HITACHI
Inspire the Next

Nonstop Database

# HiRDB Version 9

Description

3020-6-450-50(E)

## ■ Relevant program products

List of program products:

**For Red Hat Enterprise Linux 6 (64-bit x86_64) operating systems:**

P-9W62-4592 HiRDB Server Version 9 09-50

P-9W62-2D92 HiRDB/Run Time Version 9(64) 09-50

P-9S62-2B92 HiRDB/Run Time Version 9 09-50

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

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

**For Windows Server 2008 R2, Windows Server 2008 (x64), Windows Server 2012, Windows Vista Ultimate (x64), Windows Vista Business (x64), Windows Vista Enterprise (x64), Windows 7 Professional (x64), Windows 7 Enterprise (x64), Windows 7 Ultimate (x64), Windows 8 Pro (x64), Windows 8 Enterprise (x64), Windows 8.1 Pro (x64), and Windows 8.1 Enterprise (x64) operating systems:**

P-2962-9197 HiRDB Server Version 9 09-50

**For Windows Vista, Windows Server 2008, Windows Server 2012, Windows 7, Windows 8, and Windows 8.1 operating systems:**

P-2662-1197 HiRDB/Run Time Version 9 09-50

**For Windows Vista (x64), Windows Server 2008 R2, Windows Server 2008 (x64), Windows Server 2012, Windows 7 (x64), Windows 8 (Core Edition) (x64), Windows 8 Pro (x64), Windows 8 Enterprise (x64), Windows 8.1 (Core Edition) (x64), Windows 8.1 Pro (x64), and Windows 8.1 Enterprise (x64) operating systems:**

P-2962-1197 HiRDB/Run Time Version 9(64) 09-50

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

## ■ Trademarks

HITACHI, JP1, uCosminexus, HiRDB, DocumentBroker, HA Monitor are either trademarks or registered trademarks of Hitachi, Ltd. in Japan and other countries.

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

AMD is a trademark of Advanced Micro Devices, Inc.

IBM, AIX are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, AIX 5L are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, DataStage, MetaBroker, MetaStage and QualityStage are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, DB2 are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, HACMP are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, OS/390 are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

IBM, PowerHA are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

Itanium is a trademark of Intel Corporation in the United States and other countries.

Microsoft and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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

Microsoft Office and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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

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

ODBC is Microsoft's strategic interface for accessing databases.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

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

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

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

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

## ■ Restrictions

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

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

Printed in Japan.

## ■ Issued

Oct. 2015: 3020-6-450-50(E)

## ■ Copyright

# Preface

This manual provides an overview and description of the functions of *HiRDB Version 9*, a scalable database server program product.

## ■ Intended readers

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

Readers of this manual must have:

- A basic knowledge of managing UNIX or Linux systems (if you are using the UNIX edition)
- A basic knowledge of managing Windows systems (if you are using the Windows edition)
- A basic knowledge of SQL

## ■ Related publications

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

**HiRDB (for UNIX)**

- *HiRDB Version 9 Installation and Design Guide* (3000-6-452(E)), for UNIX systems
- *HiRDB Version 9 System Definition* (3000-6-453(E)), for UNIX systems
- *HiRDB Version 9 System Operation Guide* (3000-6-454(E)), for UNIX systems
- *HiRDB Version 9 Command Reference* (3000-6-455(E)), for UNIX systems
- *HiRDB Version 9 Staticizer Option Description and User's Guide* (3000-6-463), for UNIX systems[#1]
- *HiRDB Version 9 Disaster Recovery System Configuration and Operation Guide* (3000-6-464(E)), for UNIX systems
- *HiRDB Version 9 Batch Job Accelerator* (3020-6-468), for UNIX systems[#1]
- *HiRDB Version 9 Memory Database Installation and Operation Guide* (3020-6-469), for UNIX systems[#1]

**HiRDB (for Windows)**

- *HiRDB Version 9 Installation and Design Guide* (3020-6-452(E)), for Windows systems
- *HiRDB Version 9 System Definition* (3020-6-453(E))
- *HiRDB Version 9 System Operation Guide* (3020-6-454(E)), for Windows systems
- *HiRDB Version 9 Command Reference* (3020-6-455(E)), for Windows systems
- *HiRDB Version 8 Batch Job Accelerator* (3020-6-368)[#1]

**HiRDB (for both Windows and UNIX)**

- *HiRDB Version 9 UAP Development Guide* (3020-6-456(E))
- *HiRDB Version 9 SQL Reference* (3020-6-457(E))
- *HiRDB Version 9 Messages* (3020-6-458(E))
- *HiRDB Version 9 XDM/RD E2 Connection Facility* (3020-6-465)[#1]
- *HiRDB Version 9 XML Extension* (3020-6-480)[#1]
- *HiRDB Version 9 Text Search Plug-in* (3020-6-481)[#1]
- *HiRDB Version 8 Security Guide* (3020-6-359)[#1]
- *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide* (3020-6-360(E))
- *HiRDB Datareplicator Extension Version 8* (3020-6-361)[#1]
- *HiRDB Dataextractor Version 8 Description, User's Guide and Operator's Guide* (3020-6-362(E))

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

In addition, this manual refers to the Windows manual *HiRDB Version 8 Batch Job Accelerator* as *HiRDB Version 9 Batch Job Accelerator*. If you are using HiRDB for Windows, when you see references to the manual *HiRDB Version 9 Batch Job Accelerator*, read such references as *HiRDB Version 8 Batch Job Accelerator*.

**Others**

- *Job Management Partner 1/Performance Management - Agent Option for Platform* (3020-3-R48(E)), for Windows systems
- *Job Management Partner 1/Performance Management - Agent Option for Platform* (3020-3-R49(E)), for UNIX systems
- *Job Management Partner 1/Performance Management - Agent Option for HiRDB* (3020-3-R55)#1
- *JP1 V9 Job Management Partner 1/Base User's Guide* (3020-3-R71(E))
- *Job Management Partner 1/Integrated Management - Manager Overview and System Design Guide* (3020-3-R76(E))
- *JP1 Version 9 Job Management Partner 1/NETM/Audit Setup, Description and Operation Guide* (3020-3-S90)#1
- *Job Management Partner 1/Consolidated Management 2/Extensible SNMP Agent* (3020-3-T04(E)), for UNIX systems
- *Job Management Partner 1/Integrated Management - Manager System Configuration and User's Guide* (3020-3-K01(E))
- *Job Management Partner 1/Base User's Guide* (3020-3-K06(E))
- *Job Management Partner 1/Performance Management - Agent Option for Platform Description, User's Guide and Reference* (3020-3-K64(E)), for Windows systems
- *Job Management Partner 1/Performance Management - Agent Option for Platform Description, User's Guide and Reference* (3020-3-K65(E)), for UNIX systems
- *Job Management Partner 1/Performance Management - Agent Option for HiRDB* (3020-3-K70)#1
- *JP1 Version 8 JP1/Cm2/Extensible SNMP Agent* (3020-3-L04)#1
- *JP1 Version 8 Job Management Partner 1/NETM/Audit* (3020-3-L50)#1
- *Job Management Partner 1/Integrated Manager - Console* (3020-3-F01(E))
- *Job Management Partner 1/Base* (3020-3-F04(E))
- *Job Management Partner 1/Performance Management - Agent for HiRDB* (3020-3-F61(E))
- *Job Management Partner 1/Performance Management/SNMP System Observer for Extended Resource Management* (3020-3-F70(E))
- *JP1 V6 JP1/Base* (3020-3-986(E))
- *Job Management Partner 1/Performance Management - Agent for HiRDB* (3020-3-C68)#1
- *JP1 Version 6 JP1/VERITAS NetBackup v4.5 Agent for HiRDB License Description and User's Guide* (3020-3-D79(E))
- *Job Management Partner 1/System Event Service Version 5* (3000-3-154(E))
- *Job Management Partner 1/System Event Service* (3000-3-080(E))
- *EasyMT* (3000-3-575)#1
- *EasyMT* (3000-3-578)#1
- *EasyMT* (3000-3-573)#1
- *Cosminexus Application Setup Guide* (3020-3-M08(E))
- *uCosminexus Grid Processing Server User's Guide* (3000-3-E05)#1
- *OpenTP1 Version 7 System Definition* (3000-3-D52(E))
- *OpenTP1 Version 7 Programming Guide* (3000-3-D51(E))
- *OpenTP1 Version 7 Programming Reference COBOL Language* (3000-3-D55(E))
- *OpenTP1 Version 7 Programming Reference C Language* (3000-3-D54(E))
- *Open TP1 Version 7 Server Base Enterprise Option User's Guide* (3000-3-982)#1
- *TPBroker User's Guide* (3000-3-555(E))

- *For AIX Systems High-Availability System Monitoring Facility* (3000-9-130)#1, #2
- *For HP-UX Systems High-Availability System Monitoring Facility* (*3000-9-131*)#1, #2
- *For HP-UX (IPF) Systems High-Availability System Monitoring Facility* (3000-9-133)#1, #2
- *For Linux(R) Systems High-Availability System Monitoring Facility* (3000-9-132)#1, #2
- *For Linux(R) (x86) Systems HA Monitor Cluster Software* (*3000-9-140*(E)) #2
- *Hitachi HA Toolkit* (3000-9-115)#1
- *COBOL85 User's Guide* (3000-3-347)#1
- *COBOL85 Operations Guide* (3020-3-747(E))
- *DBPARTNER2 Client Operation Guide* (3020-6-027)#1
- *Hitachi Tuning Manager - Agent for RAID* (3020-3-P44)#1
- *Hitachi Tuning Manager - Storage Mapping Agent* (3020-3-P45)#1
- *VOS3 XDM/RD E2 Description* (6190-6-637)#1
- *VOS3 XDM/RD E2 SQL Reference* (6190-6-656)#1
- *VOS3 XDM E2 Messages* (XDM/RD E2) (6190-6-643)#1

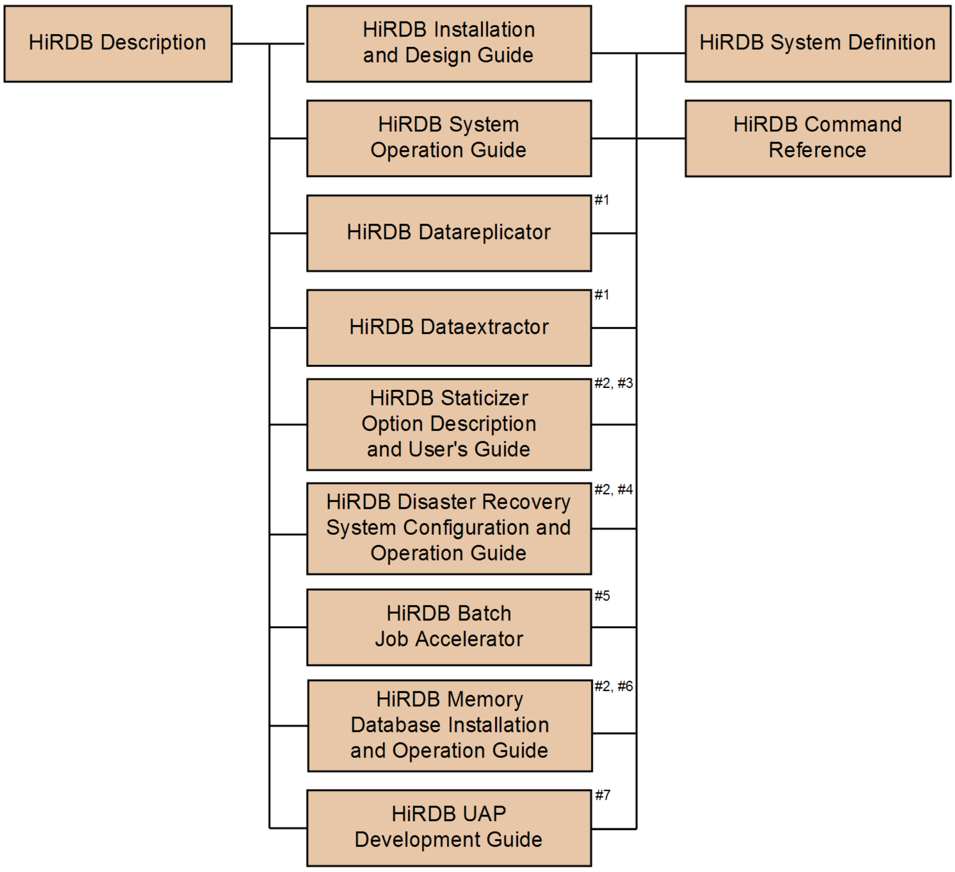
#1: This manual has been published in Japanese only; it is not available in English.

#2: In references to HA Monitor manuals, this manual omits the phrases *for AIX(R) systems*, *for HP-UX systems*, *for HP-UX (IPF) systems*, *for Linux(R) systems*, and *for Linux(R) (x86) systems*. Refer to the manual that is appropriate to the platform you are using.

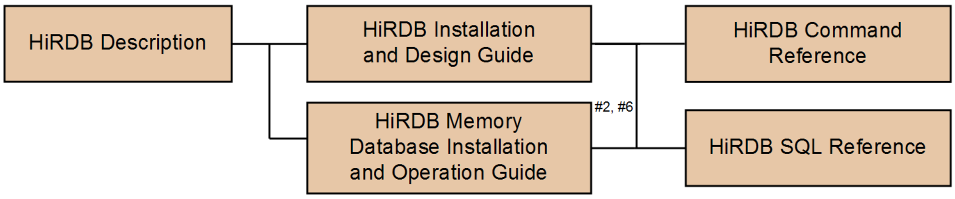
## ■ Organization of HiRDB manuals

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

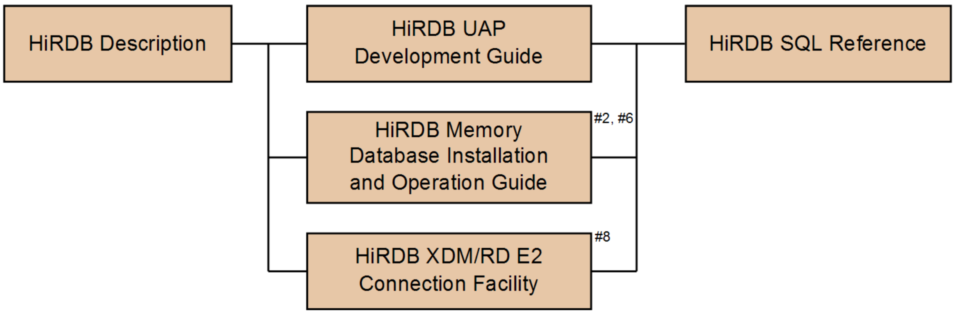
For system administrators:


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

For users who create tables:


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

For users who create or execute UAPs:


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

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

## ■ Conventions: Abbreviations for product names

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

| Full name or meaning | Abbreviation | |
|---|---|---|
| HiRDB Server Version 9 | HiRDB single server configuration | HiRDB or HiRDB Server |
| | HiRDB parallel server configuration | |
| HiRDB/Developer's Kit Version 9 | HiRDB/Developer's Kit | HiRDB Client |
| HiRDB/Developer's Kit Version 9(64) | | |
| HiRDB/Run Time Version 9 | HiRDB/Run Time | |
| HiRDB/Run Time Version 9(64) | | |
| HiRDB Accelerator Version 8 | HiRDB Accelerator | |
| HiRDB Accelerator Version 9 | | |
| HiRDB Adapter for XML - Standard Edition | HiRDB Adapter for XML | |
| HiRDB Adapter for XML - Enterprise Edition | | |
| HiRDB Advanced High Availability Version 9 | HiRDB Advanced High Availability | |
| HiRDB Control Manager | HiRDB CM | |
| HiRDB Control Manager Agent | HiRDB CM Agent | |
| HiRDB Dataextractor Version 8 | HiRDB Dataextractor | |
| HiRDB Datareplicator Version 8 | HiRDB Datareplicator | |
| HiRDB Disaster Recovery Light Edition Version 9 | HiRDB Disaster Recovery Light Edition | |
| HiRDB Non Recover Front End Server Version 9 | HiRDB Non Recover FES | |
| HiRDB Staticizer Option Version 9 | HiRDB Staticizer Option | |
| HiRDB Text Search Plug-in Version 9 | HiRDB Text Search Plug-in | |

| Full name or meaning | Abbreviation |
|---|---|
| HiRDB XML Extension Version 9 | HiRDB XML Extension |
| Single server | SDS |
| System manager | MGR |
| Front-end server | FES |
| Dictionary server | DS |
| Back-end server | BES |
| Microsoft(R) ActiveX(R) | ActiveX |
| DNCWARE ClusterPerfect (Linux Edition) | ClusterPerfect |
| DataStage(R) | DataStage |
| DB2 Universal Database for OS/390 Version 6 | DB2 |
| JP1/Magnetic Tape Access | EasyMT |
| EasyMT | |
| JP1/Automatic Job Management System 2 - Scenario Operation | JP1/AJS2-SO |
| JP1/Automatic Job Management System 3 | JP1/AJS3 |
| JP1/Automatic Job Management System 2 | |
| JP1/Cm2/Extensible SNMP Agent | JP1/ESA |
| JP1/Cm2/Extensible SNMP Agent for Mib Runtime | |
| JP1/Integrated Management - Manager | JP1/Integrated Management or JP1/IM |
| JP1/Integrated Management - View | |
| JP1/NETM/Audit - Manager | JP1/NETM/Audit |
| JP1/NETM/DM | JP1/NETM/DM |
| JP1/NETM/DM Manager | |
| JP1/Cm2/Network Node Manager | JP1/NNM |
| JP1/Performance Management | JP1/PFM |
| JP1/Performance Management - Agent Option for HiRDB | JP1/PFM-Agent for HiRDB |
| JP1/Performance Management - Agent Option for Platform | JP1/PFM-Agent for Platform |
| JP1/Performance Management/SNMP System Observer | JP1/SSO |
| JP1/VERITAS NetBackup BS V4.5 Agent for HiRDB License | JP1/VERITAS NetBackup Agent for HiRDB License |
| JP1/VERITAS NetBackup V4.5 Agent for HiRDB License | |
| JP1/VERITAS NetBackup 5 Agent for HiRDB License | |
| LifeKeeper for Linux V7 Update1 | LifeKeeper |
| MetaBroker(R) | MetaBroker |
| MetaStage(R) | MetaStage |
| Microsoft(R) Office Excel | Microsoft Excel or Excel |
| DLT(TM) | DLT |

| Full name or meaning | Abbreviation | |
|---|---|---|
| PowerBuilder$^{(R)}$ | PowerBuilder | |
| JP1/Magnetic Tape Library | MTguide | |
| JP1/VERITAS NetBackup BS v4.5 | NetBackup | |
| JP1/VERITAS NetBackup v4.5 | | |
| PowerHA for AIX, V5.5 | PowerHA | |
| PowerHA SystemMirror V6.1 | | |
| QualityStage(TM) | QualityStage | |
| OpenTP1/Server Base Enterprise Option | TP1/EE | |
| Hitachi System Information Capture | HSIC | |
| Hitachi TrueCopy | TrueCopy | |
| Hitachi TrueCopy Asynchronous | | |
| Hitachi TrueCopy basic | | |
| Hitachi TrueCopy Software | | |
| TrueCopy | | |
| TrueCopy Asynchronous | | |
| TrueCopy remote replicator | | |
| Hitachi Universal Replicator Software | Universal Replicator | |
| Universal Replicator | | |
| Microsoft$^{(R)}$ Visual C++$^{(R)}$ | Visual C++ or C++ language | |
| Oracle WebLogic Server | WebLogic Server | |
| Virtual-storage Operating System 3/Forefront System Product | VOS3/FS | VOS3 |
| Virtual-storage Operating System 3/Leading System Product | VOS3/LS | |
| Virtual-storage Operating System 3/Unific System Product | VOS3/US | |
| VOS3 Database Connection Server | DB Connection Server | |
| Extensible Data Manager/Base Extended Version 2<br>XDM Basic Program XDM/BASE E2 | XDM/BASE E2 | |
| XDM/Data Communication and Control Manager 3<br>XDM Data Communication Management System XDM/DCCM3 | XDM/DCCM3 | |
| XDM/Relational Database<br>Relational Database System XDM/RD | XDM/RD | XDM/RD |
| XDM/Relational Database Extended Version 2<br>Relational Database System XDM/RD E2 | XDM/RD E2 | |
| HP-UX 11i V2 (IPF) | HP-UX or HP-UX (IPF) | |
| HP-UX 11i V3 (IPF) | | |
| AIX 5L V5.2 | AIX 5L | AIX |
| AIX 5L V5.3 | | |

| Full name or meaning | Abbreviation | |
|---|---|---|
| AIX V6.1 | AIX V6.1 | |
| AIX V7.1 | AIX V7.1 | |
| Linux$^{(R)}$ | Linux | |
| Red Hat Enterprise Linux$^{(R)}$ AS 4(AMD64 & Intel EM64T) | Linux AS 4 | Linux |
| Red Hat Enterprise Linux$^{(R)}$ AS 4(x86) | | |
| Red Hat Enterprise Linux$^{(R)}$ ES 4(AMD64 & Intel EM64T) | Linux ES 4 | |
| Red Hat Enterprise Linux$^{(R)}$ ES 4(x86) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 Advanced Platform (x86) | Linux 5 | |
| Red Hat Enterprise Linux$^{(R)}$ 5 (x86) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 Advanced Platform (AMD/Intel 64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 (AMD/Intel 64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 6 (32-bit x86) | Linux 6 | |
| Red Hat Enterprise Linux$^{(R)}$ 6 (64-bit x86_64) | | |
| Red Hat Enterprise Linux$^{(R)}$ AS 4(AMD64 & Intel EM64T) | Linux (EM64T) | |
| Red Hat Enterprise Linux$^{(R)}$ ES 4(AMD64 & Intel EM64T) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 Advanced Platform (AMD/Intel 64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 (AMD/Intel 64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 6 (64-bit x86_64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 Advanced Platform (x86) | Linux 5 (x86) | Linux 5 |
| Red Hat Enterprise Linux$^{(R)}$ 5 (x86) | | |
| Red Hat Enterprise Linux$^{(R)}$ 5 Advanced Platform (AMD/Intel 64) | Linux 5 (AMD/Intel 64) | |
| Red Hat Enterprise Linux$^{(R)}$ 5 (AMD/Intel 64) | | |
| Red Hat Enterprise Linux$^{(R)}$ 6 (32-bit x86) | Linux 6 (32-bit x86) | Linux 6 |
| Red Hat Enterprise Linux$^{(R)}$ 6 (64-bit x86_64) | Linux 6 (64-bit x86_64) | |
| turbolinux 7 Server for AP8000 | Linux for AP8000 | |
| Microsoft$^{(R)}$ Windows NT$^{(R)}$ Workstation Operating System Version 4.0 | Windows NT | |
| Microsoft$^{(R)}$ Windows NT$^{(R)}$ Server Network Operating System Version 4.0 | | |
| Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard Edition | Windows Server 2003 Standard Edition | Windows Server 2003 |
| Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise Edition | Windows Server 2003 Enterprise Edition | |
| Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Standard x64 Edition | Windows Server 2003 Standard x64 Edition | |
| Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003, Enterprise x64 Edition | Windows Server 2003 Enterprise x64 Edition | |
| Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2003 R2, Standard Edition | Windows Server 2003 R2 | |

| Full name or meaning | Abbreviation | |
|---|---|---|
| Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition | | |
| Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition | | |
| Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition | | |
| Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition | Windows Server 2003 R2 x64 Editions | |
| Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition | | |
| Microsoft(R) Windows Server(R) 2008 Standard | Windows Server 2008 Standard | Windows Server 2008 |
| Microsoft(R) Windows Server(R) 2008 Enterprise | Windows Server 2008 Enterprise | |
| Microsoft(R) Windows Server(R) 2008 R2 Standard (x64) | Windows Server 2008 R2 | |
| Microsoft(R) Windows Server(R) 2008 R2 Enterprise (x64) | | |
| Microsoft(R) Windows Server(R) 2008 R2 Datacenter (x64) | | |
| Microsoft(R) Windows Server(R) 2008 Standard (x64) | Windows Server 2008 (x64) | |
| Microsoft(R) Windows Server(R) 2008 Enterprise (x64) | | |
| Microsoft(R) Windows Server(R) 2012 Standard | Windows Server 2012 Standard | Windows Server 2012 |
| Microsoft(R) Windows Server(R) 2012 Datacenter | Windows Server 2012 Datacenter | |
| Microsoft(R) Windows Server(R) 2012 R2 Standard | Windows Server 2012 R2 | |
| Microsoft(R) Windows Server(R) 2012 R2 Datacenter | | |
| Microsoft(R) Windows Server(R) 2003, Standard x64 Edition | Windows Server 2003 x64 Editions | Windows (x64) |
| Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition | | |
| Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition | | |
| Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition | | |
| Microsoft(R) Windows(R) XP Professional x64 Edition | Windows XP x64 Edition | |
| Microsoft(R) Windows Server(R) 2003, Enterprise Edition x64 Edition | Windows Server 2003 (IPF) | Windows(IPF) |
| Microsoft(R) Windows(R) XP Professional x64 Edition | Windows XP x64 Edition | Windows XP |
| Microsoft(R) Windows(R) XP Professional Operating System | Windows XP Professional | |
| Microsoft(R) Windows(R) XP Home Edition Operating System | Windows XP Home Edition | |
| Microsoft(R) Windows Vista(R) Home Basic | Windows Vista Home Basic | Windows Vista |
| Microsoft(R) Windows Vista(R) Home Premium | Windows Vista Home Premium | |
| Microsoft(R) Windows Vista(R) Ultimate | Windows Vista Ultimate | |

| Full name or meaning | Abbreviation | |
|---|---|---|
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business | Windows Vista Business | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise | Windows Vista Enterprise | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Home Basic (x64) | Windows Vista (x64) | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Home Premium (x64) | | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate (x64) | | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business (x64) | | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise (x64) | | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Ultimate (x64) | Windows Vista Ultimate (x64) | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Business (x64) | Windows Vista Business (x64) | |
| Microsoft$^{(R)}$ Windows Vista$^{(R)}$ Enterprise (x64) | Windows Vista Enterprise (x64) | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Home Premium | Windows 7 Home Premium | Windows 7 |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Professional | Windows 7 Professional | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Enterprise | Windows 7 Enterprise | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Ultimate | Windows 7 Ultimate | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Home Premium (x64) | Windows 7 (x64) | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Professional (x64) | | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Enterprise (x64) | | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Ultimate (x64) | | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Professional (x64) | Windows 7 Professional (x64) | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Enterprise (x64) | Windows 7 Enterprise (x64) | |
| Microsoft$^{(R)}$ Windows$^{(R)}$ 7 Ultimate (x64) | Windows 7 Ultimate (x64) | |
| Windows$^{(R)}$ 8 (Core Edition) | Windows 8 (Core Edition) | Windows 8 |
| Windows$^{(R)}$ 8 Pro | Windows 8 Pro | |
| Windows$^{(R)}$ 8 Enterprise | Windows 8 Enterprise | |
| Windows$^{(R)}$ 8 (Core Edition) (x64) | Windows 8 (Core Edition) (x64) | |
| Windows$^{(R)}$ 8 Pro (x64) | Windows 8 Pro (x64) | |
| Windows$^{(R)}$ 8 Enterprise (x64) | Windows 8 Enterprise (x64) | |
| Windows$^{(R)}$ 8.1 (Core Edition) | Windows 8.1 (Core Edition) | Windows 8.1 |

| Full name or meaning | Abbreviation | |
|---|---|---|
| Windows$^{(R)}$ 8.1 Pro | Windows 8.1 Pro | |
| Windows$^{(R)}$ 8.1 Enterprise | Windows 8.1 Enterprise | |
| Windows$^{(R)}$ 8.1 (Core Edition) (x64) | Windows 8.1 (Core Edition) (x64) | |
| Windows$^{(R)}$ 8.1 Pro (x64) | Windows 8.1 Pro (x64) | |
| Windows$^{(R)}$ 8.1 Enterprise (x64) | Windows 8.1 Enterprise (x64) | |

- Windows Server 2008 and Windows Server 2012 are often referred to generically as *Windows Server*. In addition, *Windows* is often used generically to refer to Windows Server, Windows Vista, Windows 7, Windows 8, and Windows 8.1.
- *Windows 8* is used to refer to both Windows 8 and Windows 8.1, except in cases where it is necessary to distinguish between Windows 8 and Windows 8.1.
- *UNIX* is often used generically to refer to HP-UX, AIX, and Linux.
- In UNIX, the HiRDB directory path is represented as `$PDDIR`.
- In Windows, the HiRDB directory path is represented as `%PDDIR%`. The path of the Windows installation directory is represented as `%windir%`.
- The hosts file indicates the hosts file stipulated by TCP/IP (including the `/etc/hosts` file in UNIX). In Windows, a reference to the hosts file indicates the `%windir%\system32\drivers\etc\hosts` file.
- Block special files, which can be used in Linux, are handled the same way as character special files. If you are using a block special file, note that references to character special files in this manual also include block special files.

