCLTDIR%\client\bin\adbodbcstp.dll`	ODBC data source setup program
6	`%INSTCLTDIR%\client\bin\adbclt32.dll`	HADB client (32-bit)
7	`%INSTCLTDIR%\client\bin\adbodbc32.dll`	ODBC driver (32-bit)
8	`%INSTCLTDIR%\client\bin\adbodbstp32.dll`	ODBC data source setup program (32-bit)
9	`%INSTCLTDIR%\client\lib`	Folder for storing libraries used with the HADB client
10	`%INSTCLTDIR%\client\lib\adbclt.lib`	Client library
11	`%INSTCLTDIR%\client\lib\adbclt32.lib`	Client library (32-bit)
12	`%INSTCLTDIR%\client\lib\adbjdbc8.jar`	JDBC driver (JRE 8)
13	`%INSTCLTDIR%\conf`	Folder for storing client definitions
14	`%INSTCLTDIR%\include`	Folder for storing user-provided header files
15	`%INSTCLTDIR%\include\adbcnv.h`	User-provided header files
16	`%INSTCLTDIR%\include\adbcli.h`	
17	`%INSTCLTDIR%\include\adbtypes.h`	
18	`%INSTCLTDIR%\include\adbodb.h`	
19	`%INSTCLTDIR%\lib`	Folder for storing libraries
20	`%INSTCLTDIR%\lib\sysdef`	Folder for storing definition analysis information files
21	`%INSTCLTDIR%\lib\sysdef\adbclt.def`	Definition analysis information file
22	`%INSTCLTDIR%\sample`	Folder for storing a sample application program and model definition files

No.	Folder and file names	Description
23	%INSTCLTDIR%\sample\cli_sample1.c	Sample application program
24	%INSTCLTDIR%\sample\Sample1.java	Sample application program (for JDBC)
25	%INSTCLTDIR%\sample\odbc_sample1.c	Sample application program (for ODBC)
26	%INSTCLTDIR%\sample\conf	Folder for storing model definition files
27	%INSTCLTDIR%\sample\conf\client.def	Model client definition file
28	%INSTCLTDIR%\spool	Folder for storing the HADB client's execution results logs
29	%INSTCLTDIR%\vclib	Folder for storing Microsoft(R) Visual Studio(R) 2008 runtime libraries (redistribution module)
30	%INSTCLTDIR%\vclib\msvcr90.dll	Microsoft(R) Visual Studio(R) 2008 C runtime libraries
31	%INSTCLTDIR%\vclib\msvcm90.dll	
32	%INSTCLTDIR%\vclib\msvcp90.dll	Microsoft(R) Visual Studio(R) 2008 C++ runtime libraries
33	%INSTCLTDIR%\vclib\ Microsoft.VC90.CRT.manifest	Manifest file for Microsoft(R) Visual Studio(R) 2008 runtime libraries
34	%INSTCLTDIR%\adbreg.reg	Registry registration command
35	%INSTCLTDIR%\adbunreg.reg	Registry deletion command
36	%INSTCLTDIR%\readme.txt	README file (Release Notes)

Note

The folders and files listed in the preceding table are created when an HADB client is installed. The folders and files listed in the preceding table are deleted when the HADB client is uninstalled.

Table B-2: Structure of the client directory (at installation) (32-bit edition of Windows)

No.	Folder and file names	Description
1	%INSTCLTDIR%\client	Folder for storing the commands, libraries, ODBC driver, and ODBC data source setup program used on the HADB client
2	%INSTCLTDIR%\client\bin	Folder for storing the commands, ODBC driver, and ODBC data source setup program used on the HADB client
3	%INSTCLTDIR%\client\bin\adbclt32.dll	HADB client
4	%INSTCLTDIR%\client\bin\adbodbc32.dll	ODBC driver
5	%INSTCLTDIR%\client\bin\adbodbstp32.dll	ODBC data source setup program
6	%INSTCLTDIR%\client\lib	Folder for storing libraries used with the HADB client
7	%INSTCLTDIR%\client\lib\adbclt32.lib	Client library
8	%INSTCLTDIR%\client\lib\adbjdbc8.jar	JDBC driver (JRE 8)
9	%INSTCLTDIR%\conf	Folder for storing client definitions
10	%INSTCLTDIR%\include	Folder for storing user-provided header files
11	%INSTCLTDIR%\include\adbcnv.h	User-provided header files
12	%INSTCLTDIR%\include\adbcli.h	

No.	Folder and file names	Description
13	%INSTCLTDIR%\include\adbtypes.h	
14	%INSTCLTDIR%\include\adbodb.h	
15	%INSTCLTDIR%\lib	Folder for storing libraries
16	%INSTCLTDIR%\lib\sysdef	Folder for storing definition analysis information files
17	%INSTCLTDIR%\lib\sysdef\adbclt.def	Definition analysis information file
18	%INSTCLTDIR%\sample	Folder for storing a sample application program and model definition files
19	%INSTCLTDIR%\sample\cli_sample1.c	Sample application program
20	%INSTCLTDIR%\sample\Sample1.java	Sample application program (for JDBC)
21	%INSTCLTDIR%\sample\odbc_sample1.c	Sample application program (for ODBC)
22	%INSTCLTDIR%\sample\conf	Folder for storing model definition files
23	%INSTCLTDIR%\sample\conf\client.def	Model client definition file
24	%INSTCLTDIR%\spool	Folder for storing the HADB client's execution results logs
25	%INSTCLTDIR%\vclib32	Folder for storing Microsoft(R) Visual Studio(R) 2008 runtime libraries (redistribution module)
26	%INSTCLTDIR%\vclib32\msvcr90.dll	Microsoft(R) Visual Studio(R) 2008 C runtime libraries
27	%INSTCLTDIR%\vclib32\msvcm90.dll	
28	%INSTCLTDIR%\vclib32\msvcp90.dll	Microsoft(R) Visual Studio(R) 2008 C++ runtime libraries
29	%INSTCLTDIR% \vclib32\Microsoft.VC90.CRT.manifest	Manifest file for Microsoft(R) Visual Studio(R) 2008 runtime libraries
30	%INSTCLTDIR%\adbreg32.reg	Registry registration command
31	%INSTCLTDIR%\adbunreg32.reg	Registry deletion command
32	%INSTCLTDIR%\readme.txt	README file (Release Notes)

Note

The folders and files listed in the preceding table are created when an HADB client is installed. The folders and files listed in the preceding table are deleted when the HADB client is uninstalled.

(2) Structure of client directory (during operation)

For a Windows edition of HADB client, the client directory (at installation) is used as-is during operation.

Table B-3: Structure of the client directory (during operation) (64-bit edition of Windows)

No.	Folder and file names	Description	Time the item is created	Time the item is deleted
1	%ADBCLTDIR%\client	Folder for storing the commands, libraries, ODBC driver, and ODBC data source setup program used on the HADB client	When the HADB client is installed	When the HADB client is uninstalled

No.	Folder and file names	Description	Time the item is created	Time the item is deleted
2	%ADBCLTDIR%\client\bin	Folder for storing the commands, ODBC driver, and ODBC data source setup program used on the HADB client		
3	%ADBCLTDIR%\client\bin\adbclt.dll	HADB client		
4	%ADBCLTDIR%\client\bin\adbodbc.dll	ODBC driver		
5	%ADBCLTDIR%\client\bin\adbodbcstp.dll	ODBC data source setup program		
6	%ADBCLTDIR%\client\bin\adbclt32.dll	HADB client (32-bit)		
7	%ADBCLTDIR%\client\bin\adbodbc32.dll	ODBC driver (32-bit)		
8	%ADBCLTDIR%\client\bin\adbodbstp32.dll	ODBC data source setup program (32-bit)		
9	%ADBCLTDIR%\client\lib	Folder for storing libraries used with the HADB client		
10	%ADBCLTDIR%\client\lib\adbclt.lib	Client library		
11	%ADBCLTDIR%\client\lib\adbclt32.lib	Client library (32-bit)		
12	%ADBCLTDIR%\client\lib\adbjdbc8.jar	JDBC driver (JRE 8)		
13	%ADBCLTDIR%\conf	Folder for storing client definitions		
14	%ADBCLTDIR%\include	Folder for storing user-provided header files		
15	%ADBCLTDIR%\include\adbcnv.h	User-provided header files		
16	%ADBCLTDIR%\include\adbcli.h			
17	%ADBCLTDIR%\include\adbtypes.h			
18	%ADBCLTDIR%\include\adbodb.h			
19	%ADBCLTDIR%\lib	Folder for storing libraries		
20	%ADBCLTDIR%\lib\sysdef	Folder for storing definition analysis information files		
21	%ADBCLTDIR%\lib\sysdef\adbclt.def	Definition analysis information file		
22	%ADBCLTDIR%\sample	Folder for storing a sample application program and model definition files		
23	%ADBCLTDIR%\sample\cli_sample1.c	Sample application program		