This manual also uses the following acronyms:

| Acronym | Full name or meaning |
|---|---|
| ACK | Acknowledgement |
| ACL | Access Control List |
| ADM | Adaptable Data Manager |
| ADO | ActiveX Data Objects |
| ADT | Abstract Data Type |
| AP | Application Program |
| API | Application Programming Interface |
| ASN.1 | Abstract Syntax Notation One |
| BES | Back End Server |
| BLOB | Binary Large Object |
| BOM | Byte Order Mark |
| CD-ROM | Compact Disc - Read Only Memory |
| CGI | Common Gateway Interface |
| CLOB | Character Large Object |
| CMT | Cassette Magnetic Tape |
| COBOL | Common Business Oriented Language |
| CORBA$^{(R)}$ | Common ORB Architecture |

| Acronym | Full name or meaning |
|---|---|
| CPU | Central Processing Unit |
| CSV | Comma Separated Values |
| DAO | Data Access Object |
| DAT | Digital Audio Taperecorder |
| DB | Database |
| DBM | Database Module |
| DBMS | Database Management System |
| DDL | Data Definition Language |
| DF for Windows NT | Distributing Facility for Windows NT |
| DF/UX | Distributing Facility/for UNIX |
| DIC | Dictionary Server |
| DLT | Digital Linear Tape |
| DML | Data Manipulate Language |
| DNS | Domain Name System |
| DOM | Document Object Model |
| DS | Dictionary Server |
| DTD | Document Type Definition |
| DTP | Distributed Transaction Processing |
| DWH | Data Warehouse |
| EBCDIK | Extended Binary Coded Decimal Interchange Kana |
| EUC | Extended UNIX Code |
| EX | Exclusive |
| FAT | File Allocation Table |
| FD | Floppy Disk |
| FES | Front End Server |
| FQDN | Fully Qualified Domain Name |
| FTP | File Transfer Protocol |
| GUI | Graphical User Interface |
| HBA | Host Bus Adapter |
| HD | Hard Disk |
| HDP | Hitachi Dynamic Provisioning |
| HTML | Hyper Text Markup Language |
| ID | Identification number |
| IP | Internet Protocol |
| IPF | Itanium$^{(R)}$ Processor Family |
| JAR | Java Archive File |

| Acronym | Full name or meaning |
|---|---|
| Java VM | Java Virtual Machine |
| JDBC | Java Database Connectivity |
| JDK | Java Developer's Kit |
| JFS | Journaled File System |
| JFS2 | Enhanced Journaled File System |
| JIS | Japanese Industrial Standard code |
| JP1 | Job Management Partner 1 |
| JRE | Java Runtime Environment |
| JTA | Java Transaction API |
| JTS | Java Transaction Service |
| KEIS | Kanji processing Extended Information System |
| LAN | Local Area Network |
| LDAP | Lightweight Directory Access Protocol |
| LIP | loop initialization process |
| LOB | Large Object |
| LRU | Least Recently Used |
| LTO | Linear Tape-Open |
| LU | Logical Unit |
| LUN | Logical Unit Number |
| LVM | Logical Volume Manager |
| MGR | System Manager |
| MIB | Management Information Base |
| MRCF | Multiple RAID Coupling Feature |
| MSCS | Microsoft Cluster Server |
| NAFO | Network Adapter Fail Over |
| NAPT | Network Address Port Translation |
| NAT | Network Address Translation |
| NIC | Network Interface Card |
| NIS | Network Information Service |
| NTFS | New Technology File System |
| ODBC | Open Database Connectivity |
| OLAP | Online Analytical Processing |
| OLE | Object Linking and Embedding |
| OLTP | On-Line Transaction Processing |
| OOCOBOL | Object Oriented COBOL |
| ORB | Object Request Broker |

| Acronym | Full name or meaning |
|---|---|
| OS | Operating System |
| OSI | Open Systems Interconnection |
| OTS | Object Transaction Service |
| PC | Personal Computer |
| PDM II E2 | Practical Data Manager II Extended Version 2 |
| PIC | Plug-in Code |
| PNM | Public Network Management |
| POSIX | Portable Operating System Interface for UNIX |
| PP | Program Product |
| PR | Protected Retrieve |
| PU | Protected Update |
| RAID | Redundant Arrays of Inexpensive Disk |
| RD | Relational Database |
| RDB | Relational Database |
| RDB1 | Relational Database Manager 1 |
| RDB1 E2 | Relational Database Manager 1 Extended Version 2 |
| RDO | Remote Data Objects |
| ReFS | Resilient File System |
| RiSe | Real time SAN replication |
| RM | Resource Manager |
| RMM | Resource Manager Monitor |
| RPC | Remote Procedure Call |
| SAX | Simple API for XML |
| SDS | Single Database Server |
| SGML | Standard Generalized Markup Language |
| SJIS | Shift JIS |
| SNMP | Simple Network Management Protocol |
| SQL | Structured Query Language |
| SQL/K | Structured Query Language / VOS K |
| SR | Shared Retrieve |
| SU | Shared Update |
| TCP/IP | Transmission Control Protocol / Internet Protocol |
| TM | Transaction Manager |
| TMS-4V/SP | Transaction Management System - 4V / System Product |
| UAP | User Application Program |
| UOC | User Own Coding |

| Acronym | Full name or meaning |
|---|---|
| VOS K | Virtual-storage Operating System Kindness |
| VOS1 | Virtual-storage Operating System 1 |
| VOS3 | Virtual-storage Operating System 3 |
| WS | Workstation |
| WWW | World Wide Web |
| XDM/BASE E2 | Extensible Data Manager / Base Extended Version 2 |
| XDM/DF | Extensible Data Manager / Distributing Facility |
| XDM/DS | Extensible Data Manager / Data Spreader |
| XDM/RD E2 | Extensible Data Manager / Relational Database Extended Version 2 |
| XDM/SD E2 | Extensible Data Manager / Structured Database Extended Version 2 |
| XDM/XT | Extensible Data Manager / Data Extract |
| XFIT | Extended File Transmission program |
| XML | Extensible Markup Language |

## ■ Log representations

### ■ UNIX edition

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

| OS | File |
|---|---|
| HP-UX | `/var/adm/syslog/syslog.log` |
| Solaris | `/var/adm/messages` or `/var/log/syslog` |
| AIX 5L | `/var/adm/ras/syslog` |
| Linux | `/var/log/messages` |

### ■ Windows edition

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

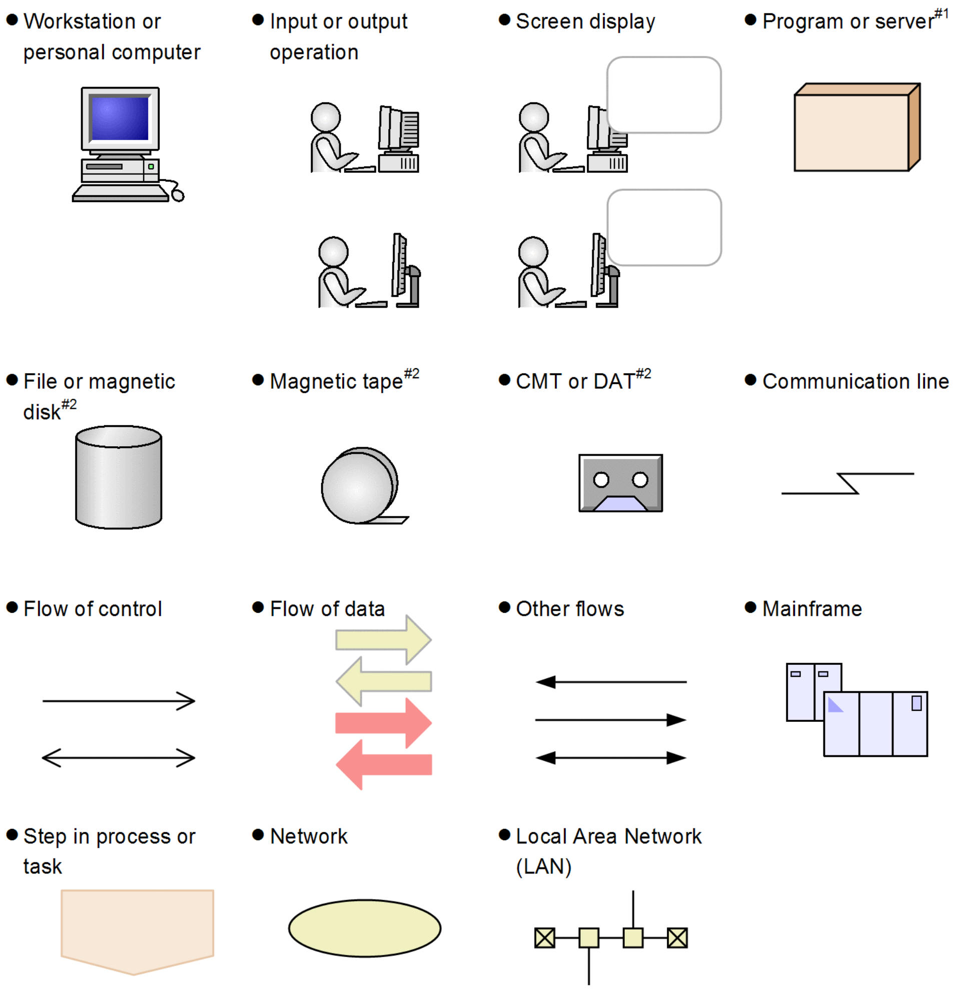
To view the event log:

1. Choose **Start**, **Programs**, **Administrative Tools (Common)**, and then **Event Viewer**.
2. Choose **Log**, and then **Application**.
3. The application log is displayed. Messages with **HiRDBSingleServer** or **HiRDBParallelServer** displayed in the **Source** column were issued by HiRDB.

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

## ■ Conventions: Diagrams

This manual uses the following conventions in diagrams:



#1: In some figures, a program is enclosed in a simple rectangle (without the shading).

#2: Input data files, unload files, and backup files can be stored on magnetic cassette tape (CMT) and digital audio tape (DAT), as well as on magnetic disk; only magnetic disk storage is described in this manual.

## ■ 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:<br>• From the **File** menu, choose **Open**.<br>• Click the **Cancel** button.<br>• 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:<br>• Write the command as follows:<br>`copy` *source-file target-file*<br>• The following message appears:<br>`A file was not found. (file =` *file-name*`)` |

| Font | Convention |
|---|---|
|  | *Italics* are also used for emphasis. For example:<br>• 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:<br>• At the prompt, enter `dir`.<br>• Use the `send` command to send mail.<br>• The following message is displayed:<br>`The password is incorrect.` |

The following table explains the symbols used in this manual:

| Symbol | Convention |
|---|---|
| \| | In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example:<br>`A\|B\|C` means `A`, or `B`, or `C`. |
| `[ ]` | In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example:<br>`[A]` means that you can specify `A` or nothing.<br>`[B\|C]` means that you can specify `B`, or `C`, or nothing. |
| `...` | In coding, an ellipsis (`...`) indicates that one or more lines of coding are not shown for purposes of brevity.<br>In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example:<br>`A, B, B, ...` means that, after you specify `A, B,` you can specify `B` as many times as necessary. |
| `()` | Parentheses indicate the range of items to which the vertical bar (`\|`) or ellipsis (`...`) is applicable. |

The following notations are used in formulas:

| Notation | Explanation |
|---|---|
| ↑ ↑ | Round up the result to the next integer.<br>Example: The result of ↑34 ÷ 3↑ is 12. |
| ↓ ↓ | Discard digits following the decimal point.<br>Example: The result of ↓34 ÷ 3↓ is 11. |
| MAX | Select the largest value as the result.<br>Example: The result of MAX(3 × 6, 4 + 7) is 18. |
| MIN | Select the smallest value as the result.<br>Example: The result of MIN(3 × 6, 4 + 7) is 11. |
| mod | mod(*a*, *b*) indicates the remainder of *a* divided by *b*.<br>Example: The result of MOD(9, 2) is 1. |

## ■ Notes on Windows path names

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

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

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

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

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

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

- HiRDB cannot use files on a networked drive, so you must install HiRDB and configure the HiRDB environment on a local drive. Files used by utilities, such as utility input and output files, must also be on the local drive.

## ■ Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

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

## ■ Conventions: Version numbers

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

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

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

## ■ Default values for omitted system definition operands in HiRDB version 09-50 and later

With respect to the default values of system definition operands, starting with HiRDB version 09-50, it is now possible to select recommended mode, which assumes the recommended value when no value is specified, or compatibility mode, which assumes the default value for a particular version when no value is specified. In general, we suggest that recommended mode be used. In this manual, the default operand values assumed by recommended mode are indicated with underlining.

## ■ Important notes on this manual

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

- Distributed database facility
- Server mode system switchover facility#
- User server hot standby#
- Rapid system switchover facility#
- Standby-less system switchover (1:1) facility
- Standby-less system switchover (effects distributed) facility

- HiRDB External Data Access facility
- Inner replica facility
- Updatable online reorganization
- Sun Java System Directory Server linkage facility
- Simple setup tool
- Extended syslog facility
- Rapid batch facility
- Memory database facility
- Linkage with JP1/NETM/Audit

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

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

#: Supported in the UNIX edition. Not supported in the Windows edition.

# Contents

# 3 Database Logical Structure 45

# 4 Database Physical Structure 103

# Appendixes

# Index

# 1 Overview

This chapter explains the characteristics, system configuration, and access modes of HiRDB. This chapter also provides an overview of HiRDB option program products and other HiRDB related products.

# 1.1 Characteristics of HiRDB

HiRDB is a database management system (DBMS) that enables you to construct relational databases appropriate to the scale of your operations.

Interlinking enables HiRDB to connect independently-operating server machines. HiRDB offers a high degree of flexibility, thanks to the adoption of the Shared Nothing method, in which a processor works exclusively on a database residing on a single disk. HiRDB can be adapted to a broad range of architectures, from a single-node configuration in which HiRDB runs on a single server machine, to a parallel processor configuration in which HiRDB runs on multiple server machines. HiRDB also makes it possible to expand your database system by adding server machines subsequently, which enables scalable system construction.

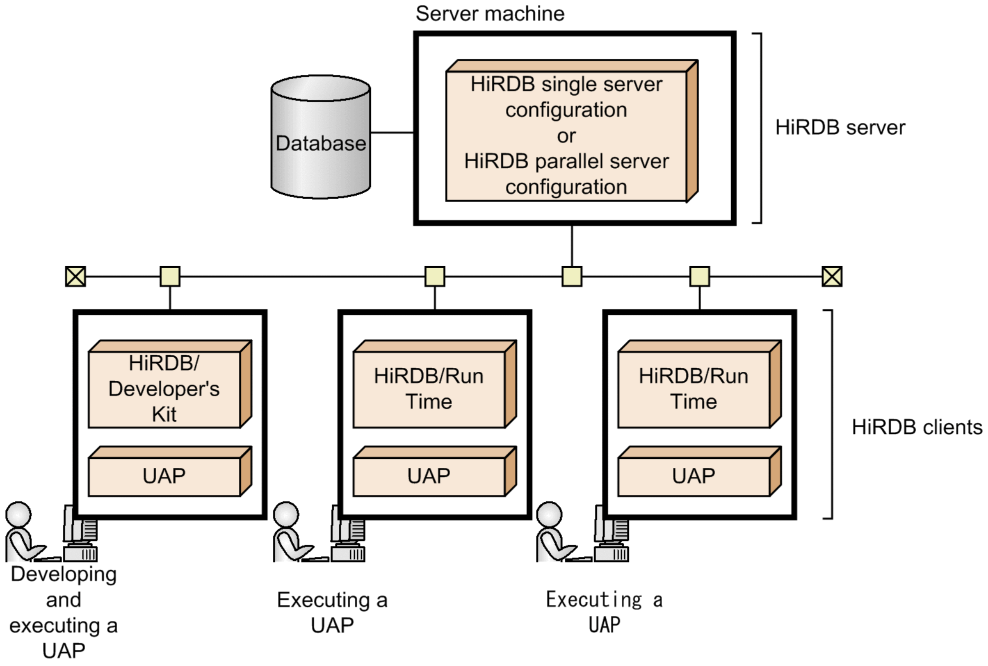
When HiRDB is configured for parallel processors, it is capable of executing data retrievals and update requests internally and in parallel, thus achieving high throughput and fast turnaround time.

In addition, when it is used in conjunction with a middleware application suite (such as DataStage), HiRDB supports the building of data warehouses, which are needed to implement customer-driven management systems.

## 1.1.1 Overview of HiRDB systems

HiRDB is used in a network environment consisting of client/server systems. A server system in which a database is installed is called a *HiRDB server*; a client system in which UAPs are developed and executed is called a *HiRDB client*. HiRDB servers and HiRDB clients are referred to as *HiRDB systems*. The following figure shows the configuration of a HiRDB system.

Figure 1–1: Configuration of a HiRDB system



### (1) HiRDB servers

The two types of HiRDB servers are HiRDB single server configurations and HiRDB parallel server configurations. You must select one of these two types, as appropriate to your system mode or the types of operations you will be performing.

A HiRDB server runs on one of the following platforms:

- HP-UX
- Solaris

- AIX
- Linux
- Windows XP
- Windows Server
- Windows Vista
- Windows 7
- Windows 8

#### (a) HiRDB single server configuration

You would use a HiRDB single server configuration for a database system that consists of a single server machine. Because of its stable processing performance and simple operations as compared with a HiRDB parallel server configuration, a HiRDB single server configuration is well-suited for small and medium-sized databases.

#### (b) HiRDB parallel server configuration

A HiRDB parallel server configuration makes it possible to link multiple server machines into a single database system so that a table can be split up and stored on multiple server machines. Because it permits different servers to perform retrieval tasks in parallel, a HiRDB parallel server configuration provides improved performance. In addition, its capability to add or update large amounts of data or back up a database concurrently means that a HiRDB parallel server configuration is able to maintain high performance even when the database becomes very large.

The Shared Nothing method, which fully exploits the hardware's capabilities, enables you to maintain stable performance by increasing the number of servers when the volume of data to be handled increases. In addition, a HiRDB parallel server configuration can achieve well-balanced parallel processing in a manner that prevents concentration of processing on a specific server by delegating sorting, joining, and similar large-overhead tasks to other servers that have smaller workloads.

### (2) HiRDB clients

The two types of HiRDB clients are the HiRDB/Developer's Kit and HiRDB/Run Time.

A HiRDB/client runs on one of the following platforms:

- HP-UX
- Solaris
- AIX
- Linux
- Linux for AP8000
- VOS3
- Windows XP
- Windows Server
- Windows Vista
- Windows 7
- Windows 8

#### (a) HiRDB/Developer's Kit

HiRDB/Developer's Kit is a program for developing (pre-processing, compiling, linking) and executing UAPs. As the development language for UAPs, you can use C, C++, COBOL85, OOCOBOL, COBOL 2002, or Java (SQLJ).

Because a HiRDB server includes the HiRDB/Developer's Kit functions, a HiRDB/Developer's Kit is not necessary for developing or executing a UAP on a HiRDB server; a HiRDB/Developer's Kit is necessary, however, to develop or execute a UAP on a client.

Tip

Execute the UAP with a HiRDB/Developer's Kit that is running on the same platform as the HiRDB/Developer's Kit that was used to develop the UAP.

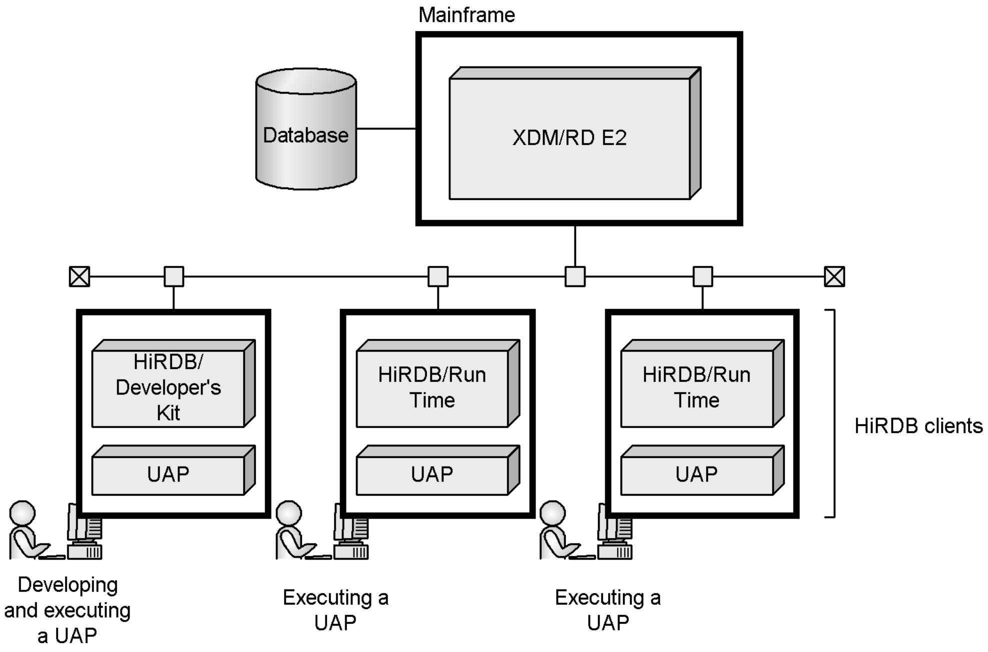
### (b) HiRDB/Run Time

HiRDB/Run Time is a run-time program that is used exclusively for executing previously created UAPs. HiRDB/Run Time cannot be used to develop (preprocess, compile, link) a UAP; it can only be used to execute a UAP.

### (c) Connecting from a HiRDB client to XDM/RD E2

At a HiRDB client, you can develop and execute UAPs that access XDM/RD E2 databases. Such UAPs can be used to access XDM/RD E2 databases from HiRDB clients. The function that performs this type of access is called the XDM/RD E2 connection facility. The following figure provides an overview of the XDM/RD E2 connection facility.

Figure 1–2: Overview of the XDM/RD E2 connection facility



**Explanation**

Using the XDM/RD E2 connection facility, you can directly access XDM/RD E2 databases from UAPs that are running on HiRDB clients.

Although ODBC or other functions can also be used to access XDM/RD E2 from a UAP running on a PC, there are limitations imposed by the application language. Using the XDM/RD E2 connection facility enables you to perform a wider variety of processing by coding SQL statements directly in the UAP, which also improves UAP development efficiency.

## 1.1.2 Advantages of using HiRDB

HiRDB is a relational database that incorporates both concurrent batch updating and parallel recovery techniques. The characteristics of HiRDB are discussed below.

### (1) Excellent scalability

The two types of HiRDB are the HiRDB single server configuration that operates on a single server machine, and the HiRDB parallel server configuration that operates on two or more server machines. You can select either a HiRDB single server configuration or a HiRDB parallel server configuration, depending on the scope of the operations to which HiRDB will be applied. You can change a HiRDB single server configuration into a HiRDB parallel server configuration, or you can increase the number of server machines used in a HiRDB parallel server configuration. Thus, you can expand your system gradually as the scale of your operations increases.

The Shared Nothing method that is incorporated into HiRDB is well-suited for parallel processing and enables you to increase the processing capacity of HiRDB in proportion to the number of processors that are used. In other words, you can increase processing capacity without having to add server machines. When the flexible hash partitioning function is used and a server machine is added, HiRDB modifies the hash function automatically, so that the data is stored automatically in the new server machine.

## (2) Achieving a high-performance system

### (a) Performance improvement through parallel processing

**Concurrent retrieval and updating of tables**

Database search and update operations can be distributed to multiple server machines, and table data can also be divided among the server machines. This feature permits concurrent searching and updating of tables, which improves performance in direct proportion to the number of server machines deployed.

**Distribution of high-overhead database processing operations**

HiRDB enables you to assign high-overhead operations, such as sorting and joining, to a separate server machine so that these operations can be performed concurrently with data accesses. This feature reduces the amount of time required to output search results.

**Reduction in batch processing time for large amounts of data**

HiRDB can reduce the amount of time required for processing because it can store masses of data concurrently, such as during system creation.

**Concurrent database reorganization**

HiRDB can reduce the amount of time required for database reorganization operations because it can concurrently reorganize databases by server.

**Concurrent backup and recovery operations**

In HiRDB, a single command can concurrently back up and recover a failed database. This feature reduces the amount of time required for backup and database recovery operations.

### (b) Fine buffer control through the use of global buffers

HiRDB enables you to allocate index data and other frequently accessed data to special buffers. This feature assures stable response by eliminating interference between buffers even in an environment where different operations are performed on a mixed basis, such as index searches and complete data searches.

### (c) Performance improvement by using synchronization point dump processing

In an operation called *synchronization point dump processing*, HiRDB stores update information up to a specific point in the database and recovery information up to that point in a file. Other systems stop accepting transactions during a synchronization point dump operation, which results in a decrease in processing performance. HiRDB, by contrast, does not limit the system's ability to accept transactions, so there is no decrease in processing performance attributable to synchronization point dump processing.

### (d) Rapid system recovery by high-speed rerun

HiRDB reduces the range over which a recovery operation must be performed in the aftermath of a system failure by periodically collecting synchronization point dumps, which enables HiRDB to quickly complete recovery processing.

In addition, HiRDB performs during a system recovery operation a *delayed rerun operation* that simultaneously starts a rollback operation and acceptance of new transactions, which can speed up system restart.

## (3) Achieving a highly reliable system

### (a) Storing in files information needed for system restart and for database recovery

**Logging information needing for system restart**

During operation, HiRDB stores in files system status information that will be needed to restart the system in the event of a failure. These files are called *status files*.

**Logging information needed for database recovery**

HiRDB stores in files a history of database update information (a system log) that would be needed for recovery of the database during a recovery operation. The files in which this information is stored are called *system log files*. In the event of an error in the system, system log files make it possible for HiRDB to correctly recover the database to its status just before the error occurred.

**Duplicate files**

HiRDB maintains duplicate status files and system log files, thus providing redundant logging of the information it will need for system and database recovery. File redundancy permits the use of the other file if a problem arises in one of the files, thus increasing system reliability.

### (b) Automatic system restart

In the event of a relatively minor error, HiRDB uses the status files to restart the system automatically, so there is no need for operator intervention.

### (c) Reducing system downtime by use of system switchover facility

In HiRDB, you can set up a standby server machine that is separate from the currently operating server machine, so that if the current server machine fails, operations can be switched smoothly to the standby machine. This feature is called the *system switchover facility*.

Microsoft Cluster Server (MSCS) is used for the system switchover facility.

## (4) Improved maintainability/operability

### (a) Intensive control from a specific server machine

In the case of a HiRDB parallel server configuration, a specified server machine can integrally and intensively control a HiRDB system running on multiple server machines. For example, you can execute a command or a utility on one server machine to start or terminate the HiRDB running on all server machines or on specified server machines.

### (b) Support for setting up the HiRDB environment

Support tools are provided for setting up the HiRDB environment. The tools that are provided are listed and described below.

Simple setup tool

A graphic user interface (GUI) is provided for setting up HiRDB. Choosing **Standard Setup**, in which you simply specify the HiRDB directory and setup directory, enables you to easily configure an environment. You can also choose **Custom Setup** in order to specify more detailed settings. The simple setup tool can also be used to update or edit system definitions that you have already created.

**Batch files** (Windows edition only)

By executing the batch files, you can automatically set up the basic HiRDB environment.

In normal cases, you can use the simple setup tool to set up the HiRDB environment.

## (5) Implementing a flexible system in an open environment

### (a) XA interface for X/Open

HiRDB can communicate with OLTP by means of the XA interface of X/Open. A HiRDB XA library is provided so that HiRDB transactions can be controlled by the Transaction Manager.

### (b) ODBC, JDBC, and OLE DB interfaces

HiRDB complies with the ODBC, JDBC, and OLE DB industry standards, which means that ODBC, JDBC, and OLE DB applications can be used in HiRDB. You can also use ADO (ADO.NET is also supported), DAO, and RDO.

## (6) Support for non-stop service

With the growing popularity of Internet businesses, there is an increasing need to conduct online operations non-stop, 24 hours per day, 365 days per year. HiRDB provides functionality based on the premise of continuous operation, 24 hours per day. For details about functionality based on 24-hour-per-day operation, see the *HiRDB Version 9 System Operation Guide*.

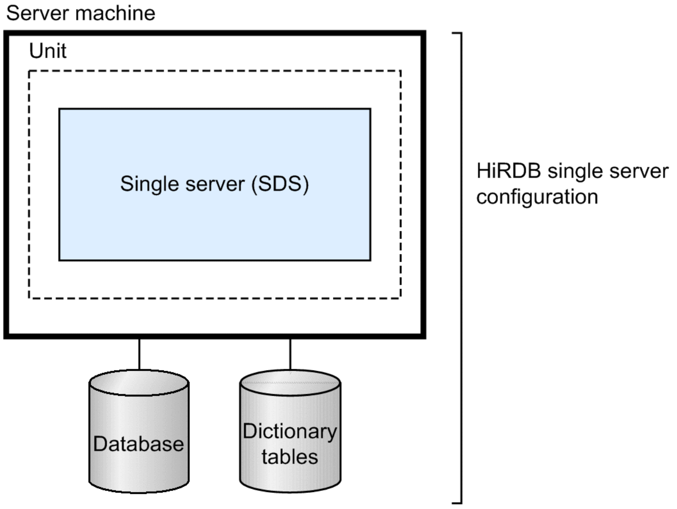
# 1.2 HiRDB system configuration

This section explains the configurations of a *HiRDB single server configuration*, a *HiRDB parallel server configuration*, and a *multi-HiRDB*.

## 1.2.1 HiRDB single server configuration

A HiRDB single server configuration consists of one unit (one single server). The following figure shows the configuration of a HiRDB single server configuration.

Figure 1–3: HiRDB single server configuration



### (1) Unit

A HiRDB single server configuration is composed of the following server:

- Single server

The unit controls and monitors execution of the server. The unit can be compared conceptually to a container in which the server is stored.

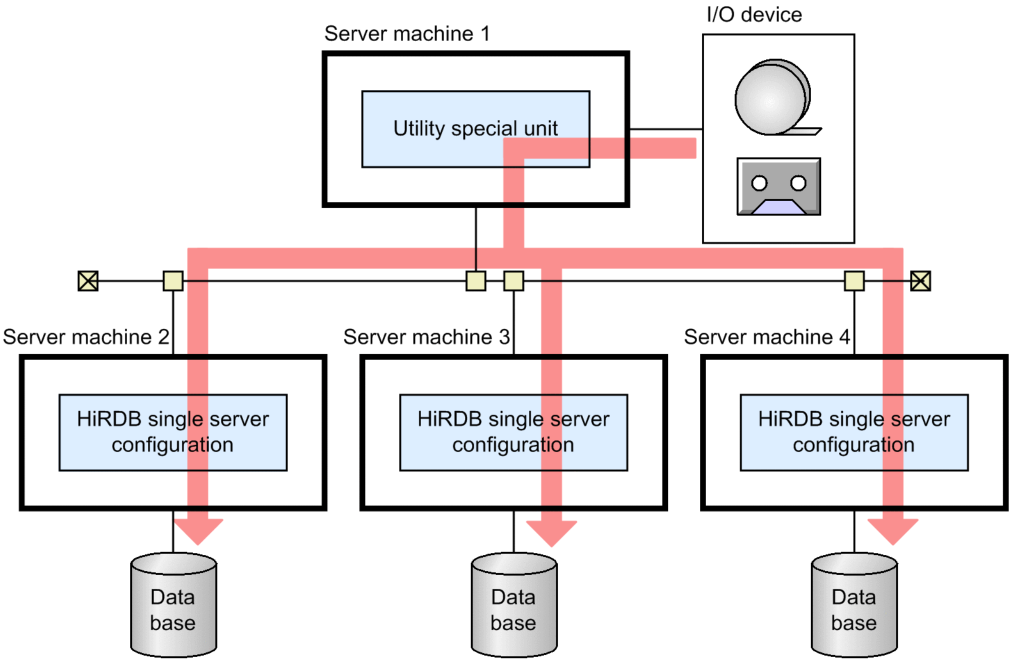
### (2) Single server (SDS: Single Database Server)

The single server is the server that manages a database (tables and indexes) and the data dictionaries (dictionary tables) that contain information about the database.

### (3) Utility special unit (UNIX edition only)

If the server machine on which a HiRDB single server configuration is implemented does not have I/O devices, such as CMT or magnetic tape drive, a server machine equipped with an I/O device can be used as a utility special unit. For example, when multiple HiRDB single server configurations are used and there is only a limited number of server machines equipped with I/O devices, a utility special unit can be set up. The following figure shows an example of a HiRDB single server configuration in which a utility special unit is set up.

Figure 1–4: Example of a HiRDB single server configuration (with a utility special unit)



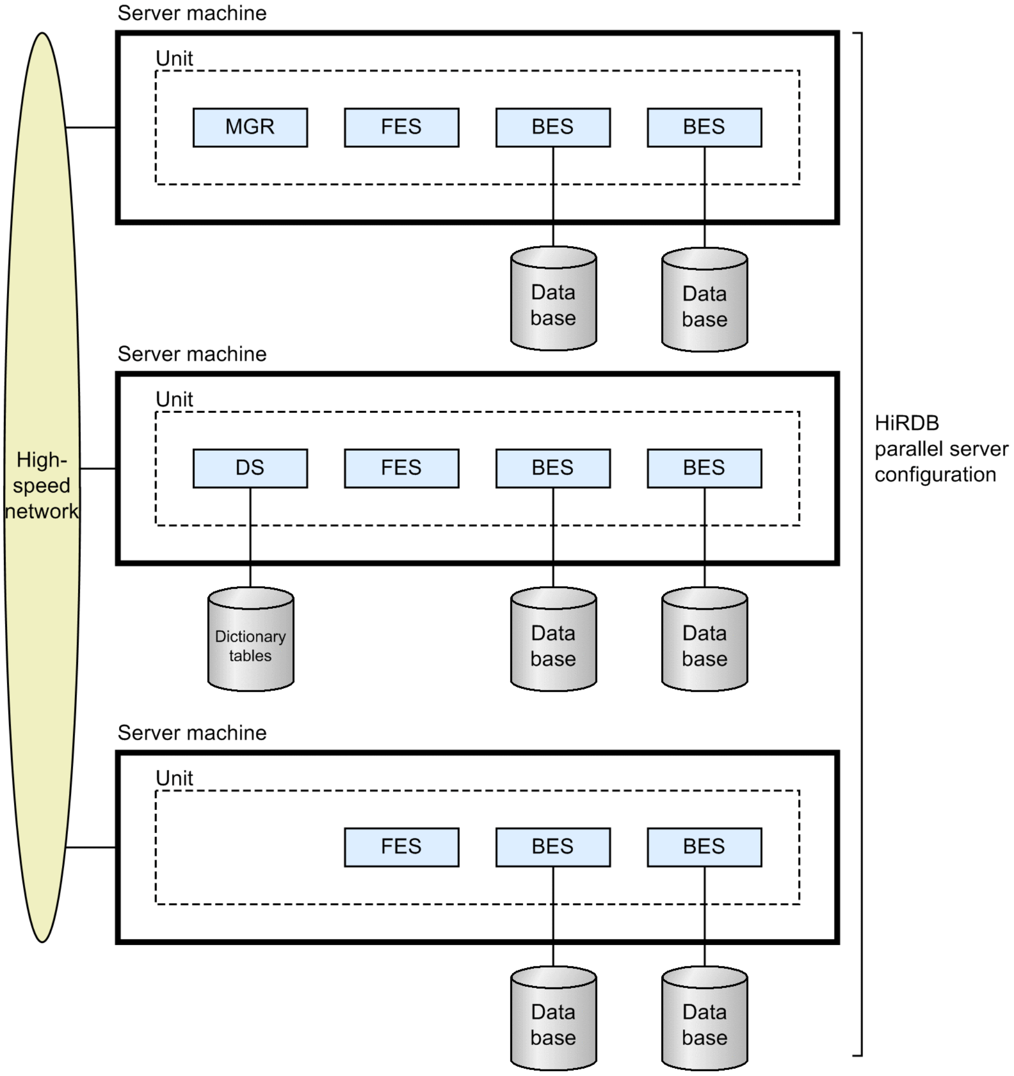
**Explanation**

- Server machines 2-4 are not equipped with I/O devices (such as CMT or magnetic tape drive). Because server machine 1 contains an I/O device, it is designated as a utility special unit.
- When source data on CMT or a magnetic tape is to be stored in the database, the I/O device attached to server machine 1 (utility special unit) is used.

## 1.2.2 HiRDB parallel server configuration

A HiRDB parallel server configuration is composed of multiple units (multiple servers). The following figure shows the configuration of a HiRDB parallel server configuration.

Figure 1–5: HiRDB parallel server configuration



**Explanation**

- This HiRDB parallel server configuration consists of three server machines.
- In the configuration in this example there are multiple front-end servers.
- Two back-end servers are provided per server machine.

## (1) Unit

A HiRDB parallel server configuration consists of the following types of servers:

- System manager
- Front-end server
- Dictionary server
- Back-end server

A unit controls and monitors server execution and manages communication between servers. A unit can be compared conceptually to a container in which servers are stored.

### (2) System manager (MGR)

The system manager is the server that controls HiRDB startup and termination. It also manages system configuration information and detects server errors.

One system manager is required per system.

### (3) Front-end server (FES)

A front-end server determines the procedure by which databases are accessed and provides directives to back-end servers on the contents of tasks that are to be executed. A front-end server also analyzes SQLs, optimizes SQLs, provides processing instructions to back-end servers, and edits search results.

Each system must have at least one front-end server (up to a maximum of 1,024 front-end servers). A configuration in which there are multiple front-end servers is called a *multi-front-end server configuration*. When SQL processing results in a high CPU workload that exceeds the processing capability of a single front-end server, a multi-front-end server configuration is appropriate. A multi-front-end server configuration can distribute the processing load among the machines on which front-end servers are running.

### (4) Dictionary server (DS)

The dictionary server provides batch management of the data dictionaries (dictionary tables) that contain database definition information.

Each system requires one dictionary server.

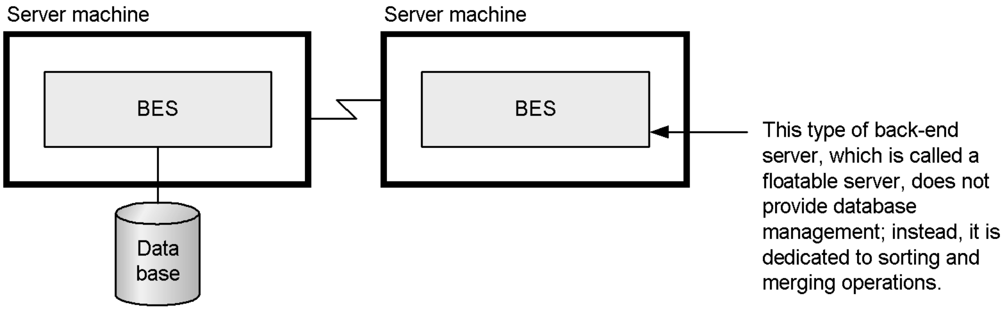
### (5) Back-end server (BES)

A back-end server manages the database. On the basis of execution directives received from front-end servers, the back-end servers access and lock the database and perform computational operations. Back-end servers also sort, merge, and join search results.

At least one back-end server is required per system (up to a maximum of 16,382 back-end servers). When multiple back-end servers are provided, a table can be divided among them so that it can be managed on a split basis.

Performance can be improved in a HiRDB parallel server configuration by providing back-end servers that do not provide database management but instead are dedicated specifically to processing high-overhead operations, such as sorting and joining. Such back-end servers are called *floatable servers*. The following figure shows a floatable server.

Figure 1–6: Floatable server


Server machine
Server machine
BES
BES
Data base
This type of back-end server, which is called a floatable server, does not provide database management; instead, it is dedicated to sorting and merging operations.

## 1.2.3 Multi-HiRDB configuration

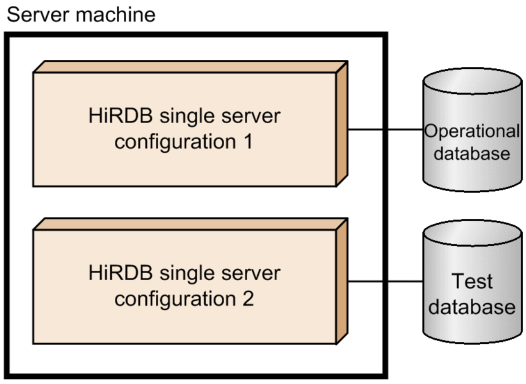
You can install multiple, separately operating HiRDB systems on one server machine. Such a system configuration is called a *multi-HiRDB* configuration. A multi-HiRDB configuration would be appropriate under the following circumstances:

- Running an operational system and a test system on the same server machine
- Running systems for different applications on the same server machine

In a multi-HiRDB configuration, you can freely combine HiRDB single server configurations and HiRDB parallel server configurations.

The following figure shows a multi-HiRDB configuration using HiRDB single server configurations.

Figure 1–7: Multi-HiRDB configuration using HiRDB single server configurations


Server machine
HiRDB single server configuration 1
Operational database
HiRDB single server configuration 2
Test database

**Explanation**

This is a multi-HiRDB configuration implemented on a HiRDB single server configuration.

HiRDB single server configuration 1 is designated as the operational system, and HiRDB single server configuration 2 serves as a test system.

# 1.3 Database access modes

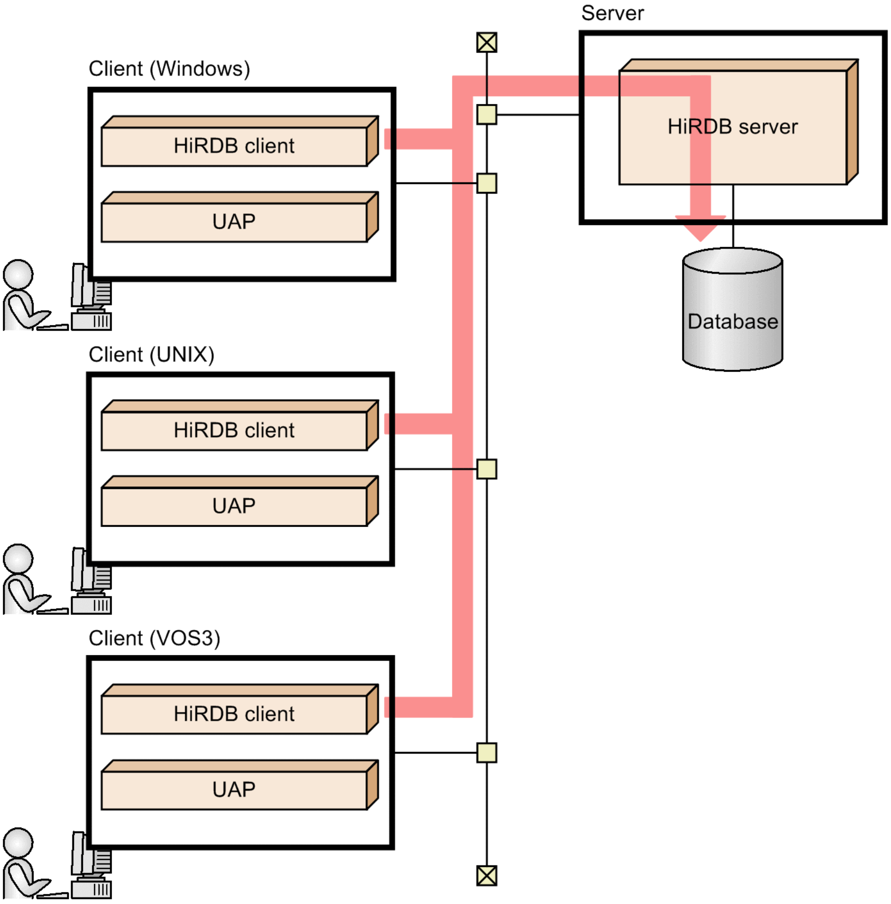
You access a HiRDB database by means of the SQL that is provided by HiRDB. This section explains the operating modes under which an SQL can be executed from a UAP.

## (1) Accessing a database by executing a UAP from a HiRDB client

This is the basic mode of operation. In it, you access a database by executing a UAP on a HiRDB client. Moreover, by using a HiRDB client that runs on VOS3, you can access HiRDB databases from UAPs running on VOS3 as well.

Note however, that the HiRDB server platforms to which you can connect are limited, depending on the version of the HiRDB client. For details, see Appendix *C. HiRDB Client and HiRDB Server Connectivity*. The following figure illustrates the operating mode in which a database is accessed by executing a UAP on a HiRDB client.

Figure 1–8: Accessing a database by executing UAPs from HiRDB clients

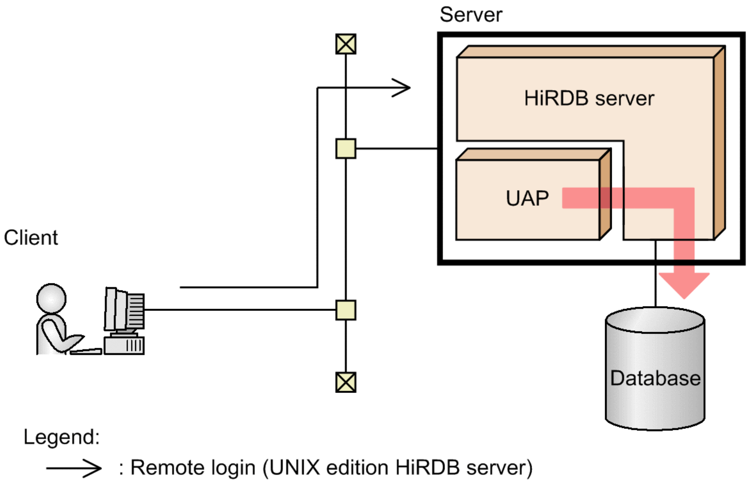


## (2) Accessing a database by executing a UAP running on the HiRDB server

In this mode, you can access a database by executing a UAP running on the HiRDB server. In the UNIX edition of the HiRDB server, if the client system does not have a HiRDB client, you can log in to the HiRDB server remotely.

The following figure illustrates the operating mode in which a database is accessed by executing a UAP running on the HiRDB server.

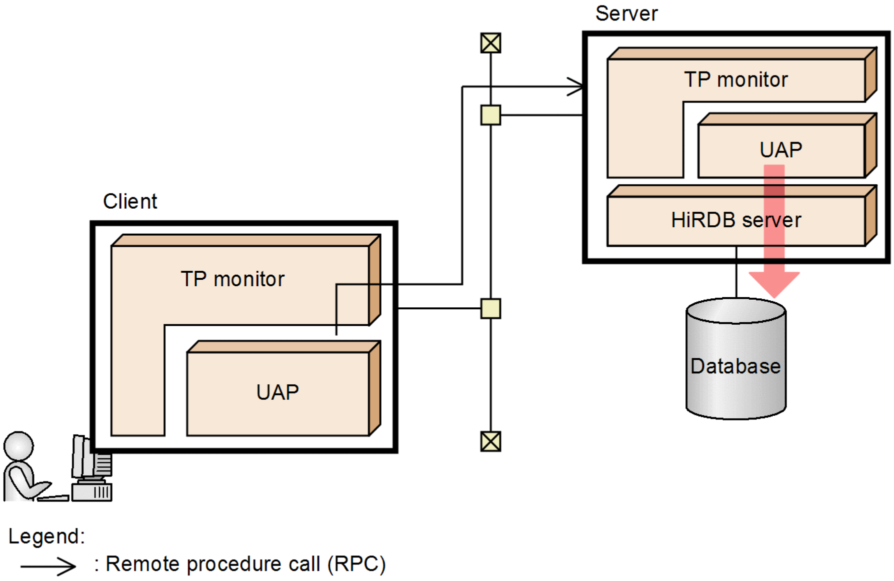
Figure 1–9: Accessing a database by executing a UAP running on the HiRDB server



## (3) Accessing a database by executing an OLTP UAP

In this mode, a HiRDB database is accessed when a UAP running on an OLTP machine (TP monitor) issues a service request. The client must have a service-requesting UAP; the server must have a service-providing UAP. The following figure illustrates the operating mode in which a database is accessed by executing an OLTP UAP.

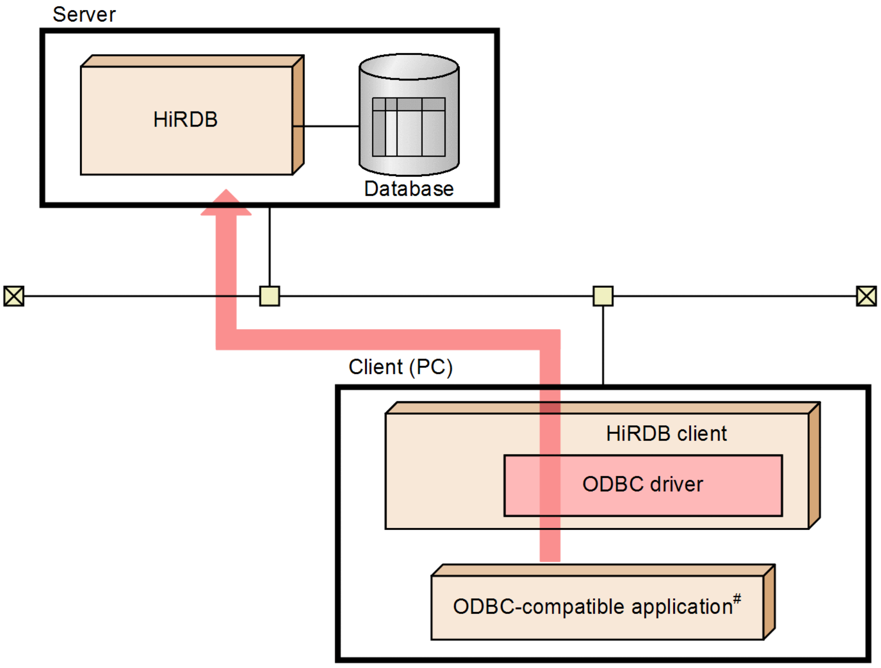
Figure 1–10: Accessing a database by executing an OLTP UAP



## (4) Linkage to ODBC-compatible applications

HiRDB provides an ODBC driver. When the ODBC driver is installed in a HiRDB client, ODBC-compatible applications can access the HiRDB database. ODBC-compatible applications include many commercially available programs, such as Microsoft Access and PowerBuilder. UAPs that use the ODBC function provided by HiRDB can also access the HiRDB database. The following figure illustrates the operating mode in which a database is accessed from an ODBC-compatible application.

Figure 1–11: Accessing a HiRDB database from an ODBC-compatible application



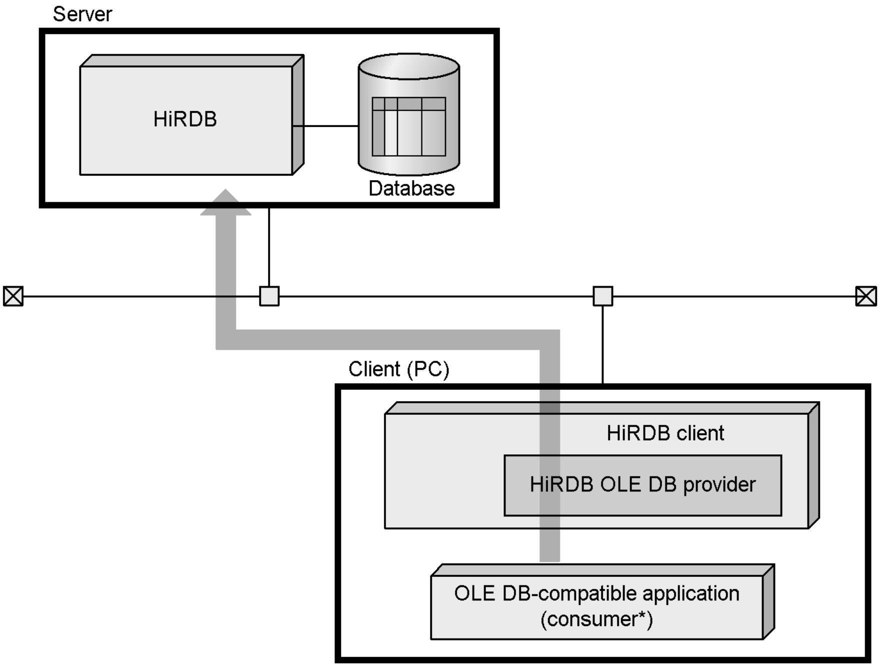
#: Includes ODBC-compatible UAPs.

For details about accessing a HiRDB database from an ODBC-compatible application, see the *HiRDB Version 9 UAP Development Guide*.

### (5) Linkage to OLE DB-compatible applications

HiRDB provides an OLE DB provider. By selecting the OLE DB provider when you install a HiRDB client, you can enable applications that support the OLE DB to access HiRDB databases. The following figure illustrates the operating mode in which a database is accessed from an OLE DB-compatible application.

Figure 1–12: Accessing a HiRDB database from an OLE DB-compatible application



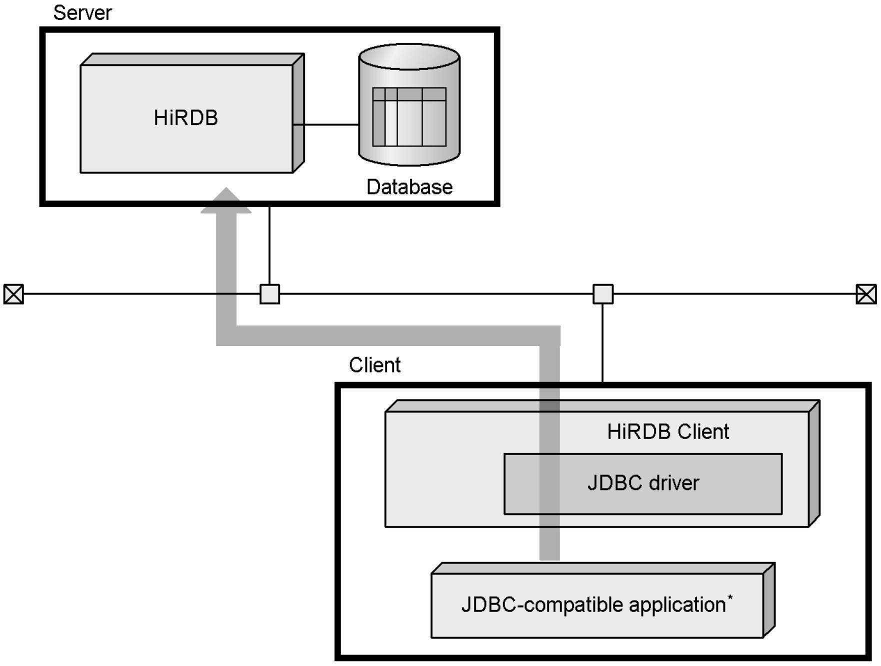
* Consumer refers to a program that calls an OLE DB method and interface.

For details about accessing a HiRDB application from an OLE DB-compatible application, see the *HiRDB Version 9 UAP Development Guide*.

## (6) Accessing a database from a JDBC-compatible application

HiRDB provides a JDBC driver. By selecting the JDBC driver when you install a HiRDB client, you can enable applications that support JDBC to access HiRDB databases. The following figure illustrates the operating mode in which a database is accessed from a JDBC-compatible application.

Figure 1–13: Accessing a HiRDB database from a JDBC-compatible application



*Also includes programs that call Java stored procedures and Java stored functions.

For details about accessing HiRDB from a JDBC-compatible application (using Java stored procedures and Java stored functions), see the *HiRDB Version 9 UAP Development Guide*.

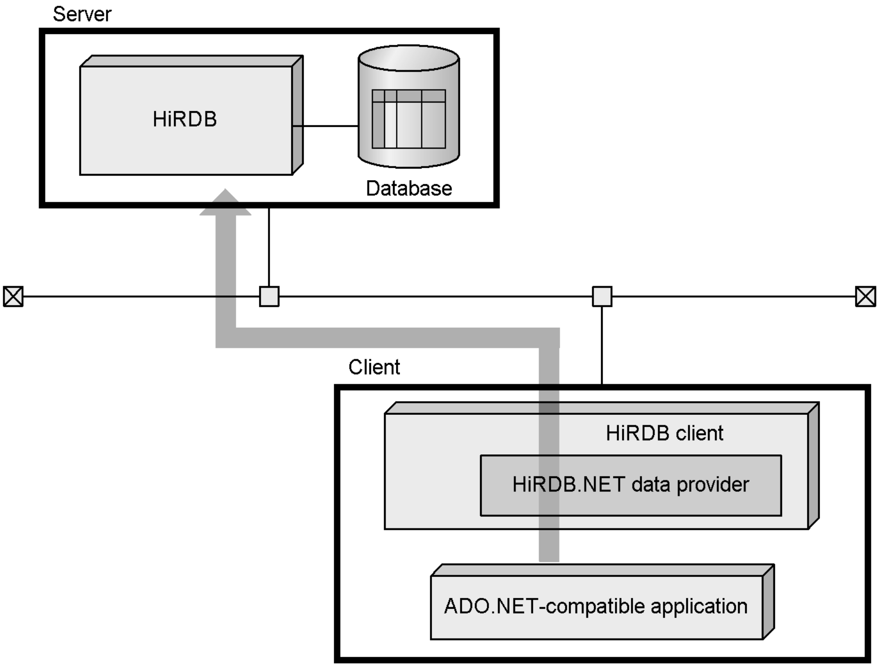
## (7) Accessing a database from an ADO.NET-compatible application

HiRDB uses ADO.NET to provide the HiRDB.NET data provider, which is needed to access HiRDB. The HiRDB.NET data provider complies with the ADO.NET specifications. By selecting the HiRDB.NET data provider when you install a HiRDB client, you can enable applications that support ADO.NET to access HiRDB databases.

HiRDB.NET data provider is bundled with a common set of basic interfaces provided by the System.Data space of Net Framework. HiRDB.NET data provider also includes its own extensions to perform array insertion and access repetition columns.

The following figure illustrates the operating mode in which a database is accessed from an ADO.NET-compatible application.

Figure 1–14: Accessing a database from an ADO.NET-compliant application



For details about accessing HiRDB from an ADO.NET-compliant application, see the *HiRDB Version 9 UAP Development Guide*.

# 2 Linking to HiRDB Option Program Products and Other HiRDB-Related Products

This chapter describes the functionality enabled by linking HiRDB to HiRDB option program products and other HiRDB-related products.

# 2.1 HiRDB option program products

This section describes the functions and operations that you can enable by using the following HiRDB option program products:

- HiRDB Advanced High Availability
- HiRDB Staticizer Option (UNIX edition only)
- HiRDB Non Recover FES
- HiRDB Disaster Recovery Light Edition (UNIX edition only)
- HiRDB Accelerator

## 2.1.1 HiRDB Advanced High Availability

Installing HiRDB Advanced High Availability allows you to use the following functions:

- Standby-less system switchover facility
- System reconfiguration command (`pdchgconf` command)
- Dynamic updating of global buffers (`pdbufmod` command)
- Table matrix partitioning
- Changing the partitioning storage conditions

### (1) Standby-less system switchover facility

The *standby-less system switchover facility* implements a system switchover method that enables HiRDB to continue providing services when a failure occurs by transferring processing to another unit. Because the standby-less system switchover facility does not require that standby resources be placed in reserve, it is significantly more economical than the previously available system switchover facility (the *standby system switchover facility*). With the previous system switchover facility, you had to set aside a server, CPU, and memory resources to be available for transfer of operations in the event of a failure. With the standby-less system switchover facility, there is no need to provide these redundant resources. You simply register another server as the alternate, and that server's unit takes over processing if a failure occurs. Although the processing performance of the unit that has taken over might be degraded, the overall cost of the system is reduced because you can make more effective use of resources.

The standby-less system switchover facility includes the *standby-less system switchover (1:1) facility* and the *standby-less system switchover (effects distributed) facility*. For details about the standby-less system switchover facility, see *8.1 System switchover facility*.

### (2) System reconfiguration command (pdchgconf command)

Previously, changing HiRDB system definitions meant having to stop HiRDB; however, by using the *system reconfiguration command* (`pdchgconf` command), you can change HiRDB system definitions without having to stop HiRDB. This allows you to perform system-related operations while HiRDB is running, such as changing the configuration of units or servers and adding system files. For details about the system reconfiguration command, see *6.5.4 System reconfiguration command (pdchgconf command)*.

### (3) Dynamic updating of global buffers (pdbufmod command)

Previously, adding, changing, or removing global buffers required you to change the `pdbuffer` operand in the HiRDB system definitions, which meant you had to stop HiRDB. Using the `pdbufmod` command, however, allows you to add, change, and remove global buffers while HiRDB is running. This is called *dynamic updating of global buffers*. For details about dynamic updating of global buffers, see *6.8.1(3) Dynamic updating of global buffers*.

### (4) Table matrix partitioning

*Table matrix partitioning* is a function that performs key range partitioning by using multiple columns as keys. The use of multiple columns as partitioning keys improves the performance of SQL code that is executed in parallel, and the use of multiple keys for searches accelerates processing by narrowing the searched range. In addition, partitioning database storage areas (RDAREAs) into smaller segments also reduces the time it takes to reorganize the database, to make backups, and to recover the database. For details about table matrix partitioning, see *3.3.9 Table matrix partitioning*.

### (5) Changing the partitioning storage conditions

Using `ALTER TABLE`, you can change the partitioning storage conditions of tables that have been row-partitioned by key range partitioning.# Changing a table's partitioning storage conditions enables you to reuse RDAREAs that contain outdated data and reduces work time. For details about changing partitioning storage conditions, see *3.3.10 Changing the partitioning storage conditions of a table*.

#

`ALTER TABLE` can be used to change a table's partitioning storage condition when one of the following partitioning methods was used:

- Boundary value specification
- Storage condition specification (if the equal sign (=) was used as the comparison operator for the storage condition)

## 2.1.2 HiRDB Staticizer Option (UNIX edition only)

Installing HiRDB Staticizer Option allows you to use the *inner replica facility*, which enables you to build database systems that support non-stop services.

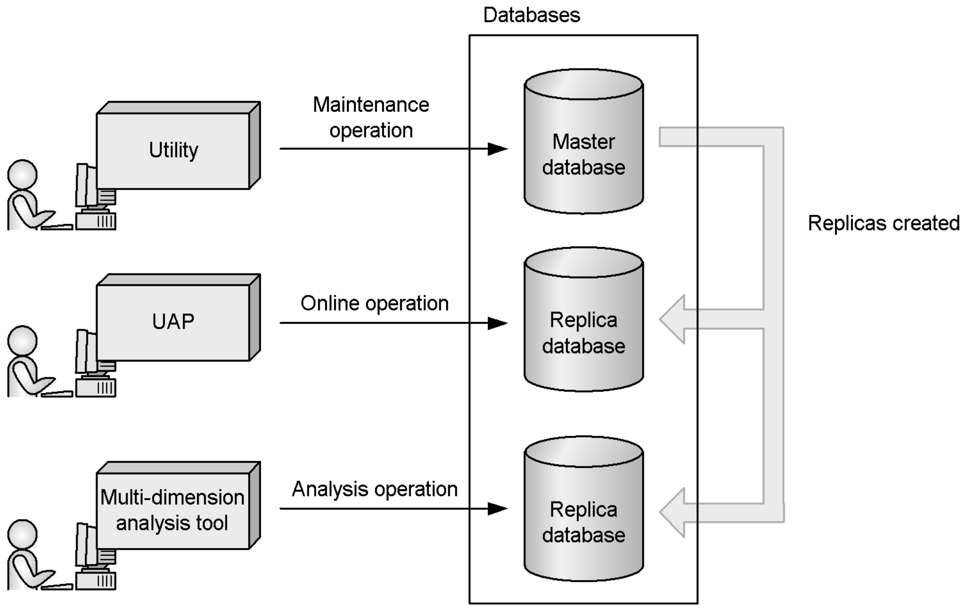
### (1) Overview of the inner replica facility

When the system operates online non-stop 24 hours per day, 365 days per year, you lose the time you need to perform database maintenance (tasks such as reorganizing databases, loading data, backing up, and so on). Moreover, creating a datamart for performing aggregate and analytical processing while online operations are ongoing might degrade network responses times when datamart operations such as extracting added or updated data are being performed. The inner replica facility remedies these kinds of situations.

Using a disk system or software that provides mirroring, the inner replica facility creates a replica database, which is a copy of the master database. Online operations use the replica database, while database maintenance and datamart creation operations use the master database. By employing the inner replica facility in this way, you can perform database maintenance and datamart creation operations concurrently with online operations.

The figure below provides an overview of the inner replica facility. For details about the inner replica facility, see the *HiRDB Staticizer Option Version 9 Description and User's Guide*.