No.	Folder and file names	Description	Time the item is created	Time the item is deleted
24	%ADBCLTDIR%\sample\Sample1.java	Sample application program (for JDBC)		
25	%ADBCLTDIR%\sample\odbc_sample1.c	Sample application program (for ODBC)		
26	%ADBCLTDIR%\sample\conf	Folder for storing model definition files		
27	%ADBCLTDIR%\sample\conf\client.def	Model client definition file		
28	%ADBCLTDIR%\spool[#1]	Folder for storing the HADB client's execution results logs		
29	%ADBCLTDIR%\spool\adbmessagecltXX.log	Client message log file[#2]	When the HADB client connects to the HADB server for the first time after the HADB client is installed	
30	%ADBCLTDIR%\spool\.adbmessageclt	Client message log file number management file		
31	%ADBCLTDIR%\vclib	Folder for storing Microsoft(R) Visual Studio(R) 2008 runtime libraries (redistribution module)	When the HADB client is installed	
32	%ADBCLTDIR%\vclib\msvcr90.dll	Microsoft(R) Visual Studio(R) 2008 C runtime libraries		
33	%ADBCLTDIR%\vclib\msvcm90.dll			
34	%ADBCLTDIR%\vclib\msvcp90.dll	Microsoft(R) Visual Studio(R) 2008 C++ runtime libraries		
35	%ADBCLTDIR%\vclib\Microsoft.VC90.CRT.manifest	Manifest file for Microsoft(R) Visual Studio(R) 2008 runtime libraries		
36	%ADBCLTDIR%\adbreg.reg	Registry registration command		
37	%ADBCLTDIR%\adbunreg.reg	Registry deletion command		
38	%ADBCLTDIR%\readme.txt	README file (Release Notes)		
39	%ADBODBTRCPATH%\adbodbtrace_PID_TID_xx.log[#1]	HADB ODBC driver trace file[#3]	When a handle is obtained by using the HADB ODBC driver for the first time	The HADB client does not delete this file.

#1

Grant write permission for the following folders to each OS user who might use the HADB client (including the ODBC driver):

- %ADBCLTDIR%\spool
- %ADBODBTRCPATH%

#2

A maximum of four client message log files are created. The maximum size of each file is specified by using the ADBMSGLOGSIZE environment variable.

#3

A maximum of two HADB ODBC driver trace files are created per process or thread. The maximum size of each file is specified by using the ADBODBTRCSIZE environment variable.

A user needs to create the folder that is used to store HADB ODBC driver trace files. When the folder becomes no longer necessary, a user needs to delete it. The HADB client does not create or delete the folder.

Table B-4: Structure of the client directory (during operation) (32-bit edition of Windows)

No.	Folder and file names	Description	Time the item is created	Time the item is deleted
1	%ADBCLTDIR%\client	Folder for storing the commands, libraries, ODBC driver, and ODBC data source setup program used on the HADB client	When the HADB client is installed	When the HADB client is uninstalled
2	%ADBCLTDIR%\client\bin	Folder for storing the commands, ODBC driver, and ODBC data source setup program used on the HADB client		
3	%ADBCLTDIR%\client\bin\adbclt32.dll	HADB client		
4	%ADBCLTDIR%\client\bin\adbodbc32.dll	ODBC driver		
5	%ADBCLTDIR%\client\bin\adbodbstp32.dll	ODBC data source setup program		
6	%ADBCLTDIR%\client\lib	Folder for storing libraries used with the HADB client		
7	%ADBCLTDIR%\client\lib\adbclt32.lib	Client library		
8	%ADBCLTDIR%\client\lib\adbjdbc8.jar	JDBC driver (JRE 8)		
9	%ADBCLTDIR%\conf	Folder for storing client definitions		
10	%ADBCLTDIR%\include	Folder for storing user-provided header files		
11	%ADBCLTDIR%\include\adbcnv.h	User-provided header files		
12	%ADBCLTDIR%\include\adbcli.h			
13	%ADBCLTDIR%\include\adbtypes.h			
14	%ADBCLTDIR%\include\adbodb.h			
15	%ADBCLTDIR%\lib	Folder for storing libraries		
16	%ADBCLTDIR%\lib\sysdef	Folder for storing definition analysis information files		
17	%ADBCLTDIR%\lib\sysdef\adbclt.def	Definition analysis information file		
18	%ADBCLTDIR%\sample	Folder for storing a sample application program and model definition files		

No.	Folder and file names	Description	Time the item is created	Time the item is deleted
19	%ADBCLTDIR%\sample\cli_sample1.c	Sample application program		
20	%ADBCLTDIR%\sample\Sample1.java	Sample application program (for JDBC)		
21	%ADBCLTDIR%\sample\odbc_sample1.c	Sample application program (for ODBC)		
22	%ADBCLTDIR%\sample\conf	Folder for storing model definition files		
23	%ADBCLTDIR%\sample\conf\client.def	Model client definition file		
24	%ADBCLTDIR%\spool[#1]	Folder for storing the HADB client's execution results logs		
25	%ADBCLTDIR%\spool\adbmessagecltXX.log	Client message log file[#2]	When the HADB client connects to the HADB server for the first time after the HADB client is installed	
26	%ADBCLTDIR%\spool\.adbmessageclt	Client message log file number management file		
27	%ADBCLTDIR%\vclib32	Folder for storing Microsoft(R) Visual Studio(R) 2008 runtime libraries (redistribution module)	When the HADB client is installed	
28	%ADBCLTDIR%\vclib32\msvcr90.dll	Microsoft(R) Visual Studio(R) 2008 C runtime libraries		
29	%ADBCLTDIR%\vclib32\msvcm90.dll			
30	%ADBCLTDIR%\vclib32\msvcp90.dll	Microsoft(R) Visual Studio(R) 2008 C++ runtime libraries		
31	%ADBCLTDIR%\vclib32\Microsoft.VC90.CRT.manifest	Manifest file for Microsoft(R) Visual Studio(R) 2008 runtime libraries		
32	%ADBCLTDIR%\adbreg32.reg	Registry registration command		
33	%ADBCLTDIR%\adbunreg32.reg	Registry deletion command		
34	%ADBCLTDIR%\readme.txt	README file (Release Notes)		
35	%ADBODBTRCPATH%\adbodbtrace_PID_TID_xx.log[#1]	HADB ODBC driver trace file[#3]	When a handle is obtained by using the HADB ODBC driver for the first time	The HADB client does not delete this file.

#1

Grant write permission for the following folders to each OS user who might use the HADB client (including the ODBC driver):

- %ADBCLTDIR%\spool
- %ADBODBTRCPATH%

#2

A maximum of four client message log files are created. The maximum size of each file is specified by using the ADBMSGLOGSIZE environment variable.

#3

A maximum of two HADB ODBC driver trace files are created per process or thread. The maximum size of each file is specified by using the ADBODBTRCSIZE environment variable.

A user needs to create the folder that is used to store HADB ODBC driver trace files. When the folder becomes no longer necessary, a user needs to delete it. The HADB client does not create or delete the folder.

B.2 HADB clients for Linux

(1) Structure of the client directory (at installation)

The table below describes the structure of the client directory (at installation) when a Linux edition of HADB client is installed.

In the table, $INSTCLTDIR indicates the HADB client directory (at installation). The HADB client directory (at installation) is the directory into which the adbinstall command installs HADB client.

Table B-5: Structure of the client directory structure (at installation)

No.	Directory and file names	Description
1	$INSTCLTDIR/client	Directory for storing the commands and libraries used on the HADB client
2	$INSTCLTDIR/client/bin	Directory for storing commands used with the HADB client
3	$INSTCLTDIR/client/bin/adbsql	SQL execution command
4	$INSTCLTDIR/client/lib	Directory for storing libraries used with the HADB client
5	$INSTCLTDIR/client/lib/libadbclt.so	Client library
6	$INSTCLTDIR/client/lib/adbjdbc8.jar	JDBC driver (JRE 8)
7	$INSTCLTDIR/conf	Directory for storing client definitions
8	$INSTCLTDIR/include	Directory for storing user-provided header files
9	$INSTCLTDIR/include/adbcnv.h	User-provided header files
10	$INSTCLTDIR/include/adbcli.h	
11	$INSTCLTDIR/include/adbtypes.h	
12	$INSTCLTDIR/include/adbodb.h	
13	$INSTCLTDIR/lib	Directory for storing libraries
14	$INSTCLTDIR/lib/adbmsg.cat	Message catalog file
15	$INSTCLTDIR/lib/sysdef	Directory for storing definition analysis information files
16	$INSTCLTDIR/lib/sysdef/adbclt.def	Definition analysis information file
17	$INSTCLTDIR/sample	Directory for storing a sample application program and model definition files
18	$INSTCLTDIR/sample/cli_sample1.c	Sample application program
19	$INSTCLTDIR/sample/Sample1.java	Sample application program (for JDBC)
20	$INSTCLTDIR/sample/odbc_sample1.c	Sample application program (for ODBC)

No.	Directory and file names	Description
21	$INSTCLTDIR/sample/conf	Directory for storing model definition files
22	$INSTCLTDIR/sample/conf/client.def	Model client definition file
23	$INSTCLTDIR/spool	Directory for storing the HADB client's execution results logs

Note

The directories and files listed in the preceding table are created when an HADB client is installed. The directories and files listed in the preceding table are deleted when the HADB client is uninstalled.

(2) Structure of the client directory (during operation)

The following table describes the structure of the client directory (during operation).

Table B-6: Structure of the client directory (during operation)

No.	Directory and file names	Description	Time the item is created	Time the item is deleted
1	$ADBCLTDIR/client	Directory for storing the commands and libraries used on the HADB client	When the HADB client is installed	When the HADB client is uninstalled
2	$ADBCLTDIR/client/bin	Directory for storing commands used with the HADB client		
3	$ADBCLTDIR/client/bin/adbsql	SQL execution command		
4	$ADBCLTDIR/client/lib	Directory for storing libraries used with the HADB client		
5	$ADBCLTDIR/client/lib/libadbclt.so	Client library		
6	$ADBCLTDIR/client/lib/adbjdbc8.jar	JDBC driver (JRE 8)		
7	$ADBCLTDIR/conf	Directory for storing client definitions		
8	$ADBCLTDIR/include	Directory for storing user-provided header files		
9	$ADBCLTDIR/include/adbcnv.h	User-provided header files		
10	$ADBCLTDIR/include/adbcli.h			
11	$ADBCLTDIR/include/adbtypes.h			
12	$ADBCLTDIR/include/adbodb.h			
13	$ADBCLTDIR/lib	Directory for storing libraries		
14	$ADBCLTDIR/lib/adbmsg.cat	Message catalog file		
15	$ADBCLTDIR/lib/sysdef	Directory for storing definition analysis information files		

No.	Directory and file names	Description	Time the item is created	Time the item is deleted
16	`$ADBCLTDIR/lib/sysdef/adbclt.def`	Definition analysis information file		
17	`$ADBCLTDIR/sample`	Directory for storing a sample application program and model definition files		
18	`$ADBCLTDIR/sample/cli_sample1.c`	Sample application program		
19	`$ADBCLTDIR/sample/Sample1.java`	Sample application program (for JDBC)		
20	`$ADBCLTDIR/sample/odbc_sample1.c`	Sample application program (for ODBC)		
21	`$ADBCLTDIR/sample/conf`	Directory for storing model definition files		
22	`$ADBCLTDIR/sample/conf/client.def`	Model client definition file		
23	`$ADBCLTDIR/spool`	Directory for storing the HADB client's execution results logs		
24	`$ADBCLTDIR/spool/adbmessagecltXX.log`	Client message log file[#]	When the HADB client connects to the HADB server for the first time after the HADB client is installed	
25	`$ADBCLTDIR/spool/.adbmessageclt`	Client message log file number management file		

#

A maximum of four client message log files are created. The maximum size of each file is specified by using the `ADBMSGLOGSIZE` environment variable.

C. Estimating the Memory Requirements for an HADB Client

This appendix explains how to estimate the memory requirements for an HADB client.

C.1 Memory required for connecting to the HADB server

The size of the memory required to connect to the HADB server is 12 kilobytes.

C.2 Memory required for communication between an HADB client and the HADB server

Use the formula shown below to determine the size of the memory required for communication between an HADB client and the HADB server.

Formula (kilobytes)

Memory used during communication = *RPCC* + *SBF* + 256

(1) How to obtain the value of RPCC

Use the following formula to obtain the value of *RPCC* (communication management information).

Formula (kilobytes)

$$RPCC = \left\lceil \left\{ 3{,}488 + (\uparrow SMSG \div 4{,}096 \uparrow \times 4{,}096) \times (2 \times max\_sql\_concurrent\_exec\_num + 1) \right\} \div 1{,}024 \right\rceil$$

SMSG: HADB server's transmission data size

For details about how to obtain *SMSG*, see the description of the *SMSG* variable in the following section in the *HADB Setup and Operation Guide*: *Determining the variable RTHD_COMMUSZ* in *Determining the real thread private memory requirement (during normal operation)* in *Determining the memory requirement during normal operation*

max_sql_concurrent_exec_num:

Maximum number of SQL statements that can be executed concurrently in one transaction

(2) How to obtain the value of SBF

The initial allocation size for *SBF* (transmission buffer) is 4,096 bytes.

If transmission data that is created exceeds the initial allocation size, an additional allocation is made of the size obtained from the following formula.

Formula (additional allocation) (kilobytes)

$SBF = \uparrow CMSG \div 4{,}096 \uparrow \times 4$

CMSG: Size of the transmission data from the HADB client

The size of the transmission data depends on the nature of the processing. The following table shows the size of the transmission data that is allocated for each type of processing.

Table C-1: Size of transmission data that is allocated for each type of processing

No.	Type of processing	Formula for determining the transmission data size (bytes)
1	Preprocessing and execution	*CMSG = clt_base_info + clt_exec_direct + clt_sql_text*
2	Cursor open processing	*CMSG = clt_base_info + clt_open + PARAM_DATA*
3	SQL statement execution	*CMSG = clt_base_info + clt_exec + PARAM_DATA*
4	Preprocessing	*CMSG = clt_base_info + clt_prepare + clt_sql_text*

Legend:

clt_base_info: Basic information about transmission data
Assign 32 bytes.

clt_exec_direct: Information specific to preprocessing and execution
Assign 44 bytes.

*clt_sql_tex*t: Size of SQL text
Assign the length of the SQL statement to be executed (bytes).

clt_open: Information specific to cursor open processing
Assign 48 bytes.

clt_exec: Information specific to execution
Assign 48 bytes

clt_prepare: Information specific to preprocessing
Assign 44 bytes.