Figure 2–1: Overview of the inner replica facility



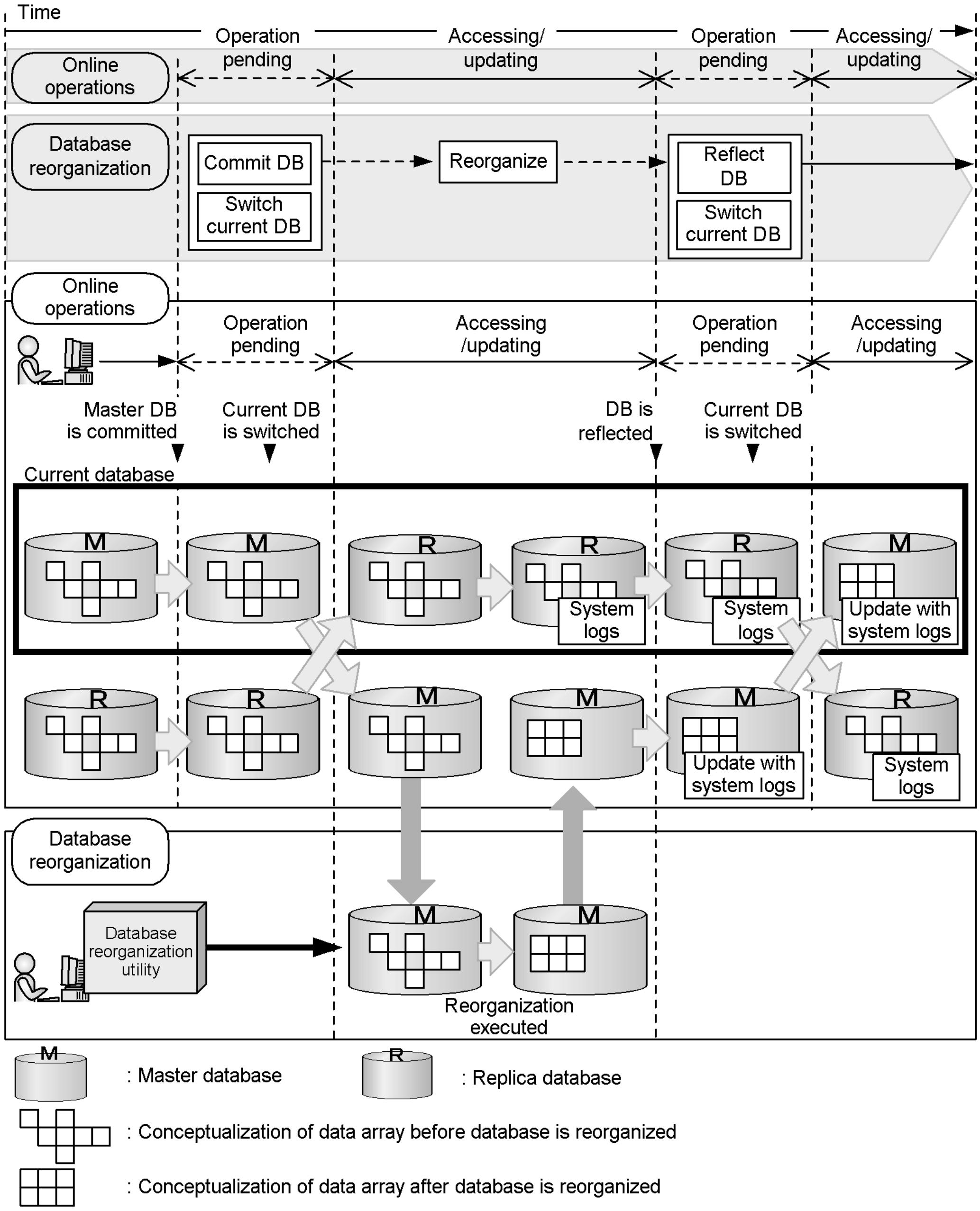
## (2) Application examples of the inner replica facility

The following subsections provide application examples of the inner replica facility.

### (a) Reorganizing a database

You can use the inner replica facility to allow users to access and update a database while it is being reorganized. This is called *updatable online reorganization*. Unless you are using the inner replica facility, databases cannot be accessed or updated while they are being reorganized. In addition, the inner replica facility minimizes adverse affects on online operations, because normal work operations are performed with the replica database. The following figure shows reorganization of a database while the inner replica facility is being used.

Figure 2–2: Database reorganization while using the inner replica facility



**Explanation**

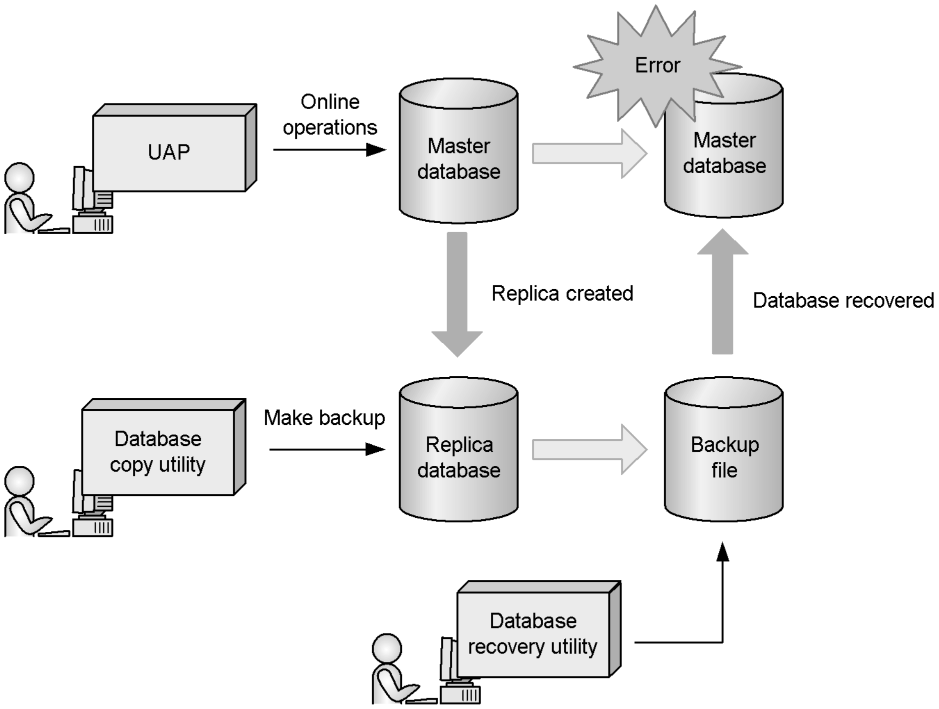
The database reorganization utility (`pdrorg` command) is used to reorganize the master database, while the replica database is used for online operations.

Before the database is reorganized, it is reflected and the current database is switched over. While this is occurring, the database cannot be accessed or updated. After the current database is switched over, the master database is reorganized, and the replica database is used for online operations. After reorganization is finished, the system logs from the replica database are used as input information to reflect to the master database, the updates that were performed during reorganization. This is called *reflection processing*.

### (b) Making a backup

You can use the inner replica facility to allow users to access and update a database while it is being backed up. The following figure shows database backup while the inner replica facility is being used.

Figure 2–3: Making a backup while using the inner replica facility



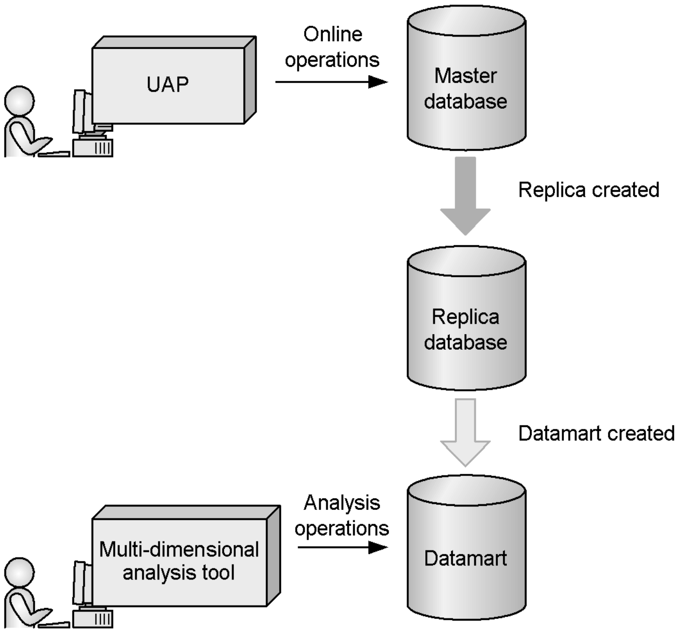
**Explanation**

A backup is made from the replica database. The master database can still be accessed and updated while the backup is being made. If a failure occurs on the master database, the backup made from the replica database can be used to recover the master database.

### (c) Creating a datamart

You can use the inner replica facility to create, from the replica database, a datamart to be used for analytical operations. This allows you to perform online operations and analytical operations at the same time. The following figure shows datamart creation while the inner replica facility is being used.

Figure 2–4: Datamart creation while using the inner replica facility



**Explanation**

A datamart for aggregate and analytical operations can be created from the replica database while online operations are ongoing. Analyzing data from a number of viewpoints increases the amount of data in the datamart, as well as the time required to create the datamart. By using the inner replica facility, you can create the datamart while online operations are being performed, without having to worry about the time it takes or the effects it has on online operations.

## 2.1.3 HiRDB Non-Recover FES

Installing Non-Recover FES enables you to use front-end servers that do not need to be recovered in the event of a failure. In this case, when the unit containing such a front-end server fails and is forcibly terminated, HiRDB automatically completes any uncommitted transactions, which means that the database is updated even if the unit containing the front-end server is not restarted. For details about recovery-unnecessary front-end servers, see *8.2 Recovery-unnecessary front-end servers*.

## 2.1.4 HiRDB Disaster Recovery Light Edition (UNIX edition only)

Installing HiRDB Disaster Recovery Light Edition enables you to use the log-only synchronous method for Real Time SAN Replication. Compared with the all synchronous, all asynchronous, and hybrid methods, the log-only synchronous method allows you to reduce communication traffic. For details about the log-only synchronous method for Real time SAN Replication, see *8.3.7 Log-only synchronous method*

## 2.1.5 HiRDB Accelerator

Installing HiRDB Accelerator enables you to use the following facility:

- Rapid batch facility

### (1) Rapid batch facility

Once you have installed HiRDB Accelerator, you can use in-memory data processing to accelerate batch-processing speed. In-memory data processing loads all of the data in RDAREAs into memory in a batch operation. During batch processing, only the data in memory is updated; the data on the disk is not updated. Upon completion of batch processing, the updated data in memory is written onto the disk in a batch operation. Since no disk input or output occurs during batch processing, the time taken for batch processing is shortened accordingly. For details about in-memory data processing, see *6.8.2 In-memory data processing*.

# 2.2 Linkage to data linkage products

This section describes the facilities that you can enable by using data linkage products.

## 2.2.1 Linkage to HiRDB Datareplicator and HiRDB Dataextractor

Linking to HiRDB Datareplicator and HiRDB Dataextractor allows you to use the *replication facility*. The replication facility provides the ability to duplicate the content of a distributed database into another database. Using the replication facility, you can duplicate information from one database into another database on a different system, which helps support the management of data in a distributed system environment. The replication facility includes the following two modes:

- Data linkage facility
- Database extraction/reflection service facility

### (1) Data linkage facility

The data linkage facility causes the updates to a database residing on a mainframe DBMS or a HiRDB system to be reflected automatically in HiRDB databases at local nodes. The data linkage facility requires the HiRDB Datareplicator, which is a related HiRDB product. The data linkage facility has the following characteristics:

- At fixed time intervals, the data linkage facility copies in sequence all updates to a corporate database into departmental databases so that the latest data available in the corporate database can be used from the departmental databases.
- The data linkage facility can extract selected data from a corporate database and copy in sequence all updates to the selected data from the corporate database into departmental databases. In this manner, the data linkage facility is able to provide data that is well-suited for data warehouse applications.

**Note**

If you use a HiRDB facility not supported by HiRDB Datareplicator, the data linkage facility might become disabled. Furthermore, some columns might not be processed by the data linkage facility, depending on the column attributes. For details about the column attributes that are not supported by the data linkage facility, see the *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide*. For the most recent, detailed information, see the online manuals published on the HiRDB website.

### (2) Database extraction/reflection service facility

The database extraction/reflection service facility transfers data stored in a mainframe DBMS or another HiRDB system to HiRDB databases at local nodes. The database extraction/reflection service facility requires HiRDB Dataextractor, which is a related HiRDB product. The database extraction/reflection service facility has the following characteristics:

- At a specific time, the database extraction/reflection service facility can copy in batch information from a corporate database into departmental databases. In this manner, departmental database tables can be initialized or all data can be updated to the latest version.
- By specifying data extraction conditions, the database extraction/reflection service facility can be used to extract selected data from a corporate database so that departmental databases that are appropriate for particular types of operations can be created.
- The database extraction/reflection service facility eliminates the need for UAPs that extract data or provide such operations as character code set conversion and file transfer.

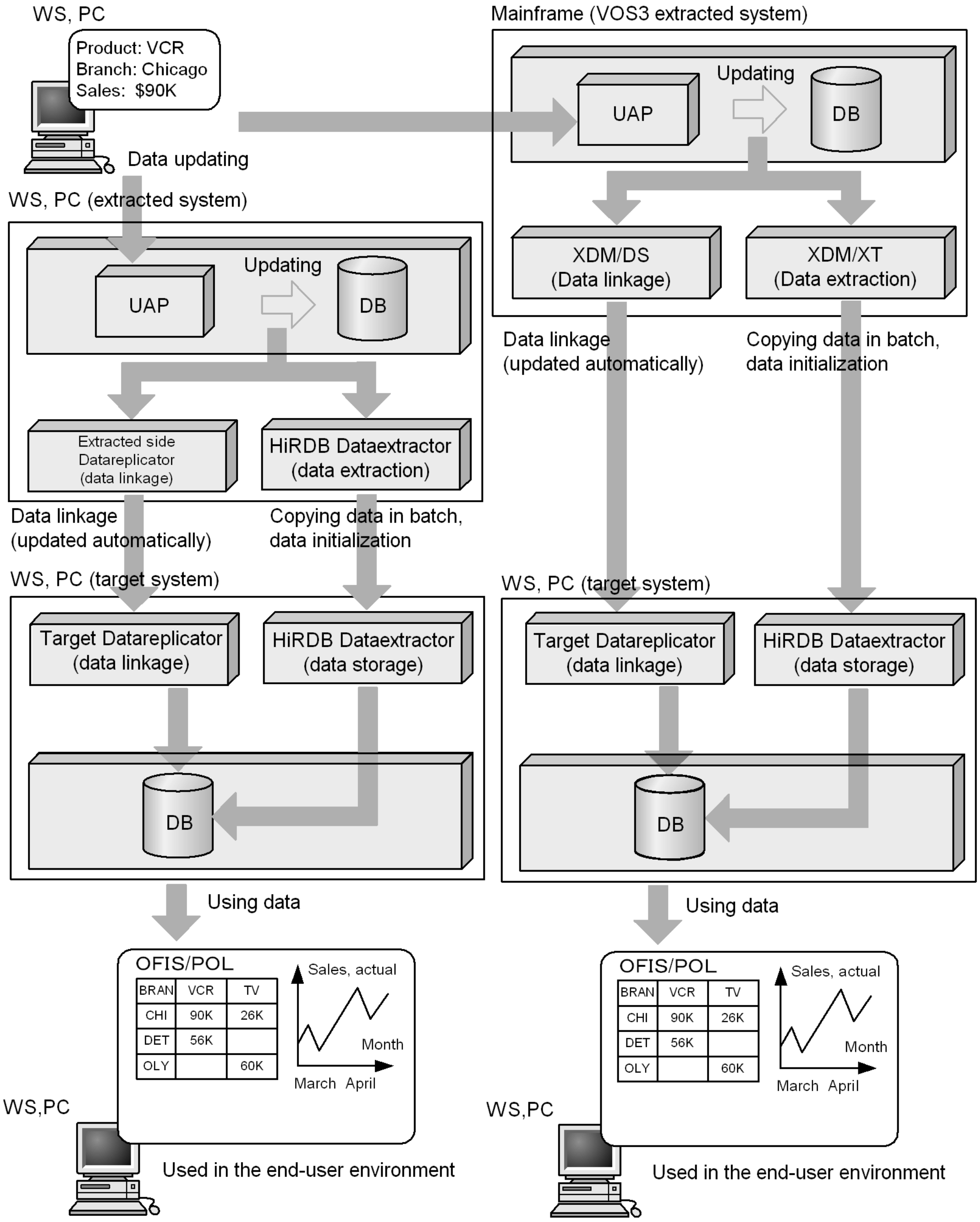
**Note**

Some columns are not eligible for processing by the database extraction/reflection service facility, depending on the column attributes. For details about the column attributes that are not supported by the database extraction/reflection service facility, see the *HiRDB Dataextractor Version 8 Description, User's Guide and Operator's Guide*.

### (3) Example of using the replication facility

The following figure shows an example of using the replication facility.

Figure 2–5: Example of using the replication facility



For details about application examples of the replication facility, see the *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide*, and the *HiRDB Dataextractor Version 8 Description, User's Guide and Operator's Guide*.

### (4) Products required for replication facility

#### (a) Products required in HiRDB systems

The data linkage facility requires HiRDB Datareplicator. The database extraction/reflection service facility requires HiRDB Dataextractor, except when the remote system is RDB1 E2 or PDM II E2 of VOS1.

### (b) DBMS products that can be linked to HiRDB using the replication facility

The following table lists DBMS products that can be linked to HiRDB by means of the replication facility.

Table 2–1: Products that can be linked to HiRDB by means of the replication facility

| Linkable DBMS product (and applicable operating system) | Required replication facility product | |
|---|---|---|
| | Data linkage | Database extraction/reflection service facility |
| XDM/RD E2 (VOS3) | XDM/DS | XDM/XT |
| XDM/SD E2 (VOS3) | | |
| ADM (VOS3) | VOS3 Database Datareplicator or XDM/DS | |
| PDM II E2 (VOS3)[#1] | VOS3 Database Datareplicator or XDM/DS | |
| | File transfer program[#2] | |
| PDM II E2 (VOS1) | File transfer program[#2] | PDM II Dataextractor |
| TMS-4V/SP (VOS3) | VOS3 Database Datareplicator or XDM/DS | XDM/XT |
| RDB1 E2 (VOS1) | File transfer program[#2] | RDB1 Dataextractor |

#1: In the case of PDM II E2 of VOS3, it is possible to perform data linkage using a data linkage product at the same time data linkage is being performed using a file transfer program.

#2: Both XFIT and an XFIT-related product are needed for the file transfer program. For details, see the *HiRDB Datareplicator Version 8 Description, User's Guide and Operator's Guide*.

## (5) Setting the HiRDB environment when the data linkage facility is used

When the data linkage facility is used (in conjunction with extracting data for updating HiRDB databases), the following system definition operands must be specified:

`pd_rpl_hdepath`

Specify the HiRDB Datareplicator operation directory for the extracted side.

`pd_rpl_init_start`

Specify when the HiRDB linkage facility is to be started.

`pd_log_rpl_no_standby_file_opr`

Specify the action to be taken if system log information cannot be created in the swapped-in file.

## 2.3 Linkage to OLTP products

Transaction processing can be performed when HiRDB is linked to OLTP. This section provides an overview of linking to OLTP products. For details about the OLTP linkage procedures, see the *HiRDB Version 9 Installation and Design Guide*; for details about creating a UAP that uses a linkage to OLTP, see the *HiRDB Version 9 UAP Development Guide*.

### 2.3.1 OLTP products to which HiRDB can be linked

HiRDB can be linked to the following OLTP products:

- OpenTP1
- TPBroker for C++
- TUXEDO
- WebLogic Server
- OpenTP1/Server Base Enterprise Option (hereafter, TP1/EE)

Depending on the OS, however, you might not be able to link your HiRDB system to an OLTP product. The following table shows the possibility of linkage to OLTP products by OS.

Table 2–2: Linkage to OLTP by OS

| HiRDB system | OLTP product | | | | |
|---|---|---|---|---|---|
| | OpenTP1 | TPBroker for C++ | TUXEDO | WebLogic Server | TP1/EE |
| HP-UX | Y | Y | N | Y | N |
| Solaris | Y | Y | Y | Y | N |
| AIX | Y | N | N | N | Y |
| Linux | Y | N | N | N | Y |
| Windows | Y | Y | Y | Y | N |

Legend:

Y: Can be linked.

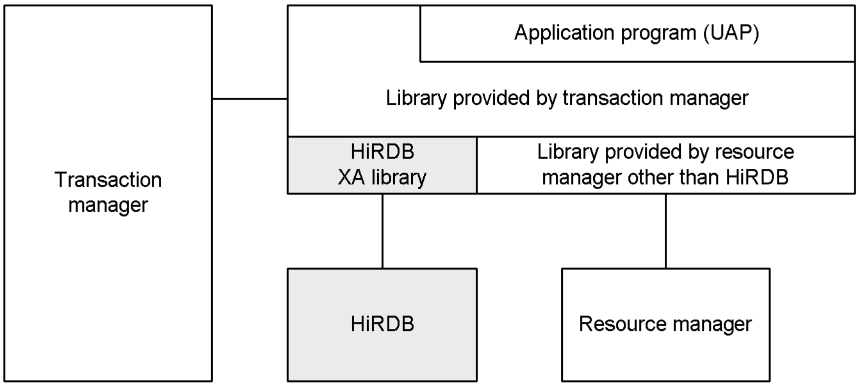
N: Cannot be linked.

### 2.3.2 HiRDB XA library

HiRDB uses the X/Open XA interface for linkage to OLTP. This interface, which is based on the X/Open standards, specifies the connection between the transaction manager (TM) and the resource manager (RM) in a distributed transaction processing (DTP) system. Use of the XA interface enables the resource manager's transaction processing to be controlled by the transaction manager. To do this, you must link the library provided by the transaction manager to your UAP.

HiRDB provides the HiRDB XA library so that it can control UAP processing. This library conforms to the XA interface specifications of the X/Open DTP software architecture. The following figure shows the position of HiRDB in an X/Open DTP model.

Figure 2–6: Position of HiRDB in X/Open DTP model


Transaction manager
Application program (UAP)
Library provided by transaction manager
HiRDB XA library
Library provided by resource manager other than HiRDB
HiRDB
Resource manager

## 2.3.3 Functions provided by the HiRDB XA library

The following table lists the functions provided by the HiRDB XA library.

Table 2–3: Functions provided by the HiRDB XA library

| Facility | Explanation |
|---|---|
| Transaction transfer | Executes transaction commit processing in a process different from the one that was used when the UAP accessed HiRDB. The UAP in this case uses the HiRDB XA library to connect to HiRDB.<br>You specify in the `PDXAMODE` operand of the client environment definition whether the transaction transfer function is to be used. For details on transaction transfer, see the *HiRDB Version 9 Installation and Design Guide*. |
| Single-phase optimization | Optimizes two-phase commitment control into a single phase. |
| Read-only | Transaction manager performs optimization without issuing a commit request in the second phase, if HiRDB resource has not been updated when a prepare request is issued. |
| Dynamic transaction registration | HiRDB registers dynamically a transaction immediately before executing a UAP. |
| Multi-connection facility | This function separately executes multiple CONNECTs to a HiRDB server from a single process. For details on the multi-connection facility under the X/Open XA interface environment, see the *HiRDB Version 9 UAP Development Guide*. |

**Note**

The HiRDB XA library does not provide the asynchronous XA call function (function that allows the transaction manager to issue asynchronous calls to the HiRDB XA library).

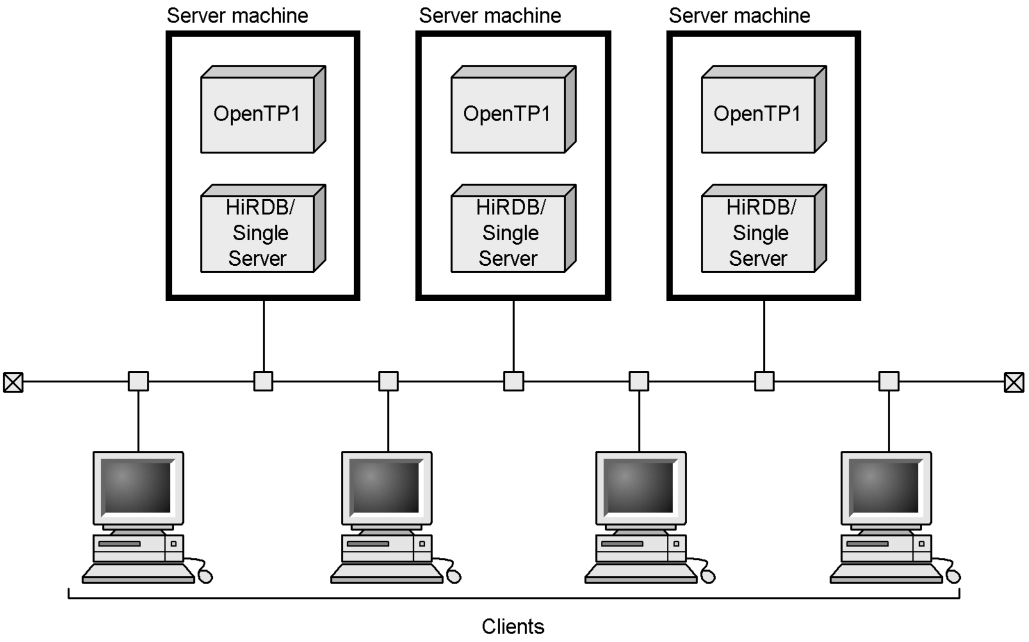
## 2.3.4 System configuration

This section uses OpenTP1 as an example for explaining the configuration of a system that uses OLTP.

### (1) Linkage to a HiRDB single server configuration

By linking HiRDB single server configurations to OpenTP1, processing at multiple HiRDB single server configurations can be executed centrally. In such a case, the database is partitioned and allocated using a method such as key range partitioning, and OpenTP1 running on each server machine allocates processing to each HiRDB single server configuration. In this way, large-scale transaction processing can be implemented. Linkage with OpenTP1 is the recommended means of integrating multiple HiRDB single server configurations. The following figure shows linkage between HiRDB single server configurations and OpenTP1.

Figure 2–7: Linkage between HiRDB single server configurations and OpenTp1



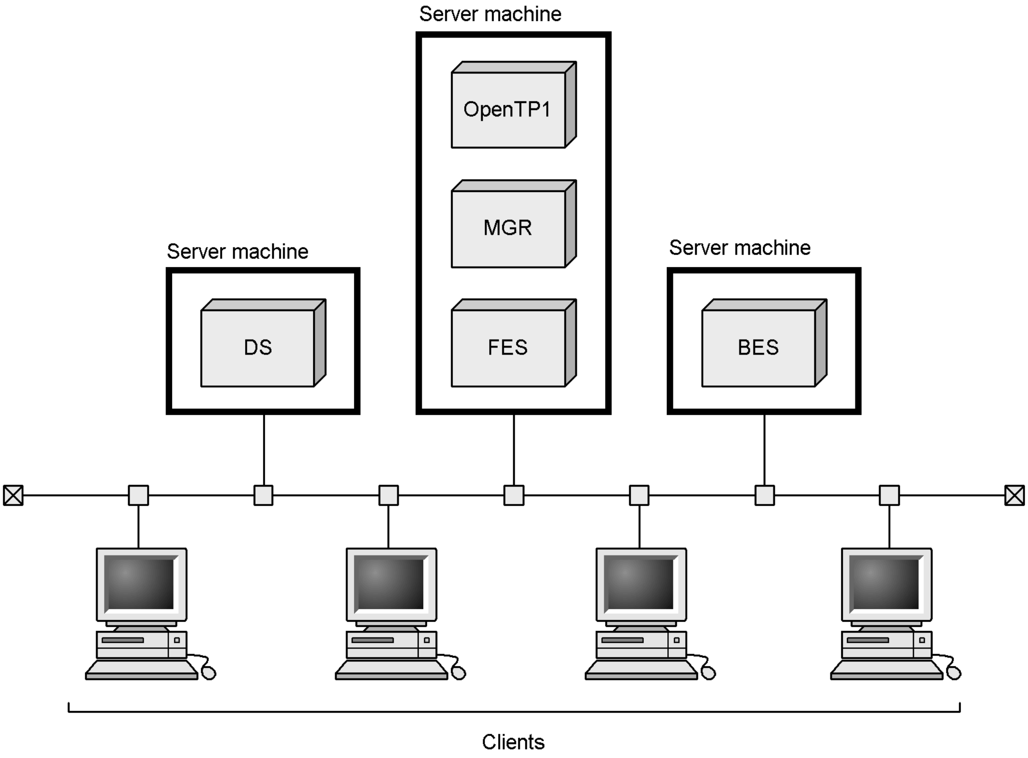
**Explanation**

Each of the three server machines is provided with a HiRDB single server configuration and OpenTP1; transaction processing is allocated to each of the three HiRDB single server configurations.

## (2) Linkage to a HiRDB parallel server configuration

By linking a HiRDB parallel server configuration to OpenTP1, high-performance, high-load transaction processing can be achieved. The following figure shows linkage between a HiRDB parallel server configuration and OpenTP1.

Figure 2–8: Linkage between a HiRDB parallel server configuration and OpenTP1



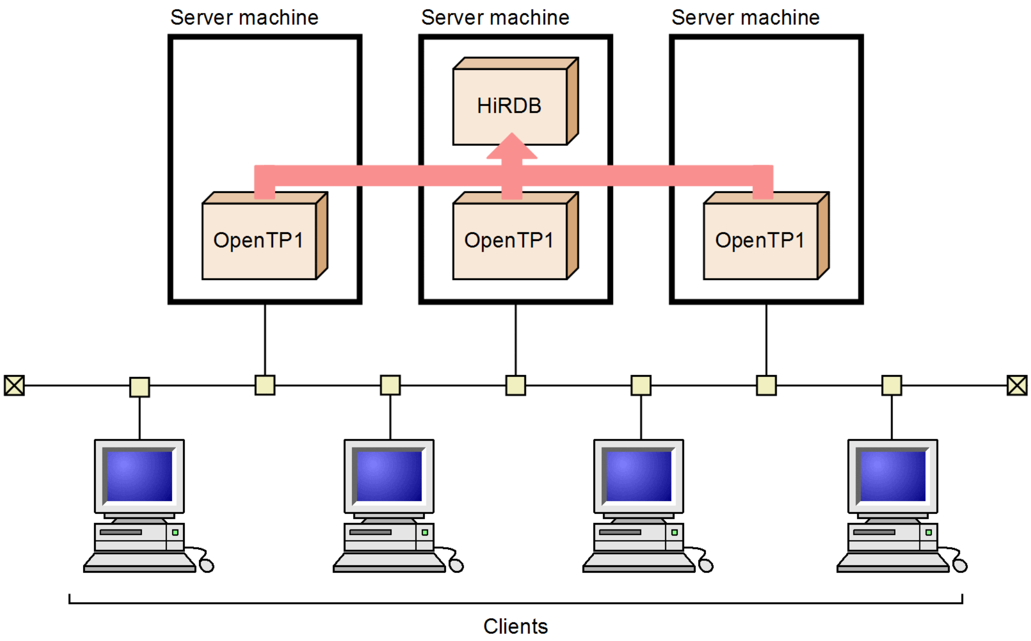
**Explanation**

OpenTP1 is provided at the server machine where the system manager and front-end server are installed in order to control transaction processing.

## (3) Linkage to multiple OpenTP1

The following figure shows linkage to multiple OpenTP1 instances.

Figure 2–9: Linkage to multiple OpenTP1



Note: The OLTP identifier (PDTMID in the client environment definition) of each OpenTP1 must be unique.

**Note**

In the above configuration, a single HiRDB and three OpenTP1 instances communicate in a client-server setup to manage transaction processing.

## 2.3.5 Registration of HiRDB in the transaction manager

The two methods explained as follows are provided for registering HiRDB into the transaction manager as a resource manager.

**Dynamic registration**

When this method is used, the UAP is placed under the control of the transaction manager when the UAP issues the first SQL statement in a transaction. In the case of a UAP that accesses multiple resource managers including HiRDB, or in the case of a UAP that might not access HiRDB at all, dynamic registration reduces the transaction manager's transaction control overhead with respect to HiRDB.