PARAM_DATA: Information about dynamic parameters
Use the following formula to obtain the value:
Formula (bytes)

$$PARAM\_DATA = \left\{ \sum_{i=1}^{param\_num} param\_size_{(i)} \right\} \times array\_num$$

param_num: Number of dynamic parameters specified in the SQL statement
param_size(i): Data size specified for each dynamic parameter
array_num: Number of sets of dynamic parameters updated in the batch mode#

#
The value depends on the application program implementation method:

- Application programs that use CLI functions
  The value is the number specified in the `ArrayCount` argument of `a_rdb_SQLBindArrayParams()`. If the parameters were bound by using `a_rdb_SQLBindParams()`, the value is always `1`.
- Application programs that use a JDBC driver
  The value is the number of parameter lists registered by using the `addBatch` method. If you do not use `executeBatch` or `executeLargeBatch`, the value is always `1`.

Index

A

a_rdb_CNV_BINARYchar() 1000
a_rdb_CNV_charBINARY() 989
a_rdb_CNV_charDATE() 991
a_rdb_CNV_charDECIMAL() 992
a_rdb_CNV_charTIME() 994
a_rdb_CNV_charTIMESTAMP() 996
a_rdb_CNV_charVARBINARY() 998
a_rdb_CNV_DATEchar() 1002
a_rdb_CNV_DECIMALchar() 1004
a_rdb_CNV_TIMEchar() 1006
a_rdb_CNV_TIMESTAMPchar() 1007
a_rdb_CNV_VARBINARYchar() 1009
a_rdb_SQLAllocConnect() 961
a_rdb_SQLAllocStmt() 970
a_rdb_SQLBindArrayParams() 971
a_rdb_SQLBindCols() 973
a_rdb_SQLBindParams() 974
a_rdb_SQLCancel() 967
a_rdb_SQLCloseCursor() 975
a_rdb_SQLColumnInfo_t structure 1015
a_rdb_SQLConnect() 962
a_rdb_SQLDataType_t structure 1017
a_rdb_SQLDescribeCols() 976
a_rdb_SQLDescribeParams() 979
a_rdb_SQLDisconnect() 965
a_rdb_SQLEndTran() 968
a_rdb_SQLExecDirect() 981
a_rdb_SQLExecute() 982
a_rdb_SQLFetch() 983
a_rdb_SQLFreeConnect() 966
a_rdb_SQLFreeStmt() 984
a_rdb_SQLInd_t 1018
a_rdb_SQLNameInfo_t structure 1016
a_rdb_SQLNumParams() 985
a_rdb_SQLNumResultCols() 986
a_rdb_SQLParameterInfo_t structure 1018
a_rdb_SQLPrepare() 987
a_rdb_SQLResultInfo_t structure 1019
a_rdb_SQLSetConnectAttr() 963
abbreviations for products 10
absolute(int row) 429
acceptsURL(String url) 323
access path 224
  example of 228
acquiring dynamic parameter information
  a_rdb_SQLDescribeParams() 979
  example of using CLI function 949
acquiring number of dynamic parameters
  a_rdb_SQLNumParams() 985
  example of using CLI function 949
acronyms 11
action to take when hash table area has insufficient space 141
ADB_AUDITREAD 254
adb_clt_ap_name
  client definition 47
  system property 62
  URL for connection 268
  user property 273
adb_clt_fetch_size
  client definition 48
  system property 62
  URL for connection 268
  user property 273
adb_clt_group_name
  client definition 46
  system property 62
  URL for connection 268
  user property 273
adb_clt_rpc_con_wait_time
  client definition 47
  system property 62
  URL for connection 268
  user property 273
adb_clt_rpc_sql_wait_time
  client definition 47
  system property 62
  URL for connection 268
  user property 273
adb_clt_rpc_srv_host
  client definition 46
  system property 62
  user property 273
adb_clt_rpc_srv_port
  client definition 46
  system property 62
  user property 273

adb_clt_sql_order_mode
client definition 53
system property 62
URL for connection 268
user property 273
adb_clt_sql_text_out
client definition 53
system property 62
URL for connection 268
user property 273
adb_clt_trn_access_mode
client definition 53
system property 62
URL for connection 268
user property 273
adb_clt_trn_iso_lv
client definition 53
system property 62
URL for connection 268
user property 273
ADB_CSVREAD 254
adb_dbbuff_wrktbl_clt_blk_num
client definition 48
system property 62
URL for connection 268
user property 273
adb_jdbc_cache_info_max 310
adb_jdbc_exc_trc_out_path 310
adb_jdbc_info_max 310
adb_jdbc_trc_out_lv 310
adb_sql_exe_hashflt_area_size
client definition 48
system property 62
URL for connection 268
user property 273
adb_sql_exe_hashgrp_area_size
client definition 48
system property 62
URL for connection 268
user property 273
adb_sql_exe_hashtbl_area_size
client definition 48
system property 62
URL for connection 268
user property 273
adb_sql_exe_max_rthd_num
client definition 48
system property 62
URL for connection 268
user property 273
adb_sql_prep_dec_div_rs_prior
client definition 53
system property 62
URL for connection 268
user property 273
adb_sql_prep_delrsvd_use_srvdef
client definition 53
system property 62
URL for connection 268
user property 273
ADBCLTDIR (environment variable) 82, 83
ADBCLTLANG (environment variable) 82, 83
ADBMSGLOGSIZE (environment variable) 82, 83
ADBODBAPMODE (environment variable) 82
ADBODBTRC (environment variable) 82
ADBODBTRCLV (environment variable) 82
ADBODBTRCPATH (environment variable) 82
ADBODBTRCSIZE (environment variable) 82
ADBSQLNULLCHAR (environment variable) 83
addBatch
PreparedStatement interface 400
Statement interface 369
addConnectionEventListener 657
added scalar function 689
advancing OS time on client machine 97
afterLast() 430
allocating connection handle
example of using CLI function 941
allProceduresAreCallable() 509
allTablesAreSelectable() 509
antivirus software
reviewing scope of scans (if JDBC driver is used) 65
application identifier 47
application program
designing 936
tuning 223
archivable multi-chunk table
considerations when searching 206
equivalent exchange of SQL statement 215
AUTOCOMMIT specification 714
autoCommitFailureClosesAllResultSets() 510
automatic commit mode, setting 354
automatic loading (java.sql.Driver) 687

B

B-tree index
  evaluation method using 131
B-tree index selection rules 112
B-tree indexes used during execution of SQL statement 105
batch binding of dynamic parameters 971
batch transfer
  dynamic parameter values 221
  retrieval results 219
batch transfer of dynamic parameter values 221
batch transfer of retrieval results 219
batch update functionality, scope of support for 640
beforeFirst() 431
BINARY-type data
  converting (CLI function) 1000
  converting to (CLI function) 989
binding of dynamic parameters
  a_rdb_SQLBindParams() 974
  example of using CLI function 951
bookmark functionality 704
boundary alignment 954
BUILD COLUMN 252, 254

C

cancel() 370
changing OS time 97
changing OS time on client machine 97
character encoding conversion
  ODBC 702
character string displaying null value 83
checking access paths 226
checking index used for retrieval 121
class name
  Connection interface 332
  DatabaseMetaData interface 508
  Driver interface 322
  PreparedStatement interface 400
  ResultSet interface 429
  ResultSetMetaData interface 619
  Statement interface 369
CLASSPATH (environment variable) 61
clearBatch() 371
clearParameters() 400
clearWarnings()
  Connection interface 332
  ResultSet interface 432
  Statement interface 371
CLI function 956
client definition, creating 85
client directory 76, 77
client directory (Linux)
  structure of (at installation) 1036
  structure of (during operation) 1037
client directory (Windows)
  structure of (at installation) 1029
  structure of (during operation) 1031
client group name 46
client library 957
close()
  Connection interface 333
  PooledConnection interface 658
  ResultSet interface 432
  Statement interface 372
closeOnCompletion() 372
closing connection
  example of using CLI function 942
COLUMN 237
COLUMN STORE 242
columns of work table 172
commit() 333
compiling 955
concurrent processing type 639
connect(String url, Properties info) 323
connection
  closing (a_rdb_SQLDisconnect()) 965
  establishing (a_rdb_SQLConnect()) 962
  establishing (CLI function, example of) 942
connection attribute, setting 963
connection handle
  allocating (a_rdb_SQLAllocConnect()) 961
  releasing (a_rdb_SQLFreeConnect()) 966
connection information priorities 280
Connection interface 330
connection management 689
Connection Number 256
connection pool 646
ConnectionPoolDataSource interface 652
conventions
  abbreviations for products 10
  acronyms 11
  fonts and symbols 12
  KB, MB, GB, TB, PB, and EB 15

version numbers 15
conversion of character encoding (JDBC) 301
CREATE FILTER 252, 254
CREATE GLOBAL WORK TABLE 234
CREATE LOCAL WORK TABLE 234
createStatement() 334
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability) 336
createStatement(int resultSetType, int resultSetConcurrency) 335
CROSS JOIN 251
cursor
closing 432
closing (a_rdb_SQLCloseCursor()) 975
closing (CLI function, example of) 947
moving 429
opening (a_rdb_SQLExecute()) 982
opening (CLI function, example of) 946
using, to retrieve rows (CLI function, example of) 946

D

data description 1012
data source
acquiring system information for 826
connecting to (ODBC) 723
disconnecting from (ODBC) 864
data source information, acquiring 738
data types
correspondence between (ODBC) 707
mapping 290
DatabaseMetaData interface 500
dataDefinitionCausesTransactionCommit() 510
dataDefinitionIgnoredInTransactions() 511
DataSource interface 647
DataSource object
generating 278
registering into JNDI 278
DATE-type data
converting (CLI function) 1002
converting to (CLI function) 991
DECIMAL-type data
converting (CLI function) 1004
converting to (CLI function) 992
DELEGATION 235
DELETE STATEMENT 230
deletesAreDetected(int type) 511
deleting data sources (ODBC) 706
DERIVED TABLE 231
descriptor value, specifying 760
designing client definition 44
details view 224
information displayed in 246
DISTINCT 240
doesMaxRowSizeIncludeBlobs() 512
downgrading (HADB client) 92
Driver class, how to register 267
driver information, acquiring 738
Driver interface 322
driver option
obtaining 749
specifying 749
DriverPropertyInfo field setting 326
dynamic parameter
designing application program 937
how to use CLI function 948

E

EB meaning 15
environment setup (JDBC driver) 59
environment variable
CLASSPATH 61
TZ (if JDBC driver is used) 61
environment variable, specifying 82
equivalent exchange
converting to IN conditions 189
converting to WHERE clause 200
equivalent exchange to derived table for which UNION ALL set operation is specified 191
IN predicate 199
removing from OR condition 184
scalar operation 197
equivalent exchange (search conditions) 184
Equivalent exchange for a HAVING clause 200
equivalent exchange for scalar operations 197
equivalent exchange of search conditions 184
equivalent exchange related to IN predicates 199
equivalent exchange related to OR conditions
converting to IN conditions 189
equivalent exchange to derived table for which UNION ALL set operation is specified 191
removing from OR condition 184
error
how to evaluate (SQL statement) 939
information that is returned in event of (ODBC) 713

escape clause 328
escape syntax analysis 391
evaluation method
  range index 133
  using B-tree index 131
evaluation method using range index 133
event listener, registering 657
EXCEPT ALL 253
EXCEPT DISTINCT 253
exception trace log 305
execute
  PreparedStatement interface 401
  Statement interface 373
executeBatch() 374
executeLargeBatch() 375
executeLargeUpdate() 402
executeLargeUpdate(String sql) 375
executeQuery
  PreparedStatement interface 402
  Statement interface 376
executeUpdate
  PreparedStatement interface 403
  Statement interface 377
execution plan (SQL statement) 224
execution result information, acquiring 798
execution result, acquiring 798

F

FILTER 230, 245
findColumn(String columnName) 433
first() 434
font conventions 12
FULL OUTER JOIN 251
function for centrally managing client definitions (excluded operands) 58
FUNCTION NAME 254

G

GB meaning 15
generatedKeyAlwaysReturned() 513
getApName() 662
getAsciiStream(int columnIndex) 434
getAsciiStream(String columnName) 435
getAttributes 513
getAutoCommit() 337
getBestRowIdentifier 514
getBigDecimal(int columnIndex) 436
getBigDecimal(String columnName) 438
getBinaryStream(int columnIndex) 439
getBinaryStream(String columnName) 440
getBoolean(int columnIndex) 440
getBoolean(String columnName) 442
getByte(int columnIndex) 443
getByte(String columnName) 444
getBytes(int columnIndex) 445
getBytes(String columnName) 446
getCatalog() 337
getCatalogName(int column) 619
getCatalogs() 516
getCatalogSeparator() 516
getCatalogTerm() 517
getCharacterStream(int columnIndex) 447
getCharacterStream(String columnName) 448
getClientInfoProperties() 517
getColumnClassName(int column) 620
getColumnCount() 621
getColumnDisplaySize(int column) 621
getColumnLabel(int column) 623
getColumnName(int column) 623
getColumnPrivileges 518
getColumns 519
getColumnType(int column) 624
getColumnTypeName(int column) 624
getConcurrency() 449
getConnection
  DatabaseMetaData interface 521
  DataSource interface 647, 648
  PooledConnection interface 658
  Statement interface 378
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable) 522
getCursorName() 449
getDatabaseMajorVersion() 523
getDatabaseMinorVersion() 524
getDatabaseProductName() 525
getDatabaseProductVersion() 525
getDate(int columnIndex, Calendar cal) 451
getDate(int columnIndex) 450
getDate(String columnName, Calendar cal) 453
getDate(String columnName) 452
getDefaultTransactionIsolation() 526

getDouble(int columnIndex) 454
getDouble(String columnName) 456
getDriverMajorVersion() 526
getDriverMinorVersion() 527
getDriverName() 527
getDriverVersion() 528
getEncodeLang() 662
getExportedKeys 528
getExtraNameCharacters() 530
getFetchDirection()
ResultSet interface 457
Statement interface 378
getFetchSize()
ResultSet interface 457
Statement interface 379
getFloat(int columnIndex) 458
getFloat(String columnName) 460
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern) 530
getFunctions(String catalog, String schemaPattern, String functionNamePattern) 531
getHADBConnectionID() 338
getHADBConnectionSerialNum() 338
getHADBOrderMode() 339
getHADBSQLHashFltSize() 339
getHADBSQLHashTblSize() 340
getHADBSQLMaxRthdNum() 341
getHADBSQLSerialNum()
PreparedStatement interface 404
Statement interface 379
getHADBStatementHandle()
PreparedStatement interface 404
Statement interface 380
getHADBTransactionID() 341
getHoldability()
Connection interface 342
ResultSet interface 461
getHostName() 666
getIdentifierQuoteString() 532
getImportedKeys 533
getIndexInfo 534
getInt(int columnIndex) 462
getInt(String columnName) 463
getInterfaceMethodTrace() 663
getJDBCMajorVersion() 536
getJDBCMinorVersion() 537
getLargeMaxRows() 381
getLargeUpdateCount() 381
getLoginTimeout()
ConnectionPoolDataSource interface 652
DataSource interface 649
getLogWriter()
ConnectionPoolDataSource interface 653
DataSource interface 649
getLong(int columnIndex) 464
getLong(String columnName) 466
getMajorVersion() 324
getMaxBinaryLiteralLength() 537
getMaxCatalogNameLength() 538
getMaxCharLiteralLength() 538
getMaxColumnNameLength() 539
getMaxColumnsInGroupBy() 539
getMaxColumnsInIndex() 540
getMaxColumnsInOrderBy() 540
getMaxColumnsInSelect() 541
getMaxColumnsInTable() 541
getMaxConnections() 542
getMaxCursorNameLength() 542
getMaxFieldSize() 382
getMaxIndexLength() 543
getMaxLogicalLobSize() 543
getMaxProcedureNameLength() 544
getMaxRows() 383
getMaxRowSize() 544
getMaxSchemaNameLength() 545
getMaxStatementLength() 545
getMaxStatements() 546
getMaxTableNameLength() 546
getMaxTablesInSelect() 547
getMaxUserNameLength() 547
getMetaData()
Connection interface 343
PreparedStatement interface 405
ResultSet interface 467
getMinorVersion() 325
getMoreResults() 383
getNotErrorOccurred() 663
getNumericFunctions() 548
getObject(int columnIndex,Class<T> type) 470
getObject(int columnIndex) 467
getObject(String columnLabel,Class<T> type) 472
getObject(String columnName) 469
getParameterClassName(int param) 678