**Static registration**

When this method is used, the UAP is placed under the control of the transaction manager, regardless of whether the UAP issues SQL statements. If the transaction manager is OpenTP1 and the connection between the UAP and HiRDB is severed (due to abnormal termination of the unit or the server process, for example), static registration eliminates the need to restart the UAP, because an OpenTP1 facility will reconnect the UAP when a transaction is started.

For details about registering HiRDB into the transaction manager, see the *HiRDB Version 9 Installation and Design Guide*.

# 2.4 Linkage to operation support products

This section describes operation support products used with HiRDB.

- HiRDB SQL Executer
- HiRDB Control Manager
- HiRDB SQL Tuning Advisor
- JP1/Performance Management - Agent Option for HiRDB
- JP1/Base
- JP1/Integrated Management
- JP1/Automatic Job Management System 2
- JP1/NETM/Audit

## 2.4.1 HiRDB SQL Executer

HiRDB SQL Executer enables interactive execution of SQL programs. Because it can perform data validation, confirmation, modification, and relatively simple, routine operations, HiRDB SQL Executer is well-suited for database maintenance and simple validation tasks. In order to use HiRDB SQL Executer on a HiRDB client, you must have either HiRDB/Developer's Kit or HiRDB/Run Time. For details about the prerequisite products, see the HiRDB SQL Executer *Release Notes*.

## 2.4.2 HiRDB Control Manager

HiRDB Control Manager enables you to use GUI-based mouse operations on a PC to perform basic operations, such as displaying and changing HiRDB server statuses, and backing up and recovering databases. You can also centrally operate multiple HiRDB systems from a single graphic user interface, which reduces the system operation workload.

HiRDB Control Manager supports the following operations:

- Concurrent management of multiple HiRDB systems
- Display of HiRDB statuses
- Starting and stopping HiRDB
- Execution of the database copy utility and database recovery utility
- Operations on RDAREAs
- Operations on tables
- Operations on system log files
- Catalog registration and schedule execution
- Display of XDM/RD activity status together with RDAREA and table structure information
- Display of relationships between tables and disk volumes and operations from the displayed window

## 2.4.3 HiRDB SQL Tuning Advisor

HiRDB SQL Tuning Advisor is a product designed to help you locate and resolve SQL performance bottlenecks. It analyzes execution records output by HiRDB servers and clients, and displays the results together with advice in an easy-to-understand format.

HiRDB SQL Tuning Advisor provides the following features:

- Efficiently locates SQL code that might cause performance problems.
- Efficiently tunes SQL code that might cause performance problems.

HiRDB SQL Tuning Advisor provides the following two main facilities:

- SQL trace analysis facility

  The SQL trace analysis facility analyzes SQL trace information generated by the HiRDB client, and displays on screen the aggregate time in the units occupied by the UAP or SQL code. This enables you to locate SQL code bottlenecks by summing the SQL processing time and extracting SQL code that is taking a long time to execute.
- Access path analysis facility

  The access path analysis facility analyzes and displays on screen access path information generated by the HiRDB client. It can also graphically display the join relationships between tables. It can locate areas that might cause performance problems with respect to the methods used to join the tables (table scan, cross-join, and so on) and display warning advice.

In order to use HiRDB SQL Tuning Advisor on a HiRDB client, you must have either HiRDB/Developer's Kit or HiRDB/Run Time. For details about the prerequisite products, see the HiRDB SQL Tuning Advisor's *Release Notes*.

### 2.4.4 JP1/Performance Management - Agent Option for HiRDB

JP1/Performance Management - Agent Option for HiRDB is a JP1/Performance Management agent product that collects HiRDB performance data. On distributed systems combining various platforms, JP1/Performance Management allows you to centrally manage the performance of the operating systems, server applications, databases, and other resources. By using JP1/Performance Management - Agent Option for HiRDB, you can monitor HiRDB from JP1/Performance Management. This means that you can centrally manage HiRDB performance together with that of the operating systems and other server applications. With JP1/Performance Management - Agent Option for HiRDB, you can collect performance data on the following HiRDB resources:

- Global buffers
- Server processes
- HiRDB file system area for work table files
- HiRDB units and servers
- RDAREAs
- Utilization rate of locked resources management tables

For details about JP1/Performance Management - Agent Option for HiRDB, see the manual that is applicable to the version of JP1 that you are using.

- For JP1 Version 8 and later

  *JP1/Performance Management - Agent Option for HiRDB*
- For JP1 Version 6 and JP1 Version 7i

  *JP1/Performance Management - Agent for HiRDB*

### 2.4.5 JP1/Base

Events such as HiRDB startup and termination can be reported to *JP1/Base*. JP1/Base manages reported HiRDB events as JP1 events.

As a result, you can perform the following operations:

- Use JP1/Integrated Manager to manage events.
- Link to JP1/Automatic Job Management System 3 (or JP1/Automatic Job Management System 2) to automatically execute jobs.
- Use JP1/NETM/Audit to collect system audit logs and centrally manage them.

The individual products are explained starting in Section *2.4.6*.

For details about JP1/Base, see the manual that is applicable to the version of JP1 that you are using:

- JP1 Version 8 or later

  *Job Management Partner 1/Base User's Guide*
- JP1 Version 6 and JP1 Version 7i

*JP1 V6 JP1/Base* or *Job Management Partner 1/Base*

### (1) Reporting events

You specify the operands shown below to have HiRDB events reported to JP1/Base:

- `pd_jp1_use` operand: `Y`
- `pd_jp1_event_level` operand: `1` or `2`

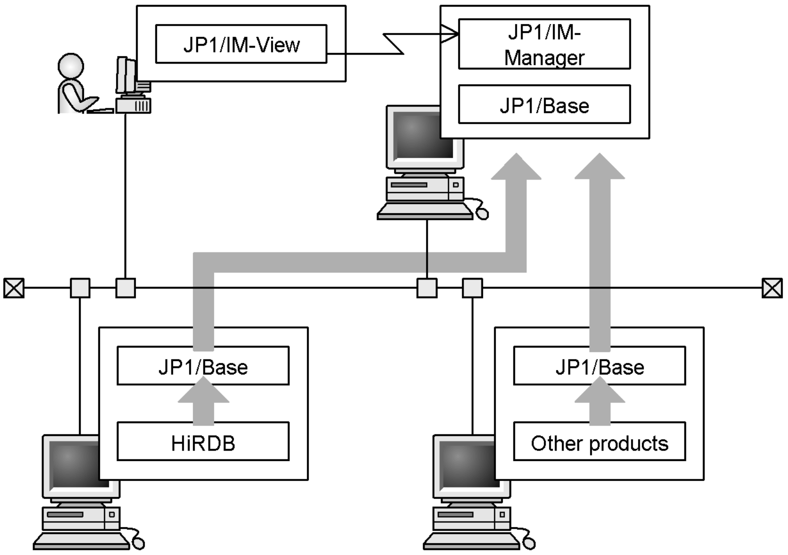
If you specify `1` for the `pd_jp1_event_level` operand, only *basic attributes* are reported. If you specify `2`, *extended attributes* are also reported.

For details about the events that can be reported, see the *HiRDB Version 9 Installation and Design Guide*.

## 2.4.6 JP1/Integrated Management

JP1/Integrated Management optimizes (filters) JP1 events managed by JP1/Base and allows events issued by the system to be managed centrally by means of JP1 events. By reporting HiRDB events to JP1/Base, you can use JP1/Integrated Management to manage HiRDB events as you would those of other products. You can check events on the windows displayed by JP1/Integrated Management. The following figure provides an overview of event monitoring using JP1/Integrated Management.

Figure 2–10: Overview of event monitoring using JP1/Integrated Management



: Flow of JP1 events

JP1/IM-View: Job Management Partner 1/Integrated Management - View
JP1/IM-Manager: Job Management Partner 1/Integrated Management - Manager

For details about the preparations for displaying HiRDB's native set of extended attributes within an overview of event monitoring using JP1/Integrated Management, see the *HiRDB Version 9 Installation and Design Guide*. For an overview of event monitoring using JP1/Integrated Management, see the manual that is applicable to the version of JP1 that you are using:

- JP1 Version 9
  *Job Management Partner 1/Integrated Management - Manager Overview and System Design Guide*
- JP1 Version 8
  *Job Management Partner 1/Integrated Management - Manager System Configuration and User's Guide*

- JP1 Version 7i

  *Job Management Partner 1/Integrated Manager - Console*

## 2.4.7 JP1/Automatic Job Management System 3

In the case of a HiRDB parallel server configuration, operations such as unloading system log files can become quite complex. In these cases, by reporting HiRDB events to JP1/Base, you can use reported events as triggers for JP1/Automatic Job Management System 3 to execute jobs automatically, thereby automating HiRDB operations.

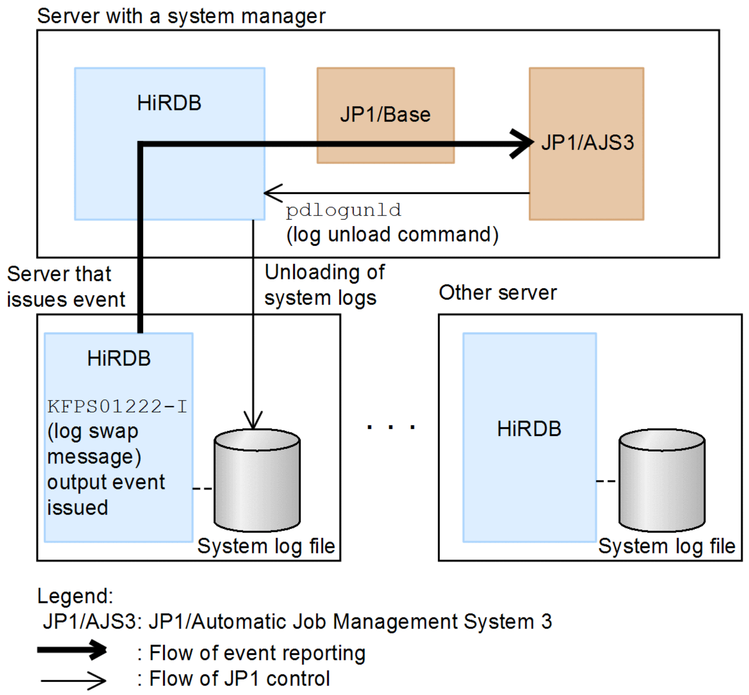
Reference note

*JP1/Automatic Job Management System 3* is the product name in JP1 Version 9.

HiRDB can also be linked to *JP1/Automatic Job Management System 2* of JP1 Version 8 or earlier.

The following figure shows automatic control when system log files are unloaded via linkage with JP1.

Figure 2–11: Automatic control when system log files are unloaded by linking with JP1



To link with JP1/Automatic Job Management System 3, or JP1/Automatic Job Management System 2, see the manual for the JP1 version that you are using.

## 2.4.8 JP1/NETM/Audit

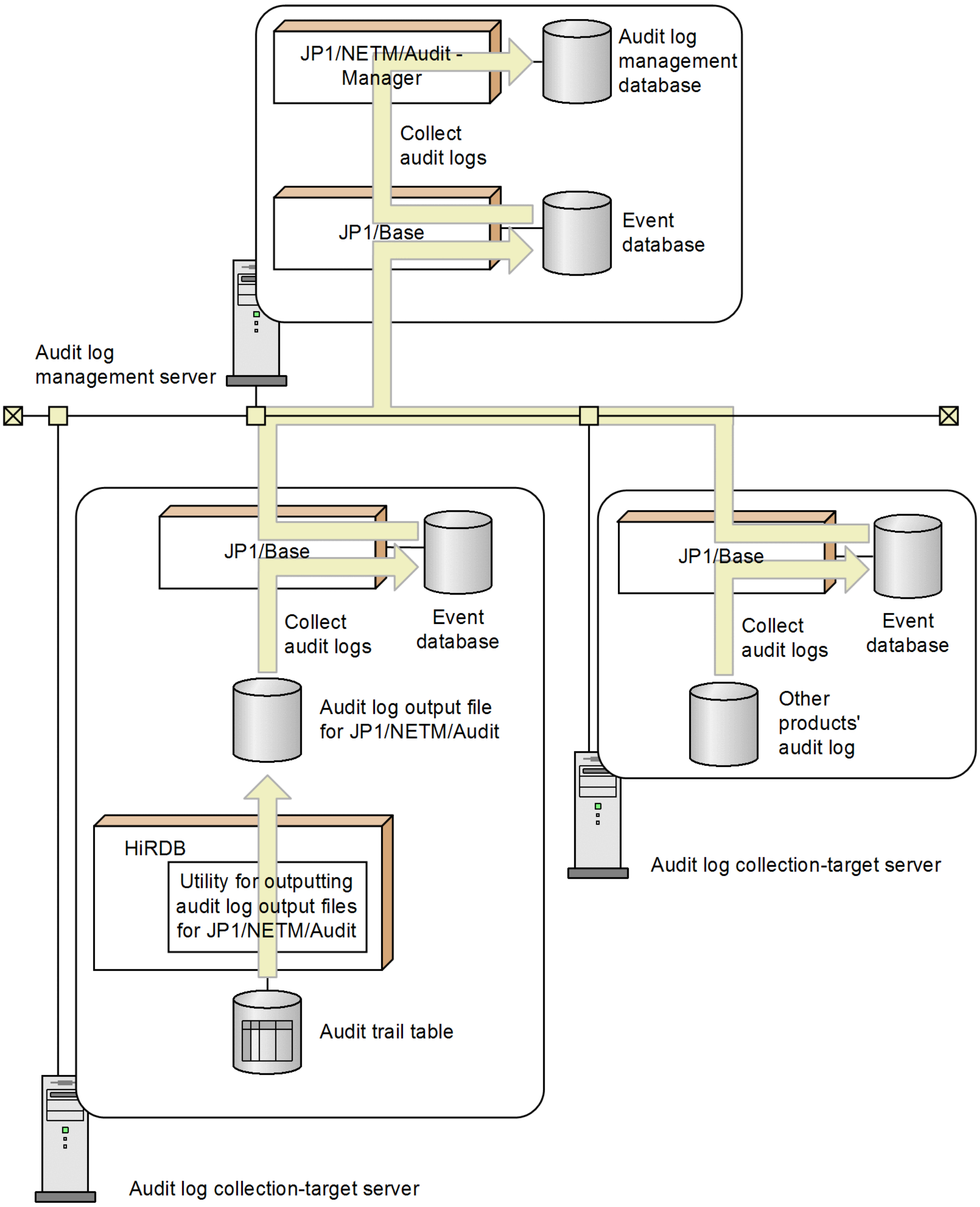
Using JP1/NETM/Audit allows you to collect and centrally manage the system's audit logs#, and maintain them for an extended period. A graphical user interface is also provided for retrieving and tallying audit logs, and for managing the backup history. Through provisioning of such tools, JP1/NETM/Audit supports evaluation and auditing of internal corporate regulations. By processing the data in the audit trail tables that are output by HiRDB for use by JP1/NETM/Audit, you can link to JP1/NETM/Audit to centrally manage the HiRDB audit trails and other products' audit logs. To process HiRDB's audit trails for JP1/NETM/Audit, you use the audit log output utility for JP1/NETM/Audit (`pdaudput` command).

#

*Audit log* is a general term for the audit trails collected from the entire system by JP1/NETM/Audit.

The following figure provides an overview of linkage with JP1/NETM/Audit.

Figure 2–12: Linkage with JP1/NETM/Audit


JP1/NETM/Audit - Manager
Audit log management database
Collect audit logs
JP1/Base
Event database
Audit log management server
JP1/Base
Collect audit logs
Event database
Audit log output file for JP1/NETM/Audit
HiRDB
Utility for outputting audit log output files for JP1/NETM/Audit
Audit trail table
Audit log collection-target server
JP1/Base
Collect audit logs
Event database
Other products' audit log
Audit log collection-target server

**Explanation**

The audit log management server (the server machine on which JP1/NETM/Audit resides) automatically collects and centrally manages the audit logs that are output to the local disk of the audit log collection-target server (the server machine on which HiRDB resides together with other products from which audit logs are to be collected).

In the audit log collection-target server where HiRDB resides, the audit log output utility for JP1/NETM/Audit (`pdaudput`) converts the necessary data in the audit trail tables so that it can be used by JP1/NETM/Audit, and then outputs it to the audit log output file for JP1/NETM/Audit. JP1/Base collects this file and stores it in the event database. The JP1/Base of the audit log management server collects audit logs from the JP1/Base of the audit log collection-target server and transfers them to JP1/NETM/Audit.

For details about linkage with JP1/NETM/Audit, see the *HiRDB Version 9 System Operation Guide*. For details about JP1/NETM/Audit, see the following manuals:

- JP1 Version 9

  *JP1 Version 9 Job Management Partner 1/NETM/Audit Setup, Description and Operation Guide*

- JP1 Version 8
  *JP1 Version 8 Job Management Partner 1/NETM/Audit*

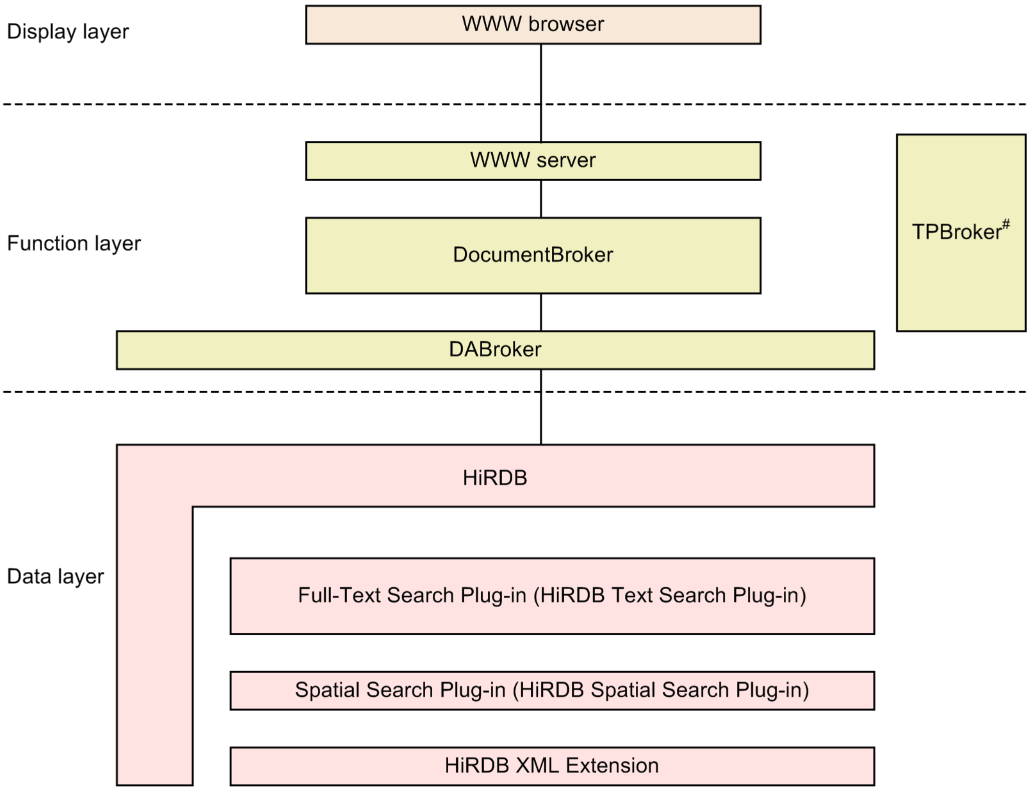
# 2.5 Linkage with products that handle multimedia information

By linking with the following products, you can build a system that supports multimedia information, such as text and spatial data, and distributed object environments:

- HiRDB Text Search Plug-in
- HiRDB Spatial Search Plug-in
- HiRDB XML Extension
- DocumentBroker
- DABroker

The following figure shows the system configuration of HiRDB-related products.

Figure 2–13: System configuration of HiRDB-related products



#: TPBroker is a product that provides communication between objects, transaction control, and operation support functions in a distributed object environment.

## (1) HiRDB Text Search Plug-in

The HiRDB Text Search Plug-in enables you to use an SQL to search both structured SGML documents and unstructured documents. In the case of searching Japanese text, the HiRDB Text Search Plug-in features an *n-gram index method* that stores the positions in which *n* characters (*n*-gram) occur in a given document so that a high-speed search of the entire document can be performed.

## (2) HiRDB Spatial Search Plug-in

The HiRDB Spatial Search Plug-in is used to retrieve spatial data (2-dimensional data), such as mapping information.

## (3) HiRDB XML Extension

When you use HiRDB XML Extension, you can handle XML data as column data. Rather than converting XML data into table format to manage it, you can simply manage the source data itself.

For XML values, you can use, for example, SQL code that processes XML values (SQL/XML) to narrow and retrieve XML documents that match a specific condition, and you can extract values that have a specific substructure.

HiRDB XML Extension provides several indexes: a substructure index called the *B-tree index*, which is used for narrowing based on predicate and uses a substructure as the key, and an XML full-text search index called the *n-gram index* for full-text searches.

## (4) DocumentBroker

The DocumentBroker product provides a basis for development and execution of applications for constructing systems that efficiently manage large volumes of documents in a distributed object environment and applied in mission-critical operations in business and in government agencies. By storing in a HiRDB database the documents to be handled by DocumentBroker and such attributes as the document creation dates, you can assure scalability, which is a feature of HiRDB, and thus construct high-performance, highly-reliable systems. By also using the HiRDB Text Search Plug-in, you can further increase the speed of document searches. DocumentBroker provides functions such as document history management and management of multiple documents through grouping.

## (5) DABroker

DABroker is a database access tool for achieving high-speed access to a large volume of data.

**Database access in a distributed object environment**

DABroker enables you to store multimedia information, such as documents, images, and audio, in a HiRDB database and access that information in a distributed object environment in a manner transparent to the specific location of the database.

**Database response improvement**

Because a multi-thread environment can be used, the amount of memory required for each connected client can be reduced. Therefore, even when the number of connected clients increases, a uniform response can be expected from the HiRDB server. Additionally, because data transfer from a client and data retrieval from the database can be executed in parallel, response performance can be improved even when the number of clients is small.

**Realization of advanced database access**

The Simple Interface facility, which consolidates complicated database access processing into eight functions, can implement focused and random searches.

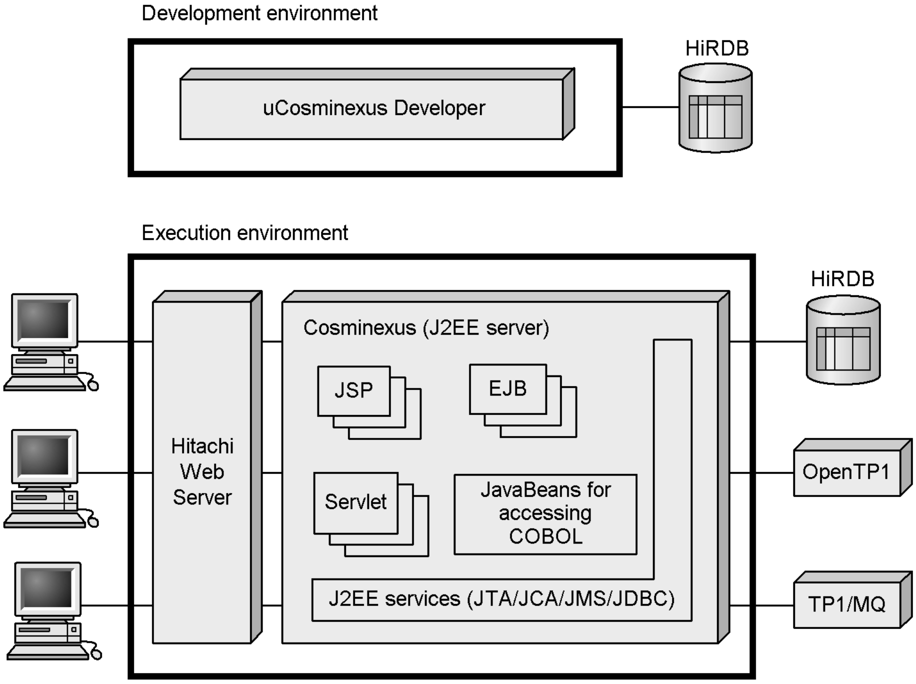
# 2.6 Linkage to Cosminexus

Linking to Cosminexus enables you to do the following:

- By using standard J2EE interfaces, such as JTA, JCA, or JMS, Cosminexus enables you to extend your legacy core systems onto a highly-reliable Web system.
- Take advantage of your legacy COBOL resources from Java applications via JavaBeans for accessing COBOL
- By linking with various uCosminexus Developer toolsets, you can improve the productivity of application development.

The following figure shows linkage to Cosminexus.

Figure 2–14: Linking to Cosminexus



uCosminexus Developer: uCosminexus Developer Professional, uCosminexus Developer Standard
EJB: Enterprise Java Beans
JCA: Java Connector Architecture
JDBC: Java DataBase Connectivity
JMS: Java Message Service
JSP: Java Server Page
JTA: Java Transaction API

# 3 Database Logical Structure

This chapter explains the logical structure of a database.

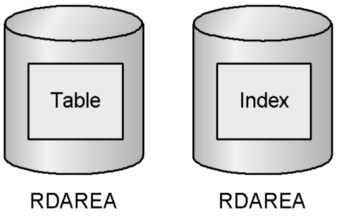
# 3.1 RDAREAs

This section describes the types of RDAREAs that exist and how to create them.

## 3.1.1 Types of RDAREAs

RDAREAs are logical areas in which tables and indexes are stored. The following figure shows the relationship between an RDAREA, a table, and an index.

Figure 3–1: Relationship between RDAREA, table, and index



The following table lists the types of RDAREAs.

Table 3–1: Types of RDAREAs

| No. | RDAREA type | Number that can be created | Required/optional |
|---|---|---|---|
| 1 | Master directory RDAREA | 1 | R |
| 2 | Data dictionary RDAREAs | 1 to 41 | R |
| 3 | Data directory RDAREA | 1 | R |
| 4 | Data dictionary LOB RDAREAs | 2 | O |
| 5 | User RDAREAs | 1 to 8,388,589 | R |
| 6 | User LOB RDAREAs | 1 to 8,388,325 | O |
| 7 | Registry RDAREA | 1 | O |
| 8 | Registry LOB RDAREA | 1 | O |
| 9 | List RDAREAs | 1 to 8,388,588 | O |

R: Required RDAREA

O: Optional RDAREA

**Notes**:

- The RDAREAs in Nos. 1 through 3 are called *system RDAREAs*.
- The RDAREAs in Nos. 4, 6, and 8 are called *LOB RDAREAs*.

### (1) Master directory RDAREA

The master directory RDAREA, which is required, stores internal system information.

### (2) Data dictionary RDAREA

A data dictionary RDAREA stores dictionary tables and indexes for dictionary tables. Dictionary tables enable users to conduct HiRDB searches. For details about dictionary tables, see the *HiRDB Version 9 UAP Development Guide*.

There must be at least one data dictionary RDAREA.

## (3) Data directory RDAREA

The data directory RDAREA, which is required, stores internal system information.

## (4) Data dictionary LOB RDAREA

The data dictionary LOB RDAREAs store definition sources and objects for stored procedures and stored functions. When you define a trigger, the data dictionary LOB RDAREA also stores the internally generated SQL object of that trigger. When you use a stored procedure or a stored function, or when you define a trigger, make sure to create a data dictionary LOB RDAREA.

The two data dictionary LOB RDAREAs are:

- Data dictionary LOB RDAREA that stores the definition sources
- Data dictionary LOB RDAREA that stores the SQL objects

Reference note

> Triggers also include triggers that HiRDB generates internally based on referential constraint actions. Therefore, you must also create a data dictionary LOB RDAREA when you define a referential constraint.

## (5) User RDAREA

User RDAREAs, which are required, store tables and indexes. A table can be stored in one or in multiple user RDAREAs. Similarly, a user RDAREA can store one or multiple tables. The same applies to indexes. There are two types of user RDAREAs:

**Public RDAREAs**

All users who are registered in HiRDB can use the public RDAREAs.

**Private RDAREAs**

Only users who have the requisite privilege to access a private RDAREA can access that RDAREA.

User RDAREAs can be either shared RDAREAs, which can be referenced from all back-end servers, or temporary table RDAREAs, which store temporary tables and temporary table indexes. For details about shared RDAREAs and temporary table RDAREAs, see the *HiRDB Version 9 Installation and Design Guide*.

## (6) User LOB RDAREA

User LOB RDAREAs store long variable-size data, such as documents, images, and audio. A user LOB RDAREA is required in order to use any of the following types of data:

- A column specified as a BLOB-type column (LOB column)
- An attribute specified as BLOB type in an abstract data type
- Plug-in index

If a table contains multiple types of data (LOB column, BLOB-type attribute in an abstract data type, or plug-in for a plug-in index), each type of data must be stored in a separate user LOB RDAREA. There are two types of user LOB RDAREAs:

**Public RDAREAs**

All users who are registered in HiRDB can use the public RDAREAs.

**Private RDAREAs**

Only users who have the requisite privilege to access a private RDAREA can access that RDAREA.

## (7) Registry RDAREA

The registry RDAREA stores registry information. If plug-ins that use the registry facility are used, it is necessary to create a registry RDAREA. When the registry facility is used, both a registry RDAREA and a registry LOB RDAREA must be created. The registry facility is required in order to use the HiRDB Text Search Plug-in. Some plug-ins do not require the registry facility or a registry RDAREA.

### (8) Registry LOB RDAREA

The registry LOB RDAREA stores registry information keys whose key length exceeds 32,000 bytes. If you use plug-ins that use the registry facility, you must create a registry LOB RDAREA. The registry facility is required in order to use the HiRDB Text Search Plug-in. Some plug-ins do not require the registry facility or a registry LOB RDAREA.

### (9) List RDAREA

A list RDAREA stores lists created by the `ASSIGN LIST` statement. A list RDAREA is required in order to perform a narrowed search.

## 3.1.2 RDAREA creation

The following facilities can be used to create RDAREAs:

- When HiRDB is first installed
  If you use either of the following environment setup support tools, RDAREAs can be created when you run the tool:
  - Simple setup tool
  - Batch file (`SPsetup.bat`) (Windows edition only)

  If you use a command to set up the HiRDB environment, you can use the following utility to create RDAREAs:
  - Database initialization utility (`pdinit`)
- When adding RDAREAs
  You can use either of the following utilities to create (add) RDAREAs:
  - Database structure modification utility (`pdmod`)
  - Registry facility initialization utility (`pdreginit`)

For details about RDAREA creation, see the *HiRDB Version 9 Installation and Design Guide* or *HiRDB Version 9 System Operation Guide*.