getParameterCount() 679
getParameterMetaData() 406
getParameterMode(int param) 679
getParameterType(int param) 680
getParameterTypeName(int param) 680
getPassword() 664
getPooledConnection() 653
getPooledConnection(String user, String password) 654
getPort() 667
getPrecision(int column) 625
getPrecision(int param) 681
getPrimaryKeys(String catalog, String schema, String table) 548
getProcedureColumns 549
getProcedures 551
getProcedureTerm() 552
getPropertyInfo(String url, Properties info) 325
getPseudoColumns(String catalog,String schemaPattern,String tableNamePattern,String columnNamePattern) 552
getQueryTimeout() 384
getResultSet() 384
getResultSetConcurrency() 385
getResultSetHoldability()
  DatabaseMetaData interface 553
  Statement interface 385
getResultSetType() 386
getRow() 473
getRowIdLifetime() 554
getScale(int column) 627
getScale(int param) 682
getSchema() 343
getSchemaName(int column) 627
getSchemas() 554
getSchemas(String catalog, String schemaPattern) 555
getSchemaTerm() 556
getSearchStringEscape() 556
getShort(int columnIndex) 473
getShort(String columnName) 475
getSQLKeywords() 557
getSQLStateType() 557
getSQLWarningKeep() 665
getStatement() 476
getString(int columnIndex) 476
getString(String columnName) 478
getStringFunctions() 558
getSuperTables 558
getSuperTypes 559
getSystemFunctions() 560
getTableName(int column) 628
getTablePrivileges 561
getTables 562
getTableTypes() 564
getTime(int columnIndex, Calendar cal) 480
getTime(int columnIndex) 479
getTime(String columnName, Calendar cal) 482
getTime(String columnName) 481
getTimeDateFunctions() 564
getTimestamp(int columnIndex, Calendar cal) 484
getTimestamp(int columnIndex) 483
getTimestamp(String columnName, Calendar cal) 486
getTimestamp(String columnName) 485
getTraceNumber() 665
getTransactionIsolation() 344
getType() 486
getTypeInfo() 565
getTypeMap() 344
getUDTs 566
getUpdateCount() 387
getURL() 567
getUser() 666
getUserName() 568
getVersionColumns 568
getWarnings()
  Connection interface 345
  ResultSet interface 487
  Statement interface 388
global hash grouping 160
GLOBAL HASH GROUPING 237
GLOBAL HASH UNIQUE 237
global work table 171
GROUPING 237
grouping method 159
  hash grouping 159
  sort grouping 161
GROUPING SET 237, 255
grouping set information 237, 255

H

HADB client
  downgrading 92
  estimating memory requirements for 1039
  uninstalling (for Linux) 81

version upgrade 88
HADB client (for Linux)
installing 79
HADB client (for Windows)
installing 76
uninstalling 77
HADB ODBC driver environment setup 705
HADB ODBC driver trace information 910
output information 923
HADB server
connecting to (CLI function, example of) 941
disconnecting from (CLI function, example of) 942
how to connect to (Java) 267
handling unresponsive application programs 86
if JDBC driver is used 66
handling unresponsiveness
if JDBC driver is used 66
if ODBC driver or CLI functions are used 86
HASH 248
hash execution
method for processing SELECT DISTINCT 167
method for processing set operation 163
methods for processing subqueries that contain external reference column 154
methods for processing subqueries that do not contain external reference column 148
hash group area size 48
hash grouping 159
global hash grouping 160
local hash grouping 159
hash join 138
HASH JOIN 245, 250
hash retrieval information 252
hash table 138
hashing 138
HAVING 239
header file 957
host
URL for connection 268
host name, specifying 46

I

identification information view 224
implicit cursor 960
INDEX 237
INDEX COLUMN 248
INDEX NAME 248
index priority 106
index scan 101
INDEX SCAN 242, 247
index specification, using 119
INDEX TYPE 248
indicator 1018
information displayed for access path (alphabetical order) 257
INNER JOIN 251
inner table 138
INSERT STATEMENT 230
insertsAreDetected(int type) 569
installing
Java Development Kit 60
Java Runtime Environment 60
JDBC driver 60
internal derived table, expanding 218
INTERSECT ALL 253
INTERSECT DISTINCT 253
isAfterLast() 488
isAutoIncrement(int column) 629
isBeforeFirst() 488
isCaseSensitive(int column) 629
isCatalogAtStart() 570
isClosed()
Connection interface 345
ResultSet interface 489
Statement interface 388
isCloseOnCompletion() 389
isCurrency(int column) 630
isDefinitelyWritable(int column) 630
isFirst() 490
isLast() 490
isNullable(int column) 631
isNullable(int param) 683
isPoolable() 389
isReadOnly()
Connection interface 346
DatabaseMetaData interface 570
isReadOnly(int column) 631
isSearchable(int column) 632
isSigned(int column) 633
isSigned(int param) 684
isValid(int timeout) 346
isWrapperFor(Class<?> iface) 691
isWritable(int column) 633

J

java.sql.Driver, automatic loading of 687
JDBC 1.2 API 321
JDBC driver
  installing 60
  replacing with revised version 70
  uninstalling 72
  upgrading 68
JDBC interface method trace 303
JDBC standards compliance, scope of 263
jdbc:hadb 268
jdbcCompliant() 328
JNDI 277
JNDI support 646
join methods 137
join type 251
JOIN TYPE 251
joined column 138

K

KB meaning 15
key 102
key condition (B-tree index) 132
key scan 102
KEY SCAN 242, 247

L

LANG (environment variable) 83
large update counts 699
last() 491
LD_LIBRARY_PATH (environment variable) 83
LEFT OUTER JOIN 251
LIMIT 241
linking 955
list of error cause codes (return values of CLI functions) 1022
local hash grouping 159
LOCAL HASH GROUPING 237
local work table 171
locatorsUpdateCopy() 571
log writer, setting
  ConnectionPoolDataSource interface 655
  DataSource interface 651

M

MB meaning 15
memory requirement, estimating (for HADB Client) 1039
method for processing SELECT DISTINCT 167
  hash execution 167
  work table execution 169
method for processing set operation 163
  hash execution 163
  work table execution 165
methods for searching tables 100
methodtrace
  URL for connection 268
  user property 273

N

nativeSQL(String sql) 347
NESTED LOOP JOIN 245, 250
nested loops row value execution 154
nested loops work table execution 153
nested-loop join 137
next() 492
non-retrieval SQL statement 373
nullPlusNonNullIsNull() 571
nullsAreSortedAtEnd() 572
nullsAreSortedAtStart() 572
nullsAreSortedHigh() 573
nullsAreSortedLow() 573
number-of-batch-transmission-rows 48

O

ODBC
  troubleshooting 912
ODBC cursor library 704
ODBC driver environment setup 705
ODBC functions, list of 718
ODBC traces 910
ORDER 243
OS time, changing
  notes 97
othersDeletesAreVisible(int type) 574
othersInsertsAreVisible(int type) 574
othersUpdatesAreVisible(int type) 575
outer table 138
overflow handling 296
ownDeletesAreVisible(int type) 575
ownInsertsAreVisible(int type) 576
ownUpdatesAreVisible(int type) 577