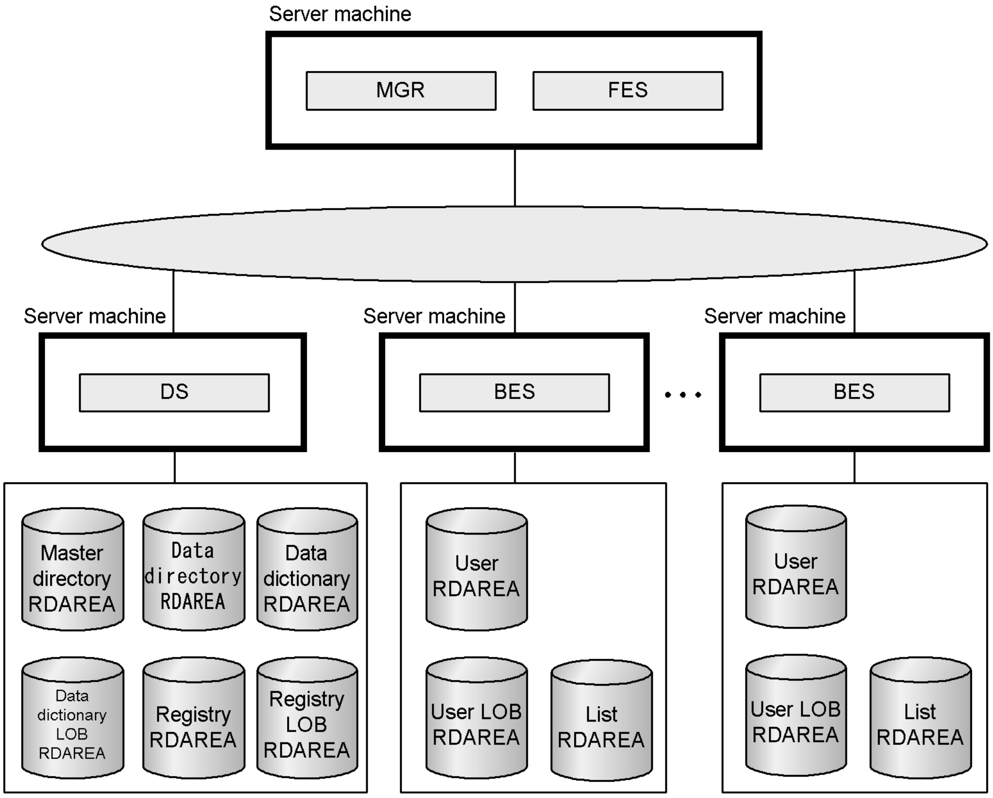
Tip

- Pay particular attention to the value of the `pd_max_rdarea_no` operand. This operand specifies the maximum number of RDAREAs. If the number of RDAREAs exceeds the value of this operand, you can no longer start HiRDB normally.
- For a HiRDB parallel server configuration, there are restrictions concerning the servers on which RDAREAs can be created, as shown in the table below. The figure following the table shows an example of an RDAREA configuration (for a HiRDB parallel server configuration).

Table 3–2: Server machines on which RDAREAs can be created (for a HiRDB parallel server configuration)

| Type of RDAREA | Server machine on which RDAREA is created |
|---|---|
| Master directory RDAREA | Server machine on which the dictionary server is running. |
| Data dictionary RDAREA | |
| Data directory RDAREA | |
| Master directory LOB RDAREA | |
| Registry RDAREA | |
| Registry LOB RDAREA | |
| User LOB RDAREA | Server machine on which a back-end server is running. |
| List RDAREA | |
| User RDAREA | |

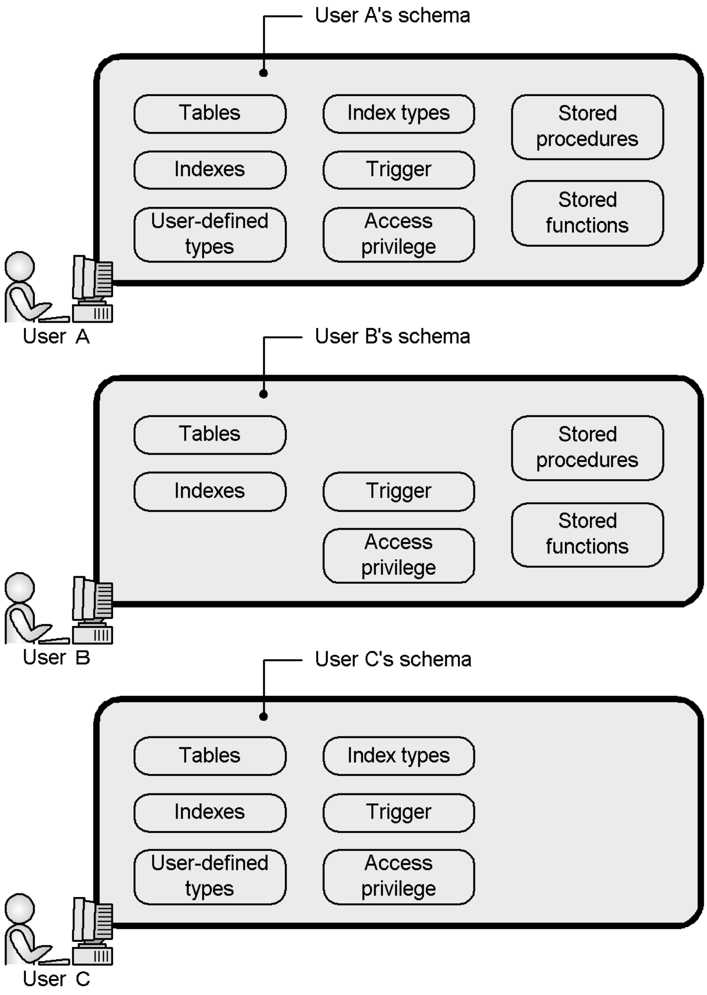
Figure 3–2: RDAREA configuration example (for a HiRDB parallel server configuration)



# 3.2 Schemas

Tables, indexes, abstract data types (user-defined types), index types, stored procedures, stored functions, triggers, and access privileges are subsumed under a concept known as a *schema*. Before you can define any of these resources, you must define a schema. To define a schema, you use the definition SQL `CREATE SCHEMA` statement. One schema can be defined for each user. The following figure illustrates the concept of schemas.

Figure 3–3: Concept of schemas



# 3.3 Tables

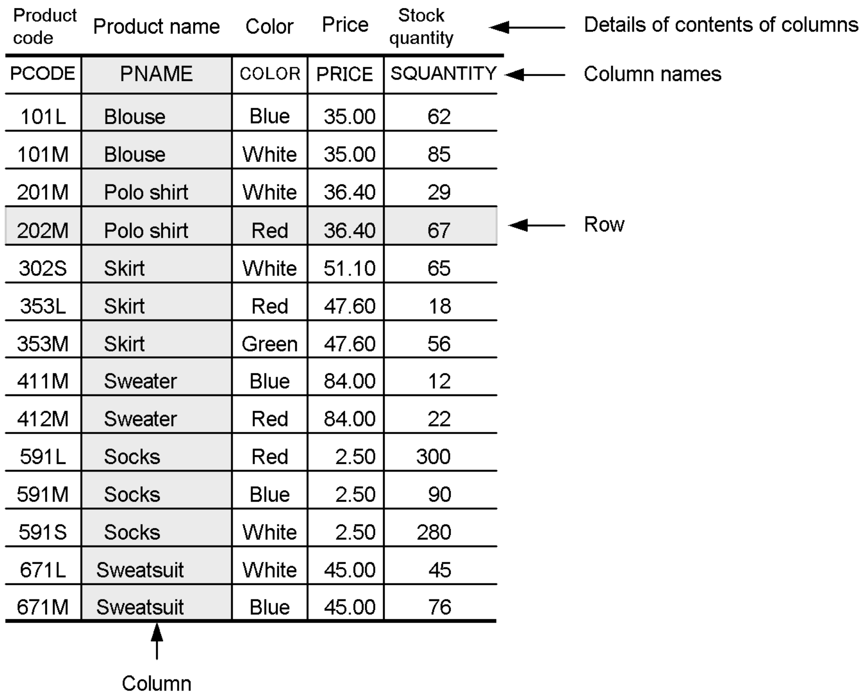
A table in HiRDB is a relational database with a tabular structure composed of rows and columns.

## 3.3.1 Basic table structure

### (1) Rows and columns

The cells in a table's horizontal plane constitute *rows*; the cells in the vertical plane constitute *columns*. The row is the unit for performing operations on a table. Each row consists of data entries from one or more columns. Each column is given a *column name*. When an operation is performed on a table, you identify the position of an item in a row by its column name. The following figure shows the structure of a table.

Figure 3–4: Example of a table



| Product code | Product name | Color | Price | Stock quantity |
|---|---|---|---|---|
| PCODE | PNAME | COLOR | PRICE | SQUANTITY |
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

**Explanation**

This example illustrates the structure of an inventory table (the table name is `STOCK`). The columns are defined with the `CREATE TABLE` in the definition SQL. Following is the definition of the `STOCK` table's columns:

```
CREATE TABLE STOCK
    (PCODE CHAR(4),
     PNAME NCHAR(8),
     COLOR NCHAR(1)
     PRICE INTEGER,
     SQUANTITY INTEGER);
```

### (2) Data types

A data type must be specified for each column and for each attribute comprising an abstract data type. Data types are classified broadly as predefined data types and user-defined data types. A predefined data type is provided by HiRDB. A user-defined data type is defined by a user as needed. The following table lists the available data types.

Table 3–3: Available data types

| Classification | | Data type | Data format |
|---|---|---|---|
| Predefined | Numeric data | INTEGER | Integer |
| | | SMALLINT | Integer |
| | | LARGE DECIMAL | Fixed point |
| | | FLOAT | Double-precision floating point |
| | | SMALLFLT | Single-precision floating point |
| | Character data | CHARACTER | Fixed-size character string |
| | | VARCHAR | Variable-size character string |
| | National character data | NCHAR | Fixed-size national character string |
| | | NVARCHAR | Variable-size national character string |
| | Mixed character data | MCHAR | Fixed-size mixed character string |
| | | MVARCHAR | Variable-size mixed character string |
| | Date data | DATE | Date |
| | Time data | TIME | Time |
| | Time-stamp data | TIMESTAMP | Precision for time-stamp data in floating-point seconds |
| | Date interval data | INTERVAL YEAR TO DAY | Date interval |
| | Time interval data | INTERVAL HOUR TO SECOND | Time interval |
| | Large object data | BLOB | Binary data string |
| | Binary data | BINARY | Binary data string |
| | Boolean data | BOOLEAN | Boolean value |
| User-defined | Abstract data type | -- | -- |

--: Not applicable

Data types are specified when a table is created with the `CREATE TABLE` of the definition SQL. The data type for an attribute comprising an abstract data type is specified with the `CREATE TYPE` of the definition SQL. For details about abstract data types, see Section *3.5 Expansion into an object relational database*.

## 3.3.2 Table normalization

The operation of removing redundant data from one table into another table is called *table normalization*. Use table normalization to improve the storage efficiency of table data as well as to enhance the capacity for concurrent executions during access processing. By normalizing a given table repeatedly, it becomes possible to transform a complex database into an optimal database. The figure below illustrates table normalization. For details about table normalization, see the *HiRDB Version 9 Installation and Design Guide*.

Figure 3–5: Table normalization

● Before normalization

STOCK

| Product number | Product code | Product name | Price | Stock quantity |
|---|---|---|---|---|
| PNO | PCODE | PNAME | PRICE | SQUANTITY |
| 01010 | 101 | Blouse | 35.00 | 62 |
| 01011 | 101 | Blouse | 35.00 | 85 |
| 02021 | 202 | Polo shirt | 36.40 | 67 |
| 03530 | 353 | Skirt | 47.60 | 18 |
| 03531 | 353 | Skirt | 47.60 | 56 |
| 04121 | 412 | Sweater | 84.00 | 22 |
| 05910 | 591 | Socks | 2.50 | 300 |
| 05911 | 591 | Socks | 2.50 | 90 |
| 05912 | 591 | Socks | 2.50 | 280 |
| 06710 | 671 | Sweatsuit | 45.00 | 45 |
| 06711 | 671 | Sweatsuit | 45.00 | 76 |

These two columns always contain corresponding values (1-to-1 correspondence).
Thus, these two columns are redundant.

● After normalization

STOCK

| PNO | PNAME | PRICE | SQUANTITY |
|---|---|---|---|
| 01010 | Blouse | 35.00 | 62 |
| 01011 | Blouse | 35.00 | 85 |
| 02021 | Polo shirt | 36.40 | 67 |
| 03530 | Skirt | 47.60 | 18 |
| 03531 | Skirt | 47.60 | 56 |
| 04121 | Sweater | 84.00 | 22 |
| 05910 | Socks | 2.50 | 300 |
| 05911 | Socks | 2.50 | 90 |
| 05912 | Socks | 2.50 | 280 |
| 06710 | Sweatsuit | 45.00 | 45 |
| 06711 | Sweatsuit | 45.00 | 76 |

PRODUCT

| PCODE | PNAME |
|---|---|
| 101 | Blouse |
| 202 | Polo shirt |
| 353 | Skirt |
| 412 | Sweater |
| 591 | Socks |
| 671 | Sweatsuit |

**Explanation**

Before normalization, the columns PCODE and PNAME in the STOCK table have a 1-to-1 correspondence; the information in these two columns is redundant. In such a case, you can create another table, named PRODUCT in this example, composed of the PCODE and PNAME columns from the STOCK table, so that in the PRODUCT table there is no duplication of information in the PCODE and PNAME columns.

## 3.3.3 FIX attribute

The attribute that is assigned to a table whose row size is fixed is called the *FIX attribute*. Specifying the FIX attribute for a fixed-size table that contains no null values speeds up column retrieval processing. Use of this technique improves access performance even when there is a large number of columns. Specify the FIX attribute when all the following conditions are satisfied:

- No columns will be added
- No columns contain the null value
- There are no variable-size columns

The FIX attribute is assigned to a table by specifying the FIX option in the CREATE TABLE of the definition SQL. For details about the FIX attribute, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.3.4 Primary key

A *primary key* makes it possible to uniquely identify the rows in a table. The columns constituting a defined primary key are subject to the *uniqueness constraint* and the *NOT NULL constraint*. The uniqueness constraint means that no value can be duplicated in the key area (column or set of columns); in other words, there is a constraint that all data entries in the key columns must be unique. The NOT NULL constraint means that the null value is not permitted as a value in the key columns.

If there are one or more columns or sets of columns (called *candidate keys*) on the basis of which the rows in a table can be uniquely identified, select a primary key from among the candidate keys. Define a primary key that is mnemonically significant and appropriate for being assigned the uniqueness and NOT NULL constraints.

The primary key is defined with the PRIMARY KEY option in the CREATE TABLE. For details about the primary key, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.3.5 Cluster key

You can store the rows in a table in either ascending or descending order. To do this, you need to define a *cluster key*. Using a cluster key for storing the rows in a table in ascending or descending order reduces the data I/O time for the following types of operations:

- Searching, updating, or deleting rows by specifying a range
- Searching or updating rows in cluster key sequence

**Applicability**

Specify a cluster key when all the following conditions are satisfied:

- Data will be accumulated in either ascending or descending order of the key values and the data will be accessed frequently in key value sequence.
- No key values are to be changed.
- The table's rows have a fixed row size.

A cluster key is defined with the CLUSTER KEY option in the CREATE TABLE. For details about the cluster key, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.3.6 Suppress option

The *suppress option* refers to abbreviating some of data in a table so that the data can be stored in a smaller space than its actual size. When the suppress option is specified, the system stores any decimal data in the table by taking only the significant digits (ignoring leading zeros) and a value indicating the actual data size. Because the suppress option stores data by truncating it to less than its actual size, less disk space is used and I/O time for text and other search operations is reduced.

The suppress option is specified with the SUPPRESS option in the CREATE TABLE. For details about the suppress option, see the *HiRDB Version 9 Installation and Design Guide*.

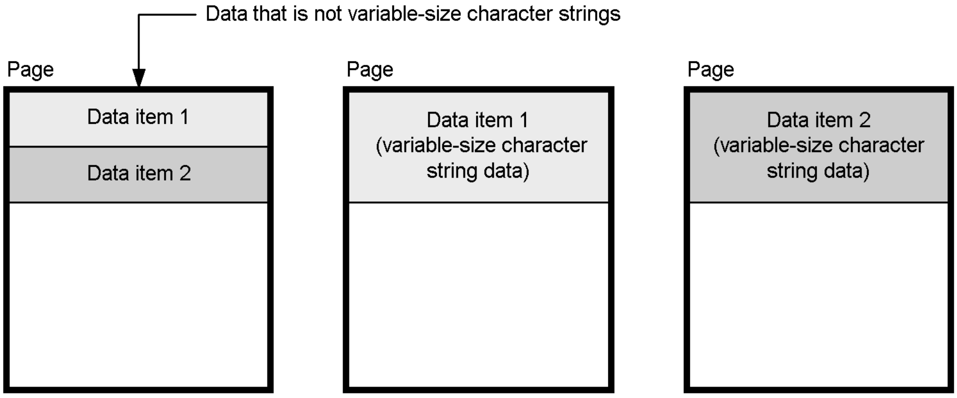
## 3.3.7 No-split option

When any of the following data types is defined in a table and the actual size of the data of any of these data types is equal to or greater than 256 bytes, the normal procedure results in each row of data being stored on multiple pages:

- `VARCHAR`
- `MVARCHAR`
- `NVARCHAR`

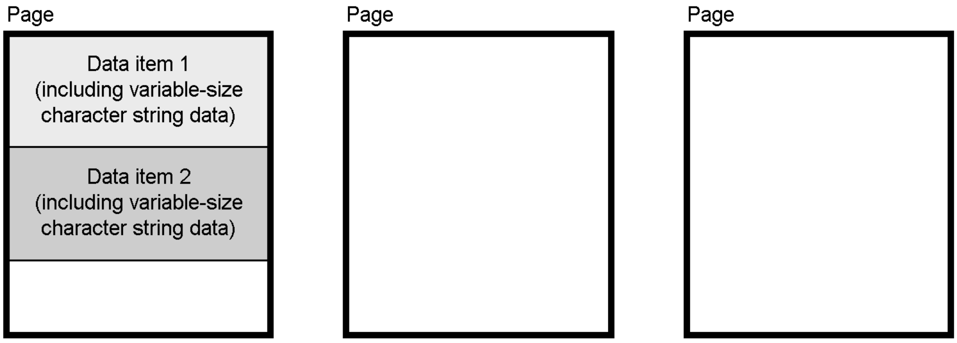
The following figure shows the data storage method that is used normally.

Figure 3–6: Normal data storage method when the actual data size of a variable-size character string is at least 256 bytes



As illustrated, data that is not part of variable-size character strings and data belonging to variable-size character string are stored on different pages, which reduces the data storage efficiency. In this case, you can improve the data storage efficiency by specifying the no-split option. When the no-split option is specified, the system stores each entire row on the same page, even if the actual data size of a variable-size character string is equal to or greater than 256 bytes. The following figure shows the data storage method that is used when the no-split option is specified.

Figure 3–7: Data storage method used when the no-split option is specified



**Explanation**

The no-split option stores all data from a row on the same page, which improves the data storage efficiency as compared to when the no-split option is not specified.

The no-split option is specified with the `NO SPLIT` option in the `CREATE TABLE` or `CREATE TYPE`. For details about the no-split option, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.3.8 Table row partitioning

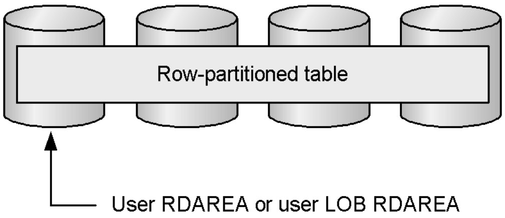
The process by which a single table is split amongst multiple user RDAREAs or user LOB RDAREAs is called *table row partitioning*. A table that is partitioned by this method is called a *row-partitioned table*. When you row-partition a table, you can use user RDAREAs or user LOB RDAREAs as the unit for storing the table's data, reorganizing the table, and making backups.

For example, you could row-partition a table to match the different types of UAPs (job types) that will access the table and then store the sections appropriate to the various UAP types in different RDAREAs. Then, when it becomes necessary to make a backup, you need stop only the UAPs that are accessing the RDAREA that is to be backed up, which facilitates overall system operation.

In the case of a HiRDB parallel server configuration, it is possible to access concurrently a table in user RDAREAs or user LOB RDAREAs under multiple back-end servers, resulting in high-speed table access and better load distribution.

The following figure shows table row partitioning.

Figure 3–8: Table row partitioning



There are two methods for row-partitioning a table:

- Key range partitioning
- Hash partitioning (flexible hash partitioning and FIX hash partitioning)

### (1) Key range partitioning

Key range partitioning is when ranges of values in a specified column are used as the conditions for row-partitioning the table. The specific column that provides the conditions for row partitioning is called the *partitioning key*. Key range partitioning is used when the actual RDAREAs in which table data is stored must be known. There are two ways to specify this type of row partitioning:

- Storage condition specification
- Boundary value specification

#### (a) Storage condition specification

Comparison operators are used to specify conditions for selecting the data to be stored in specific RDAREAs. Only one range of values, as specified by a storage condition, can be set in each RDAREA. The following figure shows an example of key range partitioning with a storage condition specified.

Figure 3–9: Key range partitioning: Example of storage condition specification

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

To be stored in RDAREA01

To be stored in RDAREA02

Partitioning key

- ● Row-partitioned table stored in RDAREA01

STOCK (data in rows 101L-353M)

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |

- ● Row-partitioned table stored in RDAREA02

STOCK (data in rows 411M-617M)

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

**Explanation**

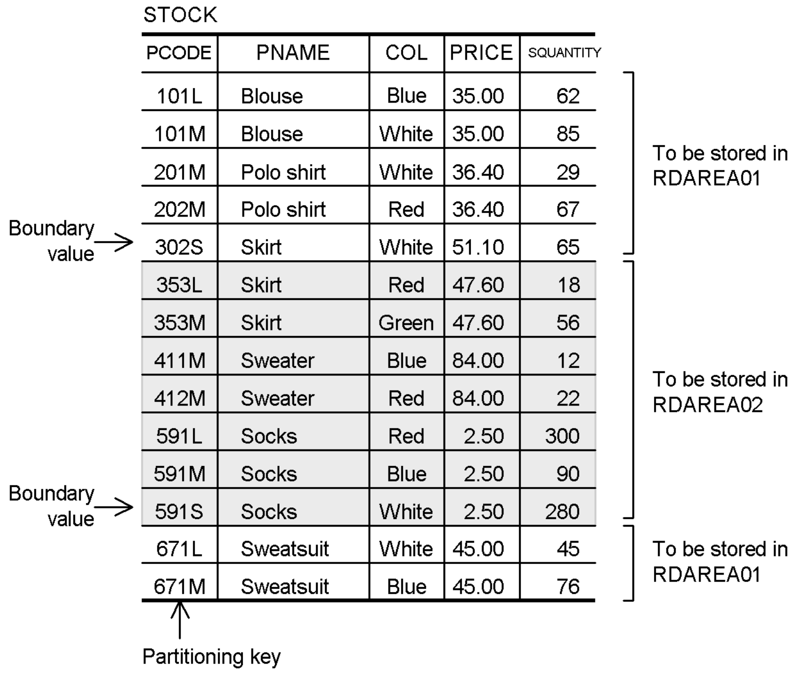
The STOCK table is row-partitioned using PCODE as the partitioning key; the results of row-partitioning are stored in RDAREA01 and RDAREA02:

```
CREATE TABLE STOCK
   (PCODE CHAR(4) NOT NULL,PNAME NCHAR(8),
   COLOR NCHAR(1),PRICE INTEGER,SQUANTITY INTEGER
   )IN ((RDAREA01)PCODE<='353M',(RDAREA02));
```

### (b) Boundary value specification

Boundary values for the data to be stored in each RDAREA are specified with literals in ascending order. Multiple ranges delimited by boundary values can be specified for each RDAREA. The following figure shows an example of key range partitioning with boundary values specified.

Figure 3–10: Key range partitioning: Example of boundary value specification



| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

● Row-partitioned table stored in RDAREA01

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

● Row-partitioned table stored in RDAREA02

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |

**Explanation**

The STOCK table is row-partitioned using PCODE as the partitioning key; the results of row-partitioning are stored in RDAREA01 and RDAREA02:

```
CREATE TABLE STOCK
   (PCODE CHAR(4) NOT NULL,PNAME NCHAR(8),
   COLOR NCHAR(1),PRICE INTEGER,SQUANTITY INTEGER
   )PARTITIONED BY PCODE
   IN ((RDAREA01)'302S',(RDAREA02)'591S',(RDAREA01));
```

## (2) Hash partitioning

Hash partitioning is when a table is row-partitioned by using a hash function to store evenly among the storage RDAREAs the values contained in the columns that make up the table. The specific column that provides the conditions for row-partitioning is called the *partitioning key*. Hash partitioning is used when it is necessary to store the same amount of table data is each of the RDAREAs. The following table provides an overview of the two types of hash partitioning.

Table 3–4: Types of hash partitioning

| Hash partitioning type | Explanation |
|---|---|
| Flexible hash partitioning | The RDAREAs in which the table is stored cannot be determined. Therefore, a search process must check all back-end servers that might contain parts of the table. |
| FIX hash partitioning | HiRDB keeps track of the RDAREAs in which the table is stored. Consequently, a search process has to check only the back-end servers that are expected to contain the table's the data. |

The following figure shows an example of hash partitioning.

Figure 3–11: Example of hash partitioning

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671L | Sweatsuit | White | 45.00 | 45 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

↑
Partitioning key

● Row-partitioned table stored in RDAREA01

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| 671M | Sweatsuit | Blue | 45.00 | 76 |

● Row-partitioned table stored in RDAREA02

STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 591L | Socks | Red | 2.50 | 300 |
| 671L | Sweatsuit | White | 45.00 | 45 |

**Explanation**

The STOCK table is row-partitioned using PCODE as the partitioning key; the results of row-partitioning are stored in RDAREA01 and RDAREA02:

Note that the actual RDAREAs in which the data is stored might differ from those in the example.

```
CREATE TABLE STOCK
   (PCODE CHAR(4) NOT NULL,PNAME NCHAR(8),
   COLOR NCHAR(1),PRICE INTEGER,SQUANTITY INTEGER
   ) [FIX]# HASH HASH6 BY PCODE
   IN (RDAREA01,RDAREA02);
```

#: This is specified for FIX hash partitioning.

## (3) Examples of table row partitioning

Allocate RDAREAs into which a row-partitioned table is stored onto different disks. If they are allocated on the same disk, contention for access to these RDAREAs can occur, resulting in decreased performance.

The concepts of *row partitioning among servers* and *row partitioning within a server* apply to a HiRDB parallel server configuration. Row-partitioning among servers is the mode in which a table is row-partitioned over multiple back-end servers. Row-partitioning within a server is the mode in which a table is row-partitioned within a single back-end server. In the case of a HiRDB single server configuration, row-partitioning is always within the server.

The figures below show examples of table row partitioning for a HiRDB single server configuration and a HiRDB parallel server configuration.

Figure 3–12: Example of table row partitioning: HiRDB single server configuration

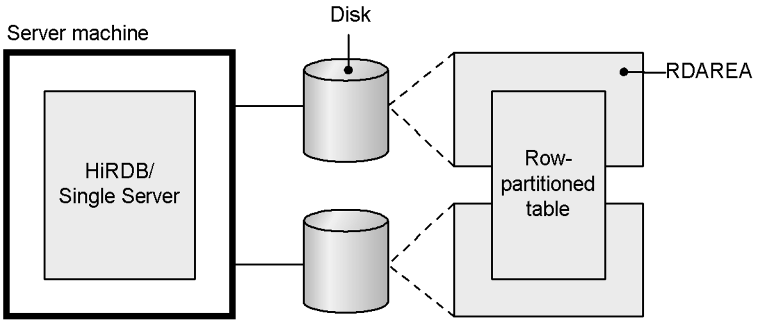
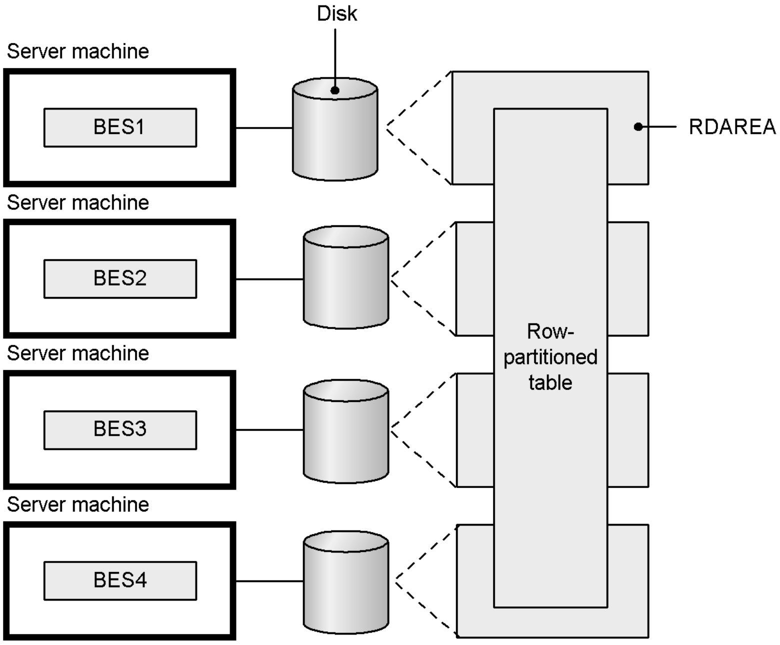


Figure 3–13: Example of table row partitioning: HiRDB parallel server configuration



**Explanation**

The table is row-partitioned among back-end servers BES1 to BES4.

## (4) Table row partitioning definition

To row-partition a table, the elements listed below must be specified in the `CREATE TABLE` definition SQL:

- Allocation of the table to RDAREAs
- Row-partitioning method (specification of storage conditions, boundary values, or hash partitioning)

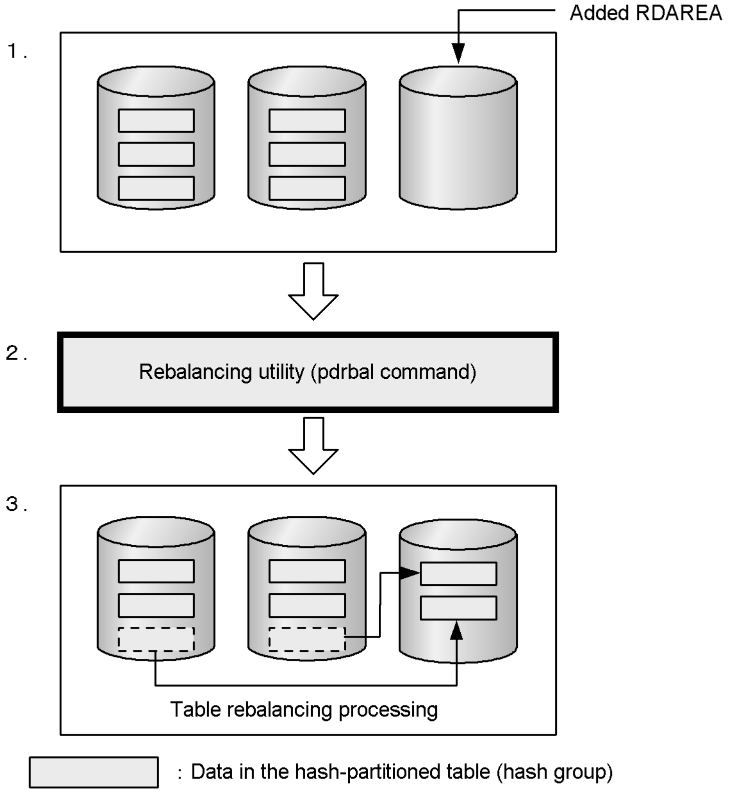
For details about the row-partitioning design considerations for improving processing performance, see the *HiRDB Version 9 Installation and Design Guide*.

## (5) Hash facility for hash row partitioning

When new RDAREAs are added to accommodate an increase in the volume of data in a hash-partitioned table (increase in the number of table partitions), there might be significant differences in the amount of data stored in the existing RDAREAs and in the new RDAREAs. The *hash facility for hash row partitioning* corrects this sort of imbalance in the amount of data in RDAREAs as a result of increasing the number of table partitions. The following figure illustrates the hash facility for hash row partitioning. This facility can be applied to both FIX hashing and flexible hashing.

For details about the hash facility for hash row partitioning, see the *HiRDB Version 9 System Operation Guide*.

Figure 3–14: Hash facility for hash row partitioning


Added RDAREA
1.
2.
Rebalancing utility (pdrbal command)
3.
Table rebalancing processing
: Data in the hash-partitioned table (hash group)

**Explanation**

1. Because the hash-partitioned table became filled with data, an additional RDAREA for storing table data was provided (the number of table partitions was increased). No data is placed in the new RDAREA, which creates a data volume imbalance.
2. The *rebalancing utility* (`pdrbal` command) is executed in order to correct the data volume imbalance.
3. Execution of the rebalancing utility moves and re-allocates data in units of hash groups in a process called *table rebalancing*. The hash facility for hash row partitioning causes HiRDB to divide data into 1,024 groups (called *hash groups*) based upon the result of hashing the primary key. For each group, an RDAREA segment is allocated, and the data is stored. This reallocation of data is also performed in units of hash groups.

**Application criteria**

- An increase in the amount of data to be stored is anticipated, so an RDAREA is needed for potential future use.#
- Because of the large amount of data in the table, it would be prohibitively difficult to re-create the table.

#: Although normally an RDAREA cannot be added to a FIX hash-partitioned table in which data is already stored, you can use the hash facility for hash row partitioning to add an RDAREA.

**Operating procedure**

Following is a summary of the operating procedure for using the hash facility for hash row partitioning:

1. When a hash partitioned table is defined, use one of the hash functions A to F to define the table as a *rebalancing table*.
2. Add a table storage RDAREA in order to increase the number of table partitions for the table.
3. Execute the rebalancing utility to rebalance the table.

## 3.3.9 Table matrix partitioning

Partitioning a table by a combination of partitioning methods using two table columns as partitioning keys is called *matrix partitioning*. The first column used as a partitioning key is called the *first dimension partitioning column*, and the second column used as a partitioning key is called the *second dimension partitioning column*. Matrix partitioning involves key range partitioning with boundary values specified for the first dimension partitioning column and then partitioning the resulting data further by the second dimension partitioning column. The following partitioning methods can be specified for the second dimension partitioning column:

- Key range partitioning with boundary values specified
- Flexible hash partitioning
- FIX hash partitioning

A table that has been matrix-partitioned is called a *matrix-partitioned table*. You can also perform matrix partitioning on indexes that have been mapped to a matrix-partitioned table. Note that you must have HiRDB Advanced High Availability to perform matrix partitioning on a table.

### (1) Benefits of table matrix partitioning

The following describes the benefits that are provided by using multiple columns as partitioning keys to split a table:

- Increase in SQL processing speed
  You can execute SQL processes in parallel and more quickly perform searches by using multiple keys to narrow the searched range.
- Decrease in operation time
  Matrix partitioning creates smaller partitions, which decreases the size of any single RDAREA, and reduces the time required to perform operations such as reorganization, backup, and failure recovery.

### (2) Application criteria

We recommend using key range partitioning with boundary values specified for both partitioning columns when the following conditions are met:

- Partitioning by the first dimension partitioning column results in a vast amount of data within each set of boundary values.
- Multiple columns need to be specified in the search condition for a UAP that accesses the table and you wish to limit the RDAREAs that are accessed by multiple columns. Or, you wish to limit the RDAREAs that are accessed only by column *n* specified in the SQL statement.

When the following conditions are met, we recommend that you combine key range partitioning with boundary values specified and hash partitioning:

- Partitioning by the first dimension partitioning column results in a vast amount of data within each set of boundary values.
- You wish to uniformly segment the range of data that was partitioned by the first dimension partitioning column.

### (3) How matrix partitioning is defined

To define matrix partitioning, specify the following in the `PARTITIONED BY MULTIDIM` operand of the `CREATE TABLE` definition SQL:

- Allocation of a table to an RDAREA
- The matrix partitioning method (partitioning key, partitioning method)

## (4) Examples of matrix partitioning

### (a) Key range partitioning with boundary values specified is used for the second dimension partitioning column

In this example, boundary values are specified for the registration date (`RDATE`) and the store number (`SNUM`) in a customer data table. The table is then matrix partitioned such that the user data is stored by registration date and store number in the user RDAREAs `USR01` to `USR06`. In this example, the number of user RDAREAs needed to store this data is (*the number of boundary values for RDATE* + 1) × (*the number of boundary values for SNUM* + 1), so the number of user RDAREAs needed is 3 × 2 = 6.

| RDATE | SNUM | |
|---|---|---|
| | 100 or less | 101 or greater |
| 2000 and earlier | `USR01` | `USR02` |
| 2001 | `USR03` | `USR04` |
| 2002 and later | `USR05` | `USR06` |

The following shows the SQL code used to matrix partition the table:

```
CREATE FIX TABLE CTBL
   (RDATE DATE, SNUM INT, CNAME NCHAR(10))
  PARTITIONED BY MULTIDIM(
  RDATE (('2000-12-31'),('2001-12-31')), . . .1.
  STORE_NO ((100))                          . . .2.
  )IN ((USR01,USR02),(USR03,USR04),(USR05,USR06))
```

**Explanation**

1. Specifies the name of the first dimension partitioning column (name of the column that is used as the first partitioning key), and its list of boundary values.
2. Specifies the name of the second dimension partitioning column (name of the column that is used as the second partitioning key), and its list of boundary values.

The following figure shows an example of matrix partitioning.

Figure 3–15: Example of matrix partitioning (combining with key range partitioning with boundary values specified)

CTBL

| RDATE | SNUM | CNAME |
|---|---|---|
| 1999-10-31 | 001 | Ed Kimura |
| 2000-01-25 | 110 | Sam Creek |
| 2001-06-30 | 005 | Tim Crane |
| 2001-12-24 | 120 | Mike Sheen |
| 2002-01-07 | 050 | Ken Stone |
| 2002-06-13 | 130 | Tom Yancy |
| 2002-08-24 | 099 | Kevin Yang |

↑ First partitioning key (RDATE)
↑ Second partitioning key (SNUM)

- Matrix-partitioned table stored in USR01

| RDATE | SNUM | CNAME |
|---|---|---|
| 1999-10-31 | 001 | Ed Kimura |

- Matrix-partitioned table stored in USR02

| RDATE | SNUM | CNAME |
|---|---|---|
| 2000-01-25 | 110 | Sam Creek |

- Matrix-partitioned table stored in USR03

| RDATE | SNUM | CNAME |
|---|---|---|
| 2001-06-30 | 005 | Tim Crane |

- Matrix-partitioned table stored in USR04

| RDATE | SNUM | CNAME |
|---|---|---|
| 2001-12-24 | 120 | Mike Sheen |

- Matrix-partitioned table stored in USR05

| RDATE | SNUM | CNAME |
|---|---|---|
| 2002-01-07 | 050 | Ken Stone |
| 2002-08-24 | 099 | Tom Yancy |

- Matrix-partitioned table stored in USR06

| RDATE | SNUM | CNAME |
|---|---|---|
| 2002-06-13 | 130 | Kevin Yang |

### (b) Hash partitioning is used for the second dimension partitioning column

This example uses FIX hash partitioning for the second dimension partitioning column.

In this example, boundary values are specified for the registration dates in a customer table, and then a hash function is used to separate the data by store number and region code into three segments. This results in a table that is matrix partitioned such that each segment of customer data is stored in one of the user RDAREAs (USR01 to USR09) shown below. The number of user RDAREAs needed to store the resulting data is (*number of boundary values* + 1) × (*desired partitions to be obtained by hash function*). In this example, the number is 3 × 3 = 9.

| Registration date | Store number and region code (divided into 3 partitions by hash function) | | |
|---|---|---|---|
| 2002 and earlier | USR01 | USR02 | USR03 |
| 2003 | USR04 | USR05 | USR06 |
| 2004 and later | USR07 | USR08 | USR09 |

The following code shows the SQL statement used to define the table to be matrix partitioned:

```
CREATE FIX TABLE customer-table
  (registration-date DATE, store-number INT, region-code INT, customer-name
NCHAR(10))
  PARTITIONED BY MULTIDIM
  (registration-date (('2002-12-31'),('2003-12-31')), ...1.
  FIX HASH HASH6 BY store-number, region-code          ...2.
  )IN ((USR01,USR02,USR03),
       (USR04,USR05,USR06),
       (USR07,USR08,USR09))
```

**Explanation**

1. Specifies the name of the first dimension partitioning column (name of the column to be used as the first partitioning key) and its list of boundary values.
2. Specifies the name of the second dimension partitioning column (name of the column to be used as the second partitioning key) and the name of the hash function.

The following figure shows an example of matrix partitioning.

Figure 3–16: Example of matrix partitioning (combination of key range partitioning with boundary values specified and hash partitioning)

CTBL

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2002-01-31 | 001 | 01 | Ed Kimura |
| 2002-05-25 | 110 | 03 | Sam Creek |
| 2002-06-30 | 005 | 01 | Tim Crane |
| 2003-01-24 | 120 | 03 | Mike Sheen |
| 2003-03-07 | 050 | 01 | Ken Stone |
| 2003-04-13 | 130 | 03 | Tom Yancy |
| 2003-08-24 | 099 | 02 | Kevin Yang |
| 2003-11-15 | 010 | 01 | Kim Nun |
| 2003-12-24 | 020 | 01 | Ted Little |
| 2004-01-27 | 080 | 02 | Lynn Felt |
| 2004-03-24 | 170 | 04 | Fawn Smith |
| 2004-07-24 | 015 | 01 | Bill Nunez |

↑ First partitioning key (RDATE)
↑ Second partitioning key (SNUM)
↑ Second partitioning key (RCODE)

- Matrix-partitioned table stored in USR01

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2002-01-31 | 001 | 01 | Ed Kimura |

- Matrix-partitioned table stored in USR02

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2002-05-25 | 110 | 03 | Sam Creek |

- Matrix-partitioned table stored in USR03

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2002-06-30 | 005 | 01 | Tim Crane |

- Matrix-partitioned table stored in USR04

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2003-01-24 | 120 | 03 | Mike Sheen |
| 2003-08-24 | 099 | 02 | Kevin Yang |

- Matrix-partitioned table stored in USR05

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2003-03-07 | 050 | 01 | Ken Stone |
| 2003-11-15 | 010 | 01 | Kim Nun |

- Matrix-partitioned table stored in USR06

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2003-04-13 | 130 | 03 | Tom Yancy |
| 2003-12-24 | 020 | 01 | Ted Little |

- Matrix-partitioned table stored in USR07

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2004-01-27 | 080 | 02 | Lynn Felt |

- Matrix-partitioned table stored in USR08

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2004-03-24 | 170 | 04 | Fawn Smith |

- Matrix-partitioned table stored in USR09

| RDATE | SNUM | RCODE | CNAME |
|---|---|---|---|
| 2004-07-24 | 015 | 01 | Bill Nunez |

## 3.3.10 Changing the partitioning storage conditions of a table

You can use `ALTER TABLE` to change the partitioning storage conditions of tables that have been row-partitioned by key range partitioning.# Changing the partitioning storage conditions of a table can reduce your workload by enabling you to reuse RDAREAs in which outdated data is stored. `ALTER TABLE` deletes a table whose partitioning storage conditions are to be changed, eliminating the need to re-create the table.

#

`ALTER TABLE` can be used to change a table's partitioning storage condition when any of the following partitioning methods was used:

- Boundary value specification
- Storage condition specification (and the equal sign (=) was used as the comparison operator for the storage condition)
- Matrix partitioning (only for key range partitioning dimensions that use boundary value specification)

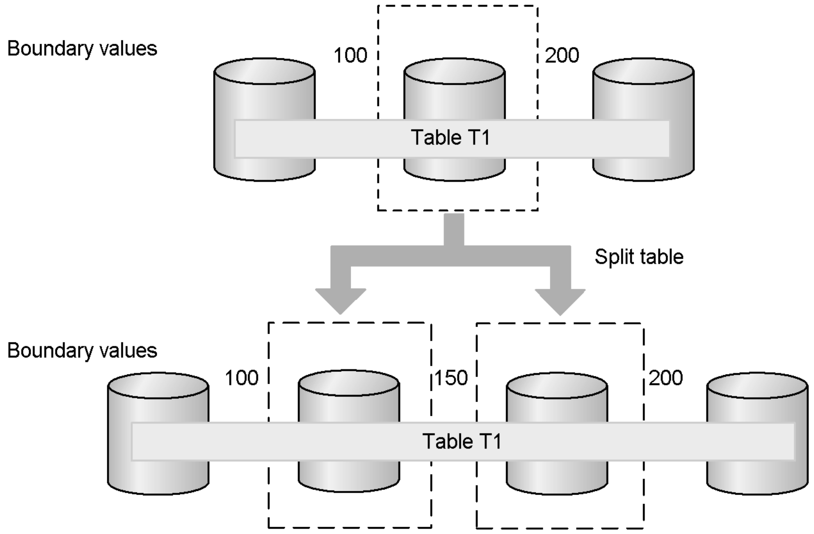
To change the partitioning storage conditions of a table, you must have HiRDB Advanced High Availability.

The following two facilities are provided for changing the partitioning storage conditions of tables.

### (1) Split facility

The split facility changes the table partitioning storage conditions, splitting data stored in a single RDAREA into multiple RDAREAs. The following figure provides an example of the split facility (using boundary value specification).

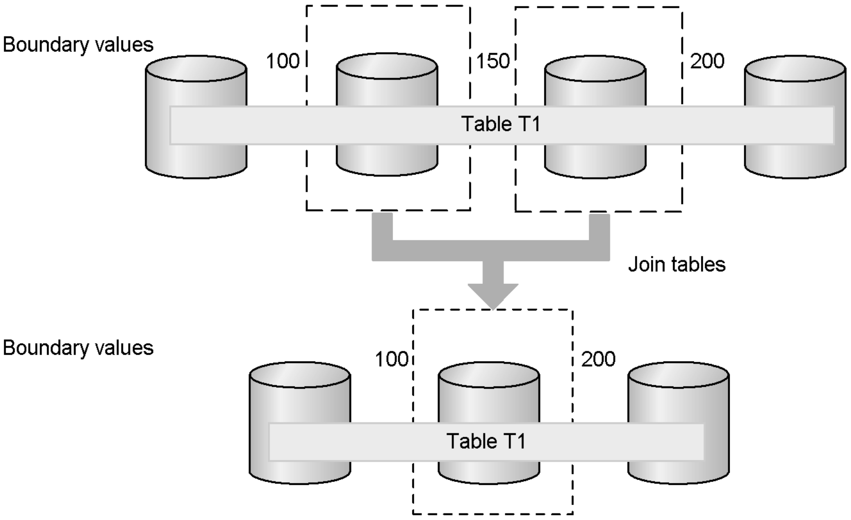
Figure 3–17: Example of the split facility (boundary value specification)



### (2) Combine facility

The combine facility changes the table partitioning storage conditions, combining data stored in multiple RDAREAs into a single RDAREA. The following figure provides an example of the combine facility (using boundary value specification).

Figure 3–18: Example of the combine facility (boundary value specification)



## 3.3.11 Falsification-prevention table

To prevent falsification of critical data due to human error or fraudulence, you can now define a falsification-prevented table, which prevents all users, including the table owner, from updating data in the table. The following table indicates the executability of operations on falsification-prevented tables.

Table 3–5: Executability of operations on falsification-prevented tables

| Operation | Falsification-prevention table | |
|---|---|---|
| | With deletion-prevention duration specification | Without deletion-prevention duration specification |
| INSERT | Y | Y |
| SELECT | Y | Y |
| Update by column (UPDATE) | Y#1 | Y#1 |
| Update by row (UPDATE) | N | N |
| DELETE | Y#2 | N |
| PURGE TABLE | N | N |
| Other data manipulation SQLs | Y | Y |

Legend:

Y: Can be executed.

N: Cannot be executed.

#1: Only updatable columns can be updated.

#2: Only data for which the deletion-prevention duration has elapsed can be deleted. If a deletion-prevention duration is not specified, the data in the table cannot be deleted.

### (1) Specification method

To prevent falsification, you specify the INSERT ONLY option (falsification prevention option) in the CREATE TABLE definition SQL. You can also use the INSERT ONLY option of ALTER TABLE to change the definition of an existing table so that it becomes a falsification-prevented table.

You can define the following types of columns when you create or change a table definition:

- Updatable column

  If you define an updatable column, you can use either of the following methods to update the data by column:

  - Always update (specify `UPDATE`)
  - Update only once from the null value to a non-null value (specify `UPDATE ONLY FROM NULL`)

  You can define updatable columns at the following times:

  - When you execute `CREATE TABLE`
  - Before you execute `ALTER TABLE CHANGE INSERT ONLY`
  - When you execute `ALTER TABLE ADD` *column-name* or `ALTER TABLE CHANGE` *column-name*#

  #: You cannot execute `ALTER TABLE CHANGE` *column-name* on a falsification-prevented table.

- Insert history maintenance column

  By defining an insert history maintenance column, you can specify a *deletion-prevented duration*. If you do not specify the deletion-prevented duration, you will not be able to delete any of the data in the table. In addition, because you cannot execute `DROP TABLE` on such a table if it contains any data, you will not be able to delete either the table itself or the data in the table. To avoid this situation, specify the deletion-prevention duration if the data hold time has been determined, or if the data hold time can be determined.

### (2) Limitations

The limitations of falsification-prevented tables and the RDAREAs in which falsification-prevented tables are stored are listed as follows. For details, see the *HiRDB Version 9 Installation and Design Guide*.

- You cannot specify the updatable column attribute for all columns and then specify the falsification prevention option.
- You cannot change existing columns to become updatable columns or change updatable columns to become regular columns.
- You must define updatable columns before you apply the falsification prevention facility.
- You cannot apply the falsification prevention facility to an existing table that already contains data. Instead, you must first unload the data from the existing table, change the table to a falsification-prevented table, and then load the data back into the table.
- You cannot delete a falsification-prevented table that contains data. The fact that you cannot delete data in a falsification-prevented table for which no deletion-prevention duration has been specified means that you cannot delete the table itself.
- You cannot use the database structure reconfiguration utility (`pdmod`) to initialize an RDAREA (`initialize rdarea`) in which a falsification-prevented table is stored.
- You cannot use the inner replica facility (UNIX edition only) on RDAREAs in which a falsification-prevented table is stored. In addition, you cannot apply the falsification prevention facility to a table that is stored in an RDAREA that uses the inner replica facility.
- Do not use a replication facility (HiRDB Dataextractor or HiRDB Datareplicator) to duplicate data in or to reflect update results to a falsification-prevented table. If you do so, the data in the copy source and copy target might become inconsistent, resulting in errors.

## 3.3.12 Tables used in numbering

Numbering means the capability to perform other tasks using report numbers and document numbers that have been created primarily for one task. The table used for numbering is dedicated to number management only. Because multiple users need to access to this table simultaneously, it is necessary to minimize the lock processing to conduct speedy updating.

In this table, a function is provided that releases row locking as soon as the table update is complete without waiting for transaction confirmation, and that prevents subsequent rollbacks.

To define a table used for numbering, the `WITHOUT ROLLBACK` option must be specified in the `CREATE TABLE` definition SQL.

For details about numbering, see the *HiRDB Version 9 UAP Development Guide*.

## 3.3.13 Repetition columns

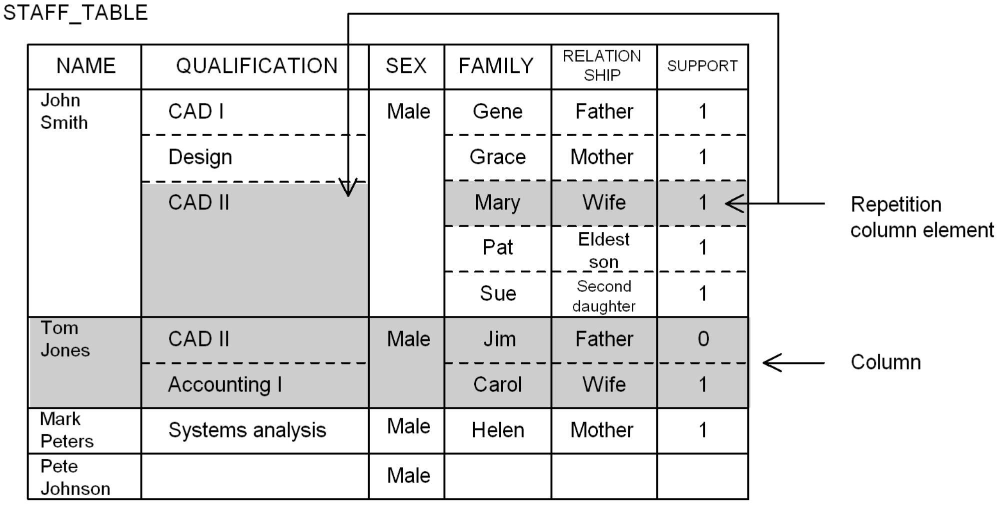
A repetition column is a column whose data values might consist of multiple elements. An *element* means each item that is stored in the same row of the repetition column. A column is defined as a repetition column in `CREATE TABLE`; the number of elements must also be defined (however, the number of elements can be increased later with the `ALTER TABLE`).

Defining a table to contain repetition columns offers the following benefits:

- If multiple tables contain identical information, the overlapping information can be eliminated, thus reducing the amount of disk space that is required.
- The need for joining multiple tables is eliminated.
- Because related data (repetition data) is stored nearby, access performance is better than when separate tables are accessed.

The following figure shows an example of a table containing repetition columns. For details about repetition columns, see the *HiRDB Version 9 Installation and Design Guide*.

Figure 3–19: Example of a table containing repetition columns



STAFF_TABLE

| NAME | QUALIFICATION | SEX | FAMILY | RELATION SHIP | SUPPORT |
|---|---|---|---|---|---|
| John Smith | CAD I<br>Design<br>CAD II | Male | Gene<br>Grace<br>Mary<br>Pat<br>Sue | Father<br>Mother<br>Wife<br>Eldest son<br>Second daughter | 1<br>1<br>1<br>1<br>1 |
| Tom Jones | CAD II<br>Accounting I | Male | Jim<br>Carol | Father<br>Wife | 0<br>1 |
| Mark Peters | Systems analysis | Male | Helen | Mother | 1 |
| Pete Johnson |  | Male |  |  |  |

Note: A blank cell means the null value.

### (1) Example of defining a repetition column

The following is the `CREATE TABLE` SQL statement that defines the table containing the repetition column in *Figure 3-19*:

```
CREATE TABLE employee list
   NAME NVARCHAR(10),
   QUALIFICATION NVARCHAR(20) ARRAY[10],
   SEX NCHAR(1),
   FAMILY NVARCHAR(5) ARRAY[10],
   RELATIONSHIP NVARCHAR(5) ARRAY[10],
   SUPPORT SMALLINT ARRAY[10]);
```

### (2) Operations for repetition columns

The operations listed below can be performed on a table containing repetition columns.

**Retrieval using a structured repetition predicate**

A structured repetition predicate is used to perform a search on multiple repetition columns in a table, where the subscripts correspond directly with the elements.

**Example 1**

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

**Updating a repetition column**

The following three repetition column updating methods are provided:

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

To update a table containing a repetition column, *repetition-column-name* `[{`*subscript* | `*}]` is used to specify the element of the repetition column that is to be updated.

**Example 2: Updating an existing element**

```
UPDATE STAFF_TABLE SET QUALIFICATION[2]=N'Accounting-II'
       WHERE NAME=N'Tom Jones'
```

**Example 3: Adding a new element**

```
UPDATE STAFF_TABLE ADD QUALIFICATION[*]=ARRAY{N'Systems analysis'}
       WHERE NAME=N'Tom Jones'
```

**Example 4: Deleting an existing element**

```
UPDATE STAFF_TABLE DELETE QUALIFICATION[2]
       WHERE NAME=N'Tom Jones'
```

**Other operations for repetition columns**

To specify a repetition column in an SQL statement, *repetition-column-name* `[`*subscript*`]` must be used.

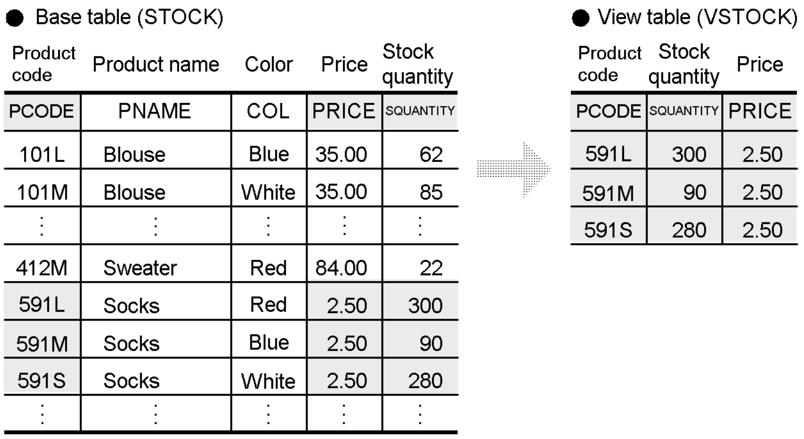
For details about the operations for tables containing repetition columns, see the *HiRDB Version 9 UAP Development Guide*.

## 3.3.14 View table

You can create a virtual table by selecting specific rows and columns from a table that is actually stored in a database (called a *base table*). Such a virtual table is called a *view table*. By creating a view table that makes only specified columns available to the public, you can effectively protect the remainder of the data. In addition, the resulting reduction in the number of columns to be manipulated improves the operability of the table.

To create a view table, you must execute the `CREATE VIEW` of the definition SQL. The following figure shows an example of a view table.

Figure 3–20: Example of a view table



● Base table (STOCK)

| Product code | Product name | Color | Price | Stock quantity |
|---|---|---|---|---|
| PCODE | PNAME | COL | PRICE | SQUANTITY |
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

● View table (VSTOCK)

| Product code | Stock quantity | Price |
|---|---|---|
| PCODE | SQUANTITY | PRICE |
| 591L | 300 | 2.50 |
| 591M | 90 | 2.50 |
| 591S | 280 | 2.50 |

**Explanation**

The VSTOCK view table is created from the STOCK base table, and is composed of the product code (PCODE), stock quantity (SQUANTITY), and unit price (PRICE) columns and only those rows with Socks in the product name (PNAME) column. The columns are arranged in the order of product code, stock quantity, and unit price.

```
CREATE VIEW VSTOCK
   AS SELECT PCODE,SQUANTITY,PRICE
   FROM STOCK
   WHERE PNAME = N'Socks'
```

## 3.3.15 Shared tables

In the case of a HiRDB parallel server configuration, when multiple tables are joined, table data is read from the back-end servers where the individual tables are located and then matching is performed at a separate back-end server. This means that multiple servers are connected to transfer data. If the range of data to be searched for matches is located on a single back-end server, matching can be completed at the single back-end server by creating that data as a shared table. A *shared table* is a table stored in a shared RDAREA that can be referenced by all back-end servers. An index defined for a shared table is called a *shared index*. For details about shared tables, see the *HiRDB Version 9 Installation and Design Guide*.

Shared tables and shared indexes can also be defined for a HiRDB single server configuration. This provides SQL and UAP compatibility with a HiRDB parallel server configuration. Shared tables and shared indexes are usually used with a HiRDB parallel server configuration because they are especially effective in HiRDB parallel server configurations. This section describes the use of shared tables with a HiRDB parallel server configuration. For details about using shared tables with a HiRDB single server configuration, see *(6) Using shared tables with a HiRDB single server configuration* below.

### (1) Advantages of shared tables

Because join processing can be completed by a single back-end server, the overhead associated with connecting between back-end servers and transferring data is reduced. Additionally, the number of back-end servers required for each transaction can be reduced, thereby improving the efficiency of parallel processing, particularly in the event of multiple executions.

### (2) Application criteria

We recommend that you create as a shared table a table that typically involves minor update processing, but which is referenced by multiple transactions, such as for join processing.

### (3) Notes on updating shared tables

When you update a shared table, all back-end servers in which RDAREAs are allocated are locked. If you execute any statement other than the UPDATE statement or PURGE TABLE statement, which do not update index key values, you must specify IN EXCLUSIVE MODE with the LOCK TABLE statement and lock all back-end servers to update the shared table. This means that any jobs that access a locked RDAREA might result in deadlock, or that global deadlock might occur among the servers.

When you use a local buffer to update a shared table, issue the LOCK statement before you update. If you update without issuing the LOCK statement and the server process terminates abnormally, the abort code Phb3008 is output (which indicates that the unit might have terminated abnormally).

### (4) Definition method

To define a shared table, specify SHARE with CREATE TABLE in the definition SQL (specify CREATE SHARE FIX TABLE). Note that shared tables must satisfy the following conditions:

- The shared table is a non-partitioned FIX table.
- The RDAREA storing the shared table or shared index is a shared RDAREA (specify SDB for the -k option of the pdfmkfs command).
- The WITHOUT ROLLBACK option is not specified.

- The table is not a referencing table defined with a referential constraint.

### (5) Limitations

The following limitations apply to using a shared table:

- You cannot search a shared table while the `LOCK TABLE` statement is being executed with `IN EXCLUSIVE MODE` specified.
- You cannot use the `ASSIGN LIST` statement to create a list of shared tables.
- You cannot specify a shared table as a replication copy destination

### (6) Using shared tables with a HiRDB single server configuration

- You cannot define shared RDAREAs in a HiRDB single server configuration; therefore, you must store shared tables and shared indexes in regular user RDAREAs. When you do this, you must provide separate user RDAREAs for shared tables and shared indexes; you must not store shared tables in user RDAREAs that store regular tables and indexes that are not shared.
- When you migrate a HiRDB single server configuration to a HiRDB parallel server configuration, do not use the database structure modification utility (`pdmod`) while shared tables or shared indexes are defined. For details about the migration procedure, see the *HiRDB Version 9 Installation and Design Guide*.

For details about the differences with a HiRDB parallel server configuration, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.3.16 Compressed tables

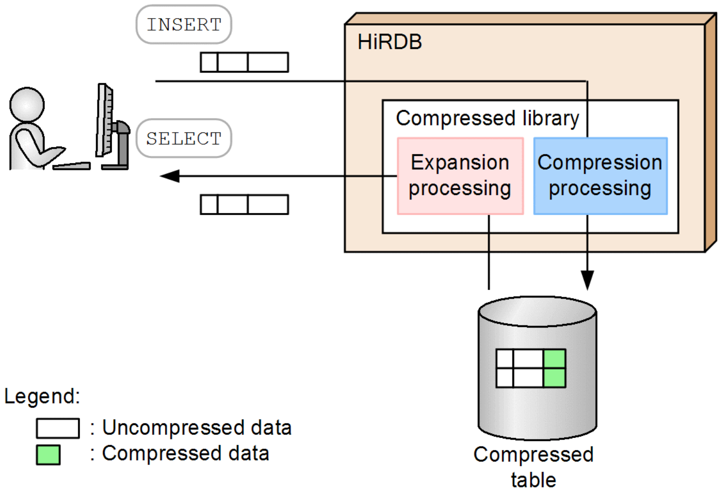
You can compress data that HiRDB stores in a table. This is called the *data compression facility*. You can specify data compression for each column individually. A column for which compression is specified is called a *compressed column*, and a table containing a compressed column is specified is called a *compressed table*. For details about compressed tables, see the *HiRDB Version 9 Installation and Design Guide*.

Data compression by HiRDB provides the following benefits:

- The size of the database can be reduced.
- There is no need for UAPs to have a data compression facility.

The following figure provides an overview of data compression.

Figure 3–21: Overview of data compression



**Explanation**

Because HiRDB executes data compression and expansion, the user need not specify these operations.