P

package name
Connection interface 332
DatabaseMetaData interface 508
Driver interface 322
JAR file 265
PreparedStatement interface 400
ResultSet interface 429
ResultSetMetaData interface 619
Statement interface 369
package name of JAR file 265
ParameterMetaData interface 677
password
URL for connection 268
user property 273
PATH (environment variable) 82, 83
pattern character string, special character that can be specified in 509
PB meaning 15
PooledConnection interface 657
port
URL for connection 268
port number, specifying 46
PreparedStatement interface 398
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) 352
prepareStatement(String sql, int resultSetType, int resultSetConcurrency) 351
prepareStatement(String sql) 350
preprocessing and execution of SQL statements
a_rdb_SQLExecDirect() 981
preprocessing of SQL statements
a_rdb_SQLPrepare() 987
previous() 492
priority (connection information) 280
PROBE COLUMN 252, 254
processing subqueries
that contain external reference column 152
that do not contain external reference column 146
property
URL for connection 268
PURGE CHUNK STATEMENT 230

Q

QUERY 234
QUERY SCAN 244
query tree 224
query tree number 224

R

range index condition 133
skipping chunks 133
skipping segments 133
range indexes used during execution of SQL statement 123
range search condition (B-tree index) 131
read-only mode 53
read-only transaction 53
read/write mode 53
read/write transaction 53
RECURSIVE 232
registering registry key
HADB client 76
ODBC 706
relative(int rows) 493
releasing connection handle
example of using CLI function 943
removeConnectionEventListener 659
replacing (revised version of HADB client) 95
replacing HADB client with revised version 95
replacing JAR file 68
restoring OS time on client machine 97
result set extended function, scope of support for 639
ResultSet interface 426
ResultSetMetaData interface 618
retrieval methods 100
retrieval result
acquiring 384
retrieval result column
acquiring information about (a_rdb_SQLDescribeCols()) 976
acquiring information about (CLI function, example of) 945
associating (a_rdb_SQLBindCols()) 973
associating (CLI function, example of) 946
retrieval result columns
acquiring number of (a_rdb_SQLNumResultCols()) 986
acquiring number of (CLI function, example of) 945
retrieval SQL statement 373
return values (CLI functions) 1022
return values of CLI functions 1022
reviewing scope of scans
antivirus software (if JDBC driver is used) 65

RIGHT OUTER JOIN 251
rollback() 353
row ID 172
ROW specification 714
row value execution 147
row, fetching (a_rdb_SQLFetch()) 983
RowSets 646

S

sample application program 1025
sample1 1025
scalar function
  added 689
  that can be specified in escape clause 319
scrollable cursors 704
scrolling type 639
search condition
  how to evaluate 131
SELECT STATEMENT 230
selection rules for text indexes 112
SET OPERATION 232
SET OPERATION TYPE 253
setApName(String name) 667
setAsciiStream 406
setAutoCommit(boolean autoCommit) 354
setBigDecimal(int parameterIndex, BigDecimal x) 407
setBinaryStream(int parameterIndex, InputStream x, int length) 408
setBoolean(int parameterIndex, boolean x) 409
setByte(int parameterIndex, byte x) 409
setBytes(int parameterIndex, byte[] x) 410
setCatalog(String catalog) 354
setCharacterStream 411
setCursorName(String name) 390
setDate(int parameterIndex, Date x, Calendar cal) 412
setDate(int parameterIndex, Date x) 411
setDouble(int parameterIndex, double x) 413
setEncodeLang(String lang) 668
setEscapeProcessing(boolean enable) 391
setFetchDirection(int direction)
  ResultSet interface 494
  Statement interface 391
setFetchSize(int rows)
  ResultSet interface 494
  Statement interface 392
setFloat(int parameterIndex, float x) 414
setHADBAuditInfo(int pos,String userinfo) 355
setHADBOrderMode(int mode) 356
setHADBSQLHashFltSize(int areaSize) 358
setHADBSQLHashTblSize(int areaSize) 359
setHADBSQLMaxRthdNum(int rthdNum) 360
setHoldability(int holdability) 363
setHostName(String name) 673
setInt(int parameterIndex, int x) 414
setInterfaceMethodTrace(boolean flag) 669
setLargeMaxRows(long max) 393
setLoginTimeout(int seconds)
  ConnectionPoolDataSource interface 655
  DataSource interface 650
setLogWriter (DataSource interface) 651
setLogWriter(PrintWriter out) (ConnectionPoolDataSource interface) 655
setLong(int parameterIndex, long x) 415
setMaxFieldSize(int max) 394
setMaxRows(int max) 395
setNotErrorOccurred(boolean mode) 670
setNull(int parameterIndex,int sqlType) 416
setObject(int parameterIndex, Object x, int targetSqlType, int scale) 418
setObject(int parameterIndex, Object x, int targetSqlType) 417
setObject(int parameterIndex, Object x) 416
setPassword(String password) 670
setPort(int port) 673
setQueryTimeout(int seconds) 395
setReadOnly(boolean readOnly) 363
setSchema(String schema) 364
setShort(int parameterIndex, short x) 419
setSQLWarningKeep(boolean mode) 671
setString(int parameterIndex, String x) 420
setTime(int parameterIndex, Time x, Calendar cal) 421
setTime(int parameterIndex, Time x) 420
setTimestamp(int parameterIndex, Timestamp x, Calendar cal) 423
setTimestamp(int parameterIndex, Timestamp x) 422
setting system properties 62
setting up environment (HADB client) 73
setting up environment for HADB client 73
setting up environment for JDBC driver 59
settings for outputting HADB ODBC driver trace information 917
setTraceNumber(int num) 672
setTransactionIsolation(int level) 365
setUser(String user) 672
size of hash filter area 48

size of hash table area 48
SKIP COND 248
sort grouping 161
SORT GROUPING 237
SORTING 239
special character 509
SPECIFIC 233, 235, 237, 240, 243, 246
SPECIFIC DISABLED 243, 246
specifying character encoding
  in Linux 83
  in Windows 82
specifying data sources (ODBC) 705
specifying environment variables
  Linux 83
  Windows 82
specifying transaction access mode
  for client definition 53
  system property 62
  URL for connection 268
  user property 273
specifying URL 268
specifying user properties 273
SQL data types, correspondence 290
SQL exception extension 688
SQL exception extension function (JDBC 4.0 API) 693
SQL processing, canceling (CLI function) 967
SQL request, creating 771
SQL Serial Number 256
SQL statement
  adding to batch 369
  canceling 370
  executing 783
  executing (a_rdb_SQLExecute()) 982
  executing (CLI function, example of) 946
  execution plan 224
  how to evaluate error 939
  preprocessing (CLI function, example of) 945
  preprocessing and executing (CLI function, example of) 951
  terminating execution of 858
  that creates work tables, considerations when executing 171
SQL statement identification information 256
SQLAllocHandle 723
SQLBindCol 806
SQLBindParameter 773
SQLBrowseConnect, SQLBrowseConnectW 733
SQLBulkOperations 817
SQLCancel 860
SQLCloseCursor 859
SQLColAttribute, SQLColAttributeW 803
SQLColumnPrivileges, SQLColumnPrivilegesW 826
SQLColumns, SQLColumnsW 829
SQLConnect, SQLConnectW 724
SQLCopyDesc 769
SQLDataSources, SQLDataSourcesW 738
SQLDescribeCol, SQLDescribeColW 800
SQLDescribeParam 780
SQLDisconnect 864
SQLDriverConnect, SQLDriverConnectW 727
SQLDrivers, SQLDriversW 740
SQLEndTran 861
SQLException interface 635
SQLExecDirect, SQLExecDirectW 785
SQLExecute 783
SQLFetch 808
SQLFetchScroll 810
SQLForeignKeys, SQLForeignKeysW 833
SQLFreeHandle 865
SQLFreeStmt 858
SQLGetConnectAttr, SQLGetConnectAttrW 751
SQLGetCursorName, SQLGetCursorNameW 776
SQLGetData 812
SQLGetDescField, SQLGetDescFieldW 760
SQLGetDescRec, SQLGetDescRecW 762
SQLGetDiagField, SQLGetDiagFieldW 820
SQLGetDiagRec, SQLGetDiagRecW 822
SQLGetEnvAttr 754
SQLGetFunctions 744
SQLGetInfo, SQLGetInfoW 742
SQLGetStmtAttr, SQLGetStmtAttrW 757
SQLGetTypeInfo, SQLGetTypeInfoW 745
SQLMoreResults 819
SQLNativeSql, SQLNativeSqlW 789
SQLNumParams 781
SQLNumResultCols 799
SQLParamData 793
SQLPrepare, SQLPrepareW 771
SQLPrimaryKeys, SQLPrimaryKeysW 838
SQLProcedureColumns, SQLProcedureColumnsW 840
SQLProcedures, SQLProceduresW 842
SQLPutData 795
SQLRowCount 798

SQLSetConnectAttr, SQLSetConnectAttrW 749
SQLSetCursorName, SQLSetCursorNameW 778
SQLSetDescField, SQLSetDescFieldW 765
SQLSetDescRec 767
SQLSetEnvAttr 753
SQLSetPos 815
SQLSetStmtAttr, SQLSetStmtAttrW 756
SQLSpecialColumns, SQLSpecialColumnsW 844
SQLStatistics, SQLStatisticsW 847
SQLTablePrivileges, SQLTablePrivilegesW 851
SQLTables, SQLTablesW 854
SQLWarning interface 636
sqlwarningkeep
  URL for connection 268
  user property 273
statement handle
  allocating (a_rdb_SQLAllocStmt()) 970
  allocating (CLI function, example of) 944
  releasing (a_rdb_SQLFreeStmt()) 984
  releasing (CLI function, example of) 948
Statement interface 367
storesLowerCaseIdentifiers() 577
storesLowerCaseQuotedIdentifiers() 578
storesMixedCaseIdentifiers() 578
storesMixedCaseQuotedIdentifiers() 579
storesUpperCaseIdentifiers() 579
storesUpperCaseQuotedIdentifiers() 580
structure 1015
SUBQUERY 230
SUBQUERY HASH 230
SUBQUERY LOOP 230
subquery processing method
  hash execution 148, 154
  nested loops row value execution 154
  nested loops work table execution 153
  row value execution 147
  work table execution 146
  work table row value execution 148
subquery, how to process 146
supportsAlterTableWithAddColumn() 580
supportsAlterTableWithDropColumn() 581
supportsANSI92EntryLevelSQL() 581
supportsANSI92FullSQL() 582
supportsANSI92IntermediateSQL() 582
supportsBatchUpdates() 583
supportsCatalogsInDataManipulation() 583
supportsCatalogsInIndexDefinitions() 584
supportsCatalogsInPrivilegeDefinitions() 584
supportsCatalogsInProcedureCalls() 585
supportsCatalogsInTableDefinitions() 585
supportsColumnAliasing() 586
supportsConvert() 586
supportsConvert(int fromType, int toType) 587
supportsCoreSQLGrammar() 588
supportsCorrelatedSubqueries() 589
supportsDataDefinitionAndDataManipulationTransactions() 589
supportsDataManipulationTransactionsOnly() 590
supportsDifferentTableCorrelationNames() 590
supportsExpressionsInOrderBy() 591
supportsExtendedSQLGrammar() 591
supportsFullOuterJoins() 592
supportsGetGeneratedKeys() 592
supportsGroupBy() 593
supportsGroupByBeyondSelect() 593
supportsGroupByUnrelated() 594
supportsIntegrityEnhancementFacility() 594
supportsLikeEscapeClause() 595
supportsLimitedOuterJoins() 595
supportsMinimumSQLGrammar() 596
supportsMixedCaseIdentifiers() 596
supportsMixedCaseQuotedIdentifiers() 597
supportsMultipleOpenResults() 597
supportsMultipleResultSets() 598
supportsMultipleTransactions() 598
supportsNamedParameters() 599
supportsNonNullableColumns() 599
supportsOpenCursorsAcrossCommit() 600
supportsOpenCursorsAcrossRollback() 600
supportsOpenStatementsAcrossCommit() 601
supportsOpenStatementsAcrossRollback() 601
supportsOrderByUnrelated() 602
supportsOuterJoins() 602
supportsPositionedDelete() 603
supportsPositionedUpdate() 603
supportsRefCursors() 604
supportsResultSetConcurrency(int type, int concurrency) 604
supportsResultSetHoldability(int holdability) 605
supportsResultSetType(int type) 606
supportsSavepoints() 606
supportsSchemasInDataManipulation() 607
supportsSchemasInIndexDefinitions() 607
supportsSchemasInPrivilegeDefinitions() 608

supportsSchemasInProcedureCalls() 608
supportsSchemasInTableDefinitions() 609
supportsSelectForUpdate() 609
supportsStatementPooling() 610
supportsStoredFunctionsUsingCallSyntax() 610
supportsStoredProcedures() 611
supportsSubqueriesInComparisons() 611
supportsSubqueriesInExists() 612
supportsSubqueriesInIns() 612
supportsSubqueriesInQuantifieds() 613
supportsTableCorrelationNames() 613
supportsTransactionIsolationLevel(int level) 614
supportsTransactions() 615
supportsUnion() 615
supportsUnionAll() 616
surrogate pair 703
symbol conventions 12
symbolic literal 1012
system information for data source, acquiring 826

T

TABLE FUNCTION DERIVED TABLE 236
table joining methods 137
table scan 100
TABLE SCAN 242, 247
TABLE VALUE CONSTRUCTOR SCAN 246
TB meaning 15
text indexes used during execution of SQL statement 105
time change
  machine on which JDBC driver has been installed 71
  OS on client machine 97
TIME-type data
  converting (CLI function) 1006
  converting to (CLI function) 994
timeout
  HADB server connection processing 47
  HADB server processing request 47
timeout value, specifying
  DataSource interface 650, 655
  Statement interface 395
TIMESTAMP-type data
  converting (CLI function) 1007
  converting to (CLI function) 996
trace level 923
tracenum
  URL for connection 268
  user property 273
transaction access mode, specifying
  for CLI functions 963
  for JDBC driver 363
  for ODBC driver 751
transaction control (designing application program) 936
Transaction ID 256
transaction, terminating (CLI function) 968
tree row number 224
tree view 224
  information displayed in 230
troubleshooting (ODBC) 909
try-with-resources statement 697
tuning 223
types of work tables
  global work table 171
  local work table 171
TZ (environment variable) 82, 83
  if JDBC driver is used 61

U

UNION ALL 253
UNION DISTINCT 253
unsupported interface
  JDBC 1.2 API 637
  JDBC 2.1 Core API 644
  JDBC 3.0 API 685
  JDBC 4.0 API 695
unwrap(Class<T> iface) 691
update operation on retrieval using cursor, effect of 938
update SQL statement 640
UPDATE STATEMENT 230
updatesAreDetected(int type) 616
URL for connection 268
  encodelang 268
USE FILTER 252, 254
user
  URL for connection 268
  user property 273
user property
  encodelang 273
usesLocalFilePerTable() 617
usesLocalFiles() 617
USING CACHE 236
USING COST 243
using datetime information of archive range column to narrow search range 208

V

value 1012
VARBINARY-type data
  converting (CLI function) 1009
  converting to (CLI function) 998
Version 256
version number conventions 15
version upgrade (HADB client) 88

W

wasNull() 495
WINDOW 241
work table execution
  method for processing SELECT DISTINCT 169
  method for processing set operation 165
  methods for processing subqueries that do not contain external reference column 146
work table row value execution 148
WORK TABLE SCAN 244
work tables created when SQL statements are executed 172
Wrapper interface 690
wrapper pattern 688

Hitachi, Ltd.
6-6, Marunouchi 1-chome, Chiyoda-ku, Tokyo, 100-8280 Japan