## (1) Application criteria

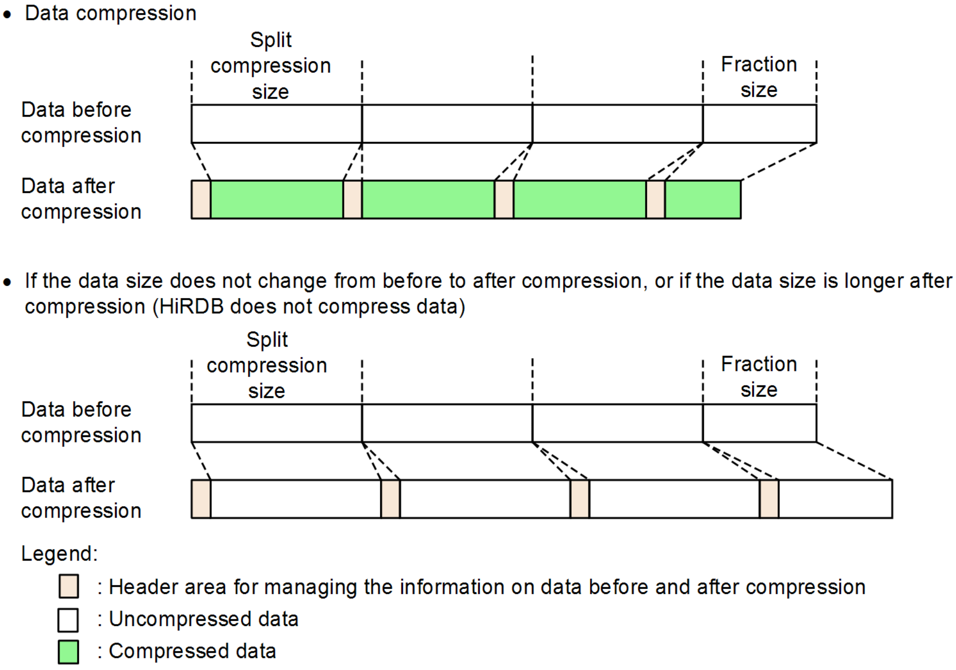
We recommend specifying tables that contain large-size, variable-length binary data, such as image and audio data, as compressed tables. Note, however, that compressing tables results in compression and expansion processing overhead. Therefore, use data compression in systems in which storage efficiency is more important than performance.

## (2) Data compression mechanism

The compression library you use for data compression is `zlib`. Using `zlib`, HiRDB compresses data into the split compression size (default: MIN (32,000 bytes, compressed column definition length)) that was specified during table definition. During this process, HiRDB adds a header area (8 bytes) to manage the data pre- and post-compression, when it is compressed to each split compression size (this is separate from the header area added by `zlib` to the compressed data).

If the data size does not change from before compression to after, or if the data size is longer after compression, HiRDB stores the data without compressing it. Consequently, the addition of a header area makes the post-compression data larger than the pre-compression data in some cases. The following figure shows an example of pre- and post-compression data.

Figure 3–22: Pre- and post-compression data


• Data compression
Split compression size
Fraction size
Data before compression
Data after compression
• If the data size does not change from before to after compression, or if the data size is longer after compression (HiRDB does not compress data)
Split compression size
Fraction size
Data before compression
Data after compression
Legend:
: Header area for managing the information on data before and after compression
: Uncompressed data
: Compressed data

## (3) Defining a compressed table

Specify compression (`COMPRESSED`) in the column definition of the `CREATE TABLE` definition SQL. As needed, also specify the split compression size. Note that the following restrictions apply to compression specification:

- You must specify compression by column (you cannot specify compression by table).
- You can specify compression only for columns with the following data types:
  - `BINARY` type with a defined size of 256 bytes or larger
  - Abstract data type (XML type)#

#

To compress data of a column with the abstract data type (XML type), you must have HiRDB XML Extension, Version 09-03 or newer.

### (4) Notes on using compressed tables

- When the data of a compressed column is processed by an SQL or utility, compression and expansion processing overhead will result.
- Although the results will vary depending on the actual data that is compressed, data compression efficiency improves as the split compression size increases. However, when the split compression size increases, executing an SQL that stores data in or extracts data from compressed columns (that is, an SQL that uses the `SUBSTR` or `POSITION` function, or that performs backend deletion updates) increases the size of the process private area that must be allocated during SQL execution.
- When a compressed table is rebalanced in shared mode, data expansion and compression are performed. Consequently, rebalancing might take longer than when there are no compressed columns. To shorten the execution time, execute rebalancing in exclusive mode.

## 3.3.17 Temporary tables

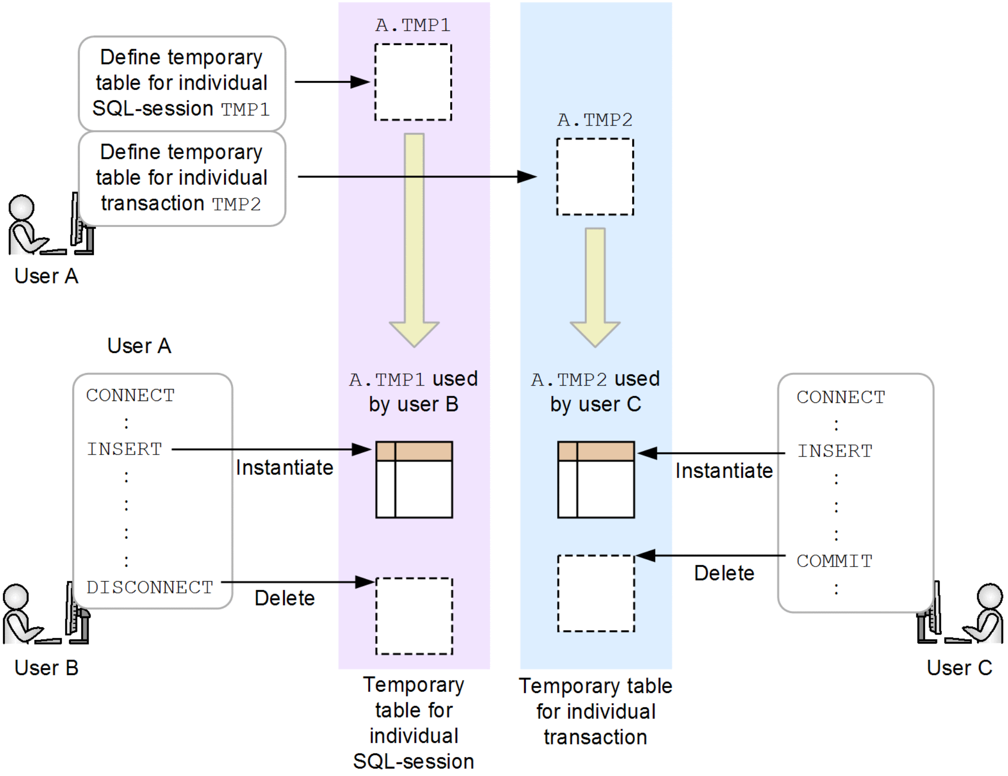
A temporary table is a base table that exists only during a transaction or SQL session. A temporary table that exists only during a transaction is called a transaction-specific temporary table, and a temporary table that exists only during an SQL session is called an SQL session-specific temporary table.

No temporary table is created when a table is defined. A table is created when the `INSERT` statement is executed on a temporary table for the first time. This process is called the instantiation of a temporary table.

Furthermore, an exclusive temporary table is created each time HiRDB connects to a table (by executing the `CONNECT` statement). Therefore, even if multiple users use temporary tables concurrently, the tables are not affected by the data operations performed by users (referencing, inserting, updating, or deleting). Temporary tables and the indexes defined in them (temporary table indexes) are stored in the temporary table RDAREA, and are automatically deleted when a transaction is completed or when an SQL session ends.

The following figure provides an overview of temporary tables.

Figure 3–23: Overview of temporary tables


A.TMP1
Define temporary table for individual SQL-session TMP1
Define temporary table for individual transaction TMP2
A.TMP2
User A
User A
CONNECT
:
INSERT
:
:
:
:
DISCONNECT
Instantiate
Delete
A.TMP1 used by user B
A.TMP2 used by user C
Instantiate
Delete
CONNECT
:
INSERT
:
:
:
COMMIT
:
User B
User C
Temporary table for individual SQL-session
Temporary table for individual transaction

**Effects of temporary tables**

- When a transaction or SQL session involves complicated processing, temporary tables can hold the intermediate processing results temporarily, and they can also be used as work tables for processing data to obtain final results.
- When certain data in a table containing many data items must be accessed frequently during a transaction or SQL session, storing this data in a temporary table can reduce the number of input/output operations, thereby improving performance.
- Because temporary tables are automatically deleted when a transaction is completed or an SQL session ends, no post-processing by a UAP is required. This reduces the UAP creation workload.

**Temporary table application criteria**

Use of temporary tables is advisable for transactions that frequently access only certain data in a table containing many data items, and for batch jobs that perform complex processing, such as temporarily storing intermediate processing results.

For details about temporary tables, see the *HiRDB Version 9 Installation and Design Guide*.

## (1) Effective duration of data in temporary tables

The effective data duration of an instantiated temporary table (the duration during which the actual body exists) differs depending on whether the temporary table is a transaction-specific temporary table, or an SQL session-specific temporary table. The following table shows start and end times for the effective data duration of temporary tables.

Table 3–6: Start and end timing for the effective data duration of temporary tables

| Temporary table type | Start timing | End timing |
|---|---|---|
| Transaction-specific temporary table | When the INSERT statement is executed on a temporary table during the transaction for the first time | When the transaction is completed |

| Temporary table type | Start timing | End timing |
|---|---|---|
| SQL session-specific temporary table | When the INSERT statement is executed on a temporary table during the SQL session for the first time | • When the SQL session ends<br>• When the back-end server that instantiated the temporary table is finished<br>• When the unit is finished that has the back-end server that instantiated the temporary table<br>• When system switching occurs on the back-end server that instantiated the temporary table, or on the unit where the back-end server is located |

! Important note

Executing a retrieval, update, or deletion operation on a temporary table outside its effective data duration produces the same result as executing an SQL on a table containing no data.

## (2) Defining temporary tables and temporary table indexes

### (a) Defining temporary tables

In the `CREATE TABLE` definition SQL, specify `GLOBAL TEMPORARY`. To define a transaction-specific temporary table, specify `ON COMMIT DELETE ROWS`; to define an SQL session-specific temporary table, specify `ON COMMIT PRESERVE ROWS`. Note that some operands cannot be specified or are ignored even if specified for temporary tables. For details, see *CREATE TABLE* in the manual *HiRDB Version 9 SQL Reference*.

### (b) Defining temporary table indexes

Temporary table indexes are defined in essentially the same way as ordinary indexes. As is the case with temporary tables, some operands cannot be specified or are ignored even if specified for temporary table indexes. For details, see the *CREATE INDEX* sections in the manual *HiRDB Version 9 SQL Reference*.

### (c) Specifying a temporary table RDAREA for storing data

Specify the name of the temporary table RDAREA to be used in the `PDTMPTBLRDAREA` client environment definition. If multiple RDAREAs are specified or if this environment definition is omitted, HiRDB determines the temporary table RDAREA for storing data according to the following rules:

- When multiple RDAREAs are specified in `PDTMPTBLRDAREA`

  When the specified RDAREAs include temporary table RDAREAs with an attribute that is exclusive for a specific SQL session and with an attribute shared among SQL sessions, the temporary table RDAREA with an attribute that is exclusive for the specific SQL session is used at higher priority.

- When specification of `PDTMPTBLRDAREA` is omitted

  The temporary table RDAREA with an attribute shared among SQL sessions is used.

Note that specification of `PDTMPTBLRDAREA` is ignored on an XDS client, and therefore specification is considered omitted.

For details about the rules for determining the storage destination RDAREA, see the *HiRDB Version 9 Installation and Design Guide*.

## (3) Restrictions on the use of temporary tables

### (a) Operation commands and utilities

The operation commands and utilities listed below cannot be executed on temporary tables. For details, see the manual *HiRDB Version 9 Command Reference*.

- `pdorbegin` command (UNIX edition only)
- Optimizing-information collection utility (`pdgetcst`)

- Database load utility (`pdload`)
- Global buffer residence utility (`pdpgbfon`)
- Free page release utility (`pdreclaim`)
- Database reorganization utility (`pdrorg`)

### (b) SQL

The SQL statements listed below cannot specify temporary tables or temporary table indexes. For details, see the manual *HiRDB Version 9 SQL Reference*.

- Definition SQLs
  `ALLOCATE MEMORY TABLE`
  `ALTER INDEX`
  `ALTER TABLE`
  `CREATE INDEX`
  `CREATE TABLE`
  `CREATE TRIGGER`
  `GRANT`
  `REVOKE`
- Data manipulation SQLs
  `ALLOCATE CURSOR` statement
  `ASSIGN LIST` statement
  `DECLARE CURSOR` statement
  Dynamic `SELECT` statement
  `SELECT` statement (table referencing, query expression format 2)
  `DELETE` statement
  `UPDATE` statement
- Control SQLs
  `COMMIT` statement
  `DISCONNECT` statement
  `ROLLBACK` statement
  `LOCK TABLE` statement
  `SET SESSION AUTHORIZATION` statement

# 3.4 Indexes

An index is a search-basis key that is assigned to one or more columns of a table for the purpose of making retrievals from that table. An index can be either a B-tree index provided by HiRDB or a plug-in index provided by a plug-in. B-tree indexes are explained here; for an explanation of plug-in indexes, see Section *10.4 Preparations for using plug-ins in HiRDB*.

When you define an index, define it on the table columns that you have determined will improve retrieval performance.

## 3.4.1 Basic structure of an index

An index consists of a key and key values. A *key* is the column name of an indexed column. The values in the column are called the *key values*. Creating an index for a column that will be used as the basis for retrievals from the table will improve the table's retrieval performance.

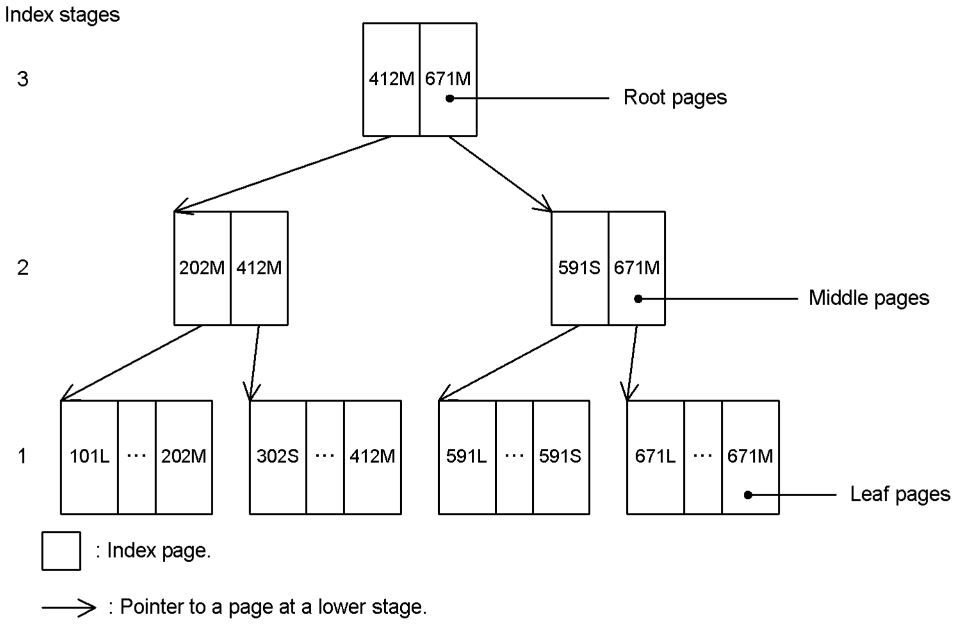
It is recommended that indexes be created for the following types of columns:

- Column used as a condition for narrowing the range of data to be searched
- Column used as a condition for joining tables
- Column used as a condition for sorting or grouping data
- Component column that defines a referential constraint (foreign key)

Indexes provided by HiRDB have a B-tree structure. The highest stage of an index in a B-tree structure is called a *root page*; the lowest stage is called a *leaf page*; a page in the middle is called a *middle page*. Root pages and middle pages point to pages at lower stages. A leaf page has a key value for each index page and its address.

The following figure shows the B-tree structure of an index.

Figure 3–24: B-tree structure of an index



**Explanation**

This is a B-tree structure index. The index is created in Step 3 on the basis of the *product code* column, `PCODE` is the key, and `101L` to `671M` are key values.

### (1) Single column index and multicolumn index

An index can be based either on a single column or on multiple columns (multicolumn index). A single column index is created by indexing the values in a single column of a table. A single column index is appropriate for retrievals that require a single column as the key. A multicolumn index is created by indexing the values in more than one column of a table. A multicolumn index is appropriate for the following purposes:

- To narrow the data that is to be retrieved to data that satisfies multiple conditions
- To group or sort the data that has been narrowed using a retrieval condition
- When some of the same columns are used in multiple multicolumn indexes created for a table

### (2) Optimizing based on cost

When multiple indexes have been created for a table, HiRDB selects and uses the index that has the least access cost and that it evaluates as being optimal for the conditions specified for the retrieval. This process in which HiRDB selects an index based on an evaluation of optimality is called *optimizing based on cost*. HiRDB evaluates the following access costs:

- Hit rate based on the specified retrieval conditions
- Number of input/output processes required for SQL processing
- CPU load required for SQL processing

HiRDB performs optimizing based on cost in order to improve table retrieval performance. Even when an SQL with retrieval conditions specified is executed, table retrieval performance does not deteriorate. This means that it is possible to create UAPs without being concerned about index access priority. However, in order for HiRDB to use the optimal index, indexes must have been defined for those columns for which retrieval conditions are specified.

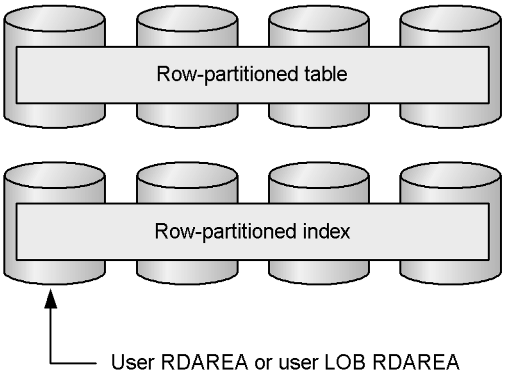
### (3) Index definition

The `CREATE INDEX` definition SQL is used to specify the columns in a table that are to be indexed. For details about design considerations related to indexes, such as index definition for retrieval performance improvement, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.4.2 Index row partitioning

The process of row-partitioning an index for a table that has been row-partitioned is called *index row partitioning*. The index must be partitioned so that it corresponds to the row-partitioned table in terms of the multiple RDAREAs in which the table is stored. Such an index is called a *row-partitioned index*. When an index is row-partitioned, each user RDAREA or user LOB RDAREA can be handled independently during index batch creation or re-creation. The following figure illustrates index row partitioning.

Figure 3–25: Index row partitioning



**Explanation**

You use the `CREATE INDEX` of the definition SQL to specify the RDAREAs in which a row-partitioned index is to be stored.

## (1) Example of index row partitioning: HiRDB single server configuration

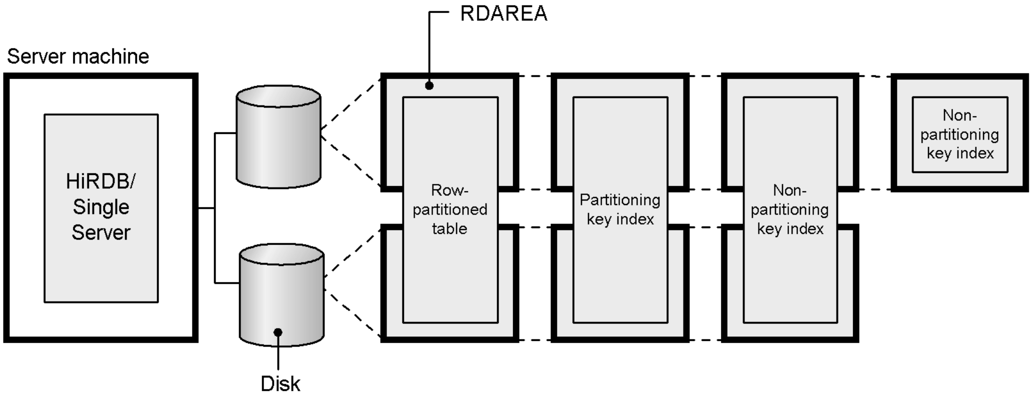
When you have a row-partitioned table, you need to know whether an index being defined is a partitioning key index or a non-partitioning key index. For details about partitioning key indexes and non-partitioning key indexes, see Section *(3) Partitioning key index and non-partitioning key index* as follows. The following table shows guidelines for row-partitioning an index by index type.

Table 3–7: Guidelines for row-partitioning an index: HiRDB single server configuration

| Index type | Partitioning guidelines |
|---|---|
| Partitioning key index | Row-partition the index so that it corresponds to the row-partitioned table. |
| Non-partitioning key index | We recommend that the index not be row-partitioned, because row-partitioning the index can reduce the performance of index-based searches.<br>However, if the table contains an extremely large amount of data, you might wish to consider row-partitioning it. Row-partitioning an index allows the system to manage the table storage RDAREAs and the index storage RDAREAs on a 1-to-1 basis, which will improve the operability of the utilities. For example, if data is loaded by RDAREA, or if RDAREAs are reorganized without the index having been row-partitioned, you will have to batch-create an index after you have loaded the data or reorganized the RDAREAs. If the index is row-partitioned, you will not have to batch-create an index after data loading or RDAREA reorganization. |

The following figure shows an example of row-partitioning an index for a HiRDB single server configuration.

Figure 3–26: Example of row-partitioning an index: HiRDB single server configuration



**Explanation**

- To prevent disk access contention, the RDAREAs storing the partitioned table and the index are allocated to different disks.
- The partitioning key index is row-partitioned.
- If better performance is important, you would not row-partition non-partitioning key indexes.
- If better operability is important, you would row-partition non-partitioning key indexes.

## (2) Example of index row partitioning: HiRDB parallel server configuration

In the case of a HiRDB parallel server configuration, the index row partitioning guidelines depend on whether the table is partitioned within a server or among servers.

#### (a) Table is row-partitioned within a server

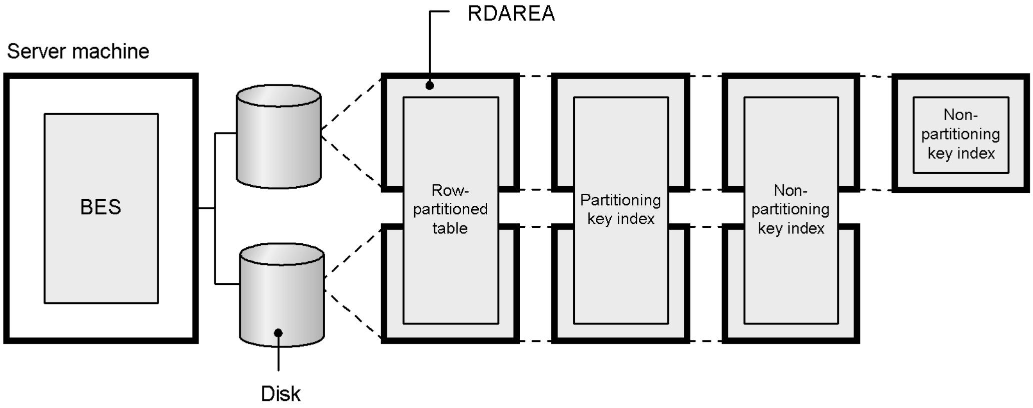
You need to know whether the index being defined is a partitioning key index or a non-partitioning key index. For details about partitioning key indexes and non-partitioning key indexes, see Section *(3) Partitioning key index and non-partitioning key index* below. The following table shows guidelines for row-partitioning an index by index type.

Table 3–8: Guidelines for row-partitioning an index: HiRDB parallel server configuration

| Index type | Partitioning guidelines |
|---|---|
| Partitioning key index | Row-partition the index so that it corresponds to the row-partitioned table. |
| Non-partitioning key index | We recommend that the index not be row-partitioned, because row-partitioning the index can reduce the performance of index-based searches.<br>However, if the table contains an extremely large amount of data, you might wish to consider row-partitioning it. Row-partitioning an index allows the system to manage the table storage RDAREAs and the index storage RDAREAs on a 1-to-1 basis, which will improve the operability of the utilities. For example, if data is loaded by RDAREA, or if RDAREAs are reorganized without the index having been row-partitioned, you will have to batch-create an index after you have loaded the data or reorganized the RDAREAs. If the index is row-partitioned, you will not have to batch-create an index after data loading or RDAREA reorganization. |

The following figure shows an example of row-partitioning an index within a server.

Figure 3–27: Example of row-partitioning an index within the server



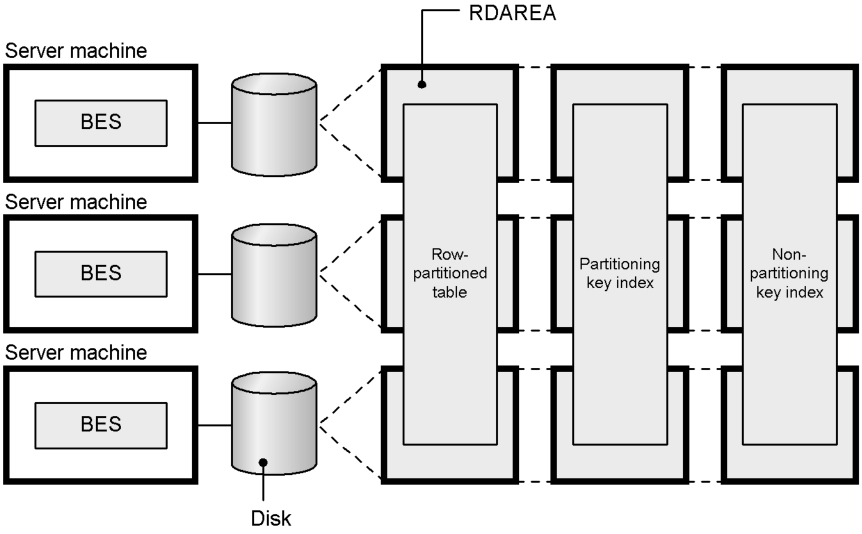
**Explanation**

- To prevent disk access contention, allocate the RDAREAs storing the partitioned table and the index to different disks.
- Row-partition a partitioning key index.
- If better performance is important, do not row-partition non-partitioning key indexes.
- If better operability is important, row-partition non-partitioning key indexes.

#### (b) Table is row-partitioned among servers

You must row-partition the index so that it corresponds to the row-partitioned table. You need not know whether the index being defined is a partitioning key index or a non-partitioning key index. The following figure shows an example of row-partitioning an index among servers.

Figure 3–28: Example of row-partitioning an index among servers



**Explanation**

- To prevent disk access contention, the RDAREAs storing the partitioned table and the index are allocated on different disks.
- Both partitioning key indexes and non-partitioning key indexes are row-partitioned.

## (3) Partitioning key index and non-partitioning key index

An index that satisfies a particular condition becomes a partitioning key index. An index that does not satisfy the condition is called a non-partitioning key index. The condition is explained below. The condition is whether the table involved is partitioned on the basis of single-column partitioning or multicolumn partitioning. When only one column is used in the table partitioning condition, the partitioning is said to be *single-column partitioning*; when multiple columns are used in the table partitioning condition, the partitioning is said to be *multicolumn partitioning*.

### (a) Single-column partitioning

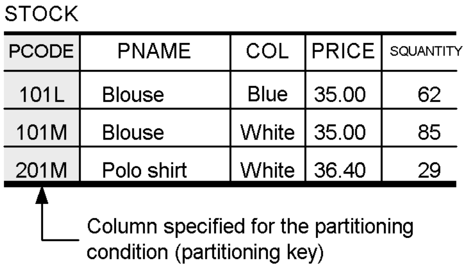
An index satisfying one of the following conditions is a partitioning key index:

**Conditions**

- Single-column index defined in the column (partitioning key) specifying the storage conditions when the table is row-partitioned.
- Multicolumn index with the column (partitioning key), defined as the first member column, specifying the storage conditions when the table is row-partitioned.

The following figure shows an example (based on the STOCK table) of an index that becomes a partitioning key index.

Figure 3–29: Partitioning key index: Single-column partitioning



STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |

Column specified for the partitioning condition (partitioning key)

**Explanation**

```
CREATE INDEX A12 ON STOCK (PCODE ASC)               1
CREATE INDEX A12 ON STOCK (PCODE ASC,PRICE DESC)    2
CREATE INDEX A12 ON STOCK (PRICE DESC,PCODE ASC)    3
```

1. If the PCODE column, which is the partitioning key, is used as an index, the index becomes a *partitioning key index*. If any other column is used as an index, the resulting index becomes a non-partitioning key index.
2. Specifying the PCODE column, which is the partitioning key, as the first constituent column of the index makes the resulting multicolumn index a *partitioning key index*.
3. Specifying the PCODE column, which is the partitioning key, as a column other than the first constituent column of the index makes the resulting multicolumn index a *non-partitioning key index*.

(b) Multicolumn partitioning

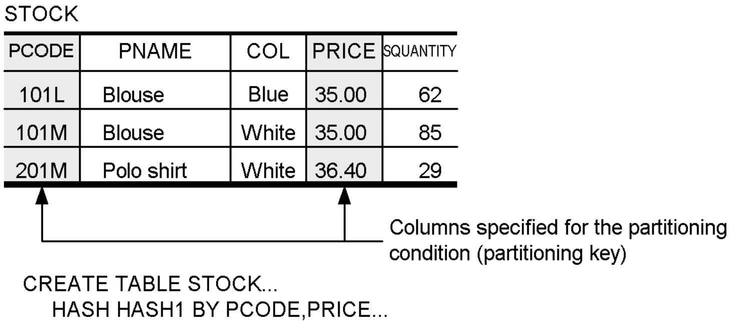
An index satisfying the following condition is a partitioning key index:

**Condition**

- The index is created on the basis of multiple columns, beginning with the partitioning key and containing all the columns specified for partitioning from the beginning and without any change in their order.

The following figure shows an example (based on the STOCK table) of an index that becomes a partitioning key index.

Figure 3–30: Partitioning key index: Multicolumn partitioning



STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |

Columns specified for the partitioning condition (partitioning key)
CREATE TABLE STOCK...
HASH HASH1 BY PCODE,PRICE...

**Explanation**

```
CREATE INDEX A12 ON STOCK (SCODE ASC,PRICE DESC)   1
CREATE INDEX A12 ON STOCK
             (SCODE ASC,PRICE DESC,SQUANTITY ASC)  2
CREATE INDEX A12 ON STOCK (PRICE DESC,PCODE ASC)   3
CREATE INDEX A12 ON STOCK
             (SCODE ASC,SQUANTITY DESC,PRICE ASC)  4
```

1. All partitioning keys (PCODE and PRICE columns) are specified, and these keys are specified in the same order as in the table definition. Therefore, this multicolumn index is a *partitioning key index*.
2. All partitioning keys (PCODE and PRICE columns) are specified, and these keys are specified in the same order as in the table definition. Therefore, this multicolumn index is a *partitioning key index*.
3. All partitioning keys (PCODE and PRICE columns) are specified, but these keys are specified in an order that differs from the table definition order. Therefore, this multicolumn index is a *non-partitioning key index*.
4. All partitioning keys (PCODE and PRICE columns) are specified, but these keys are specified in an order that differs from the table definition order. Therefore, this multicolumn index is a *non-partitioning key index*.

## 3.4.3 Index page splitting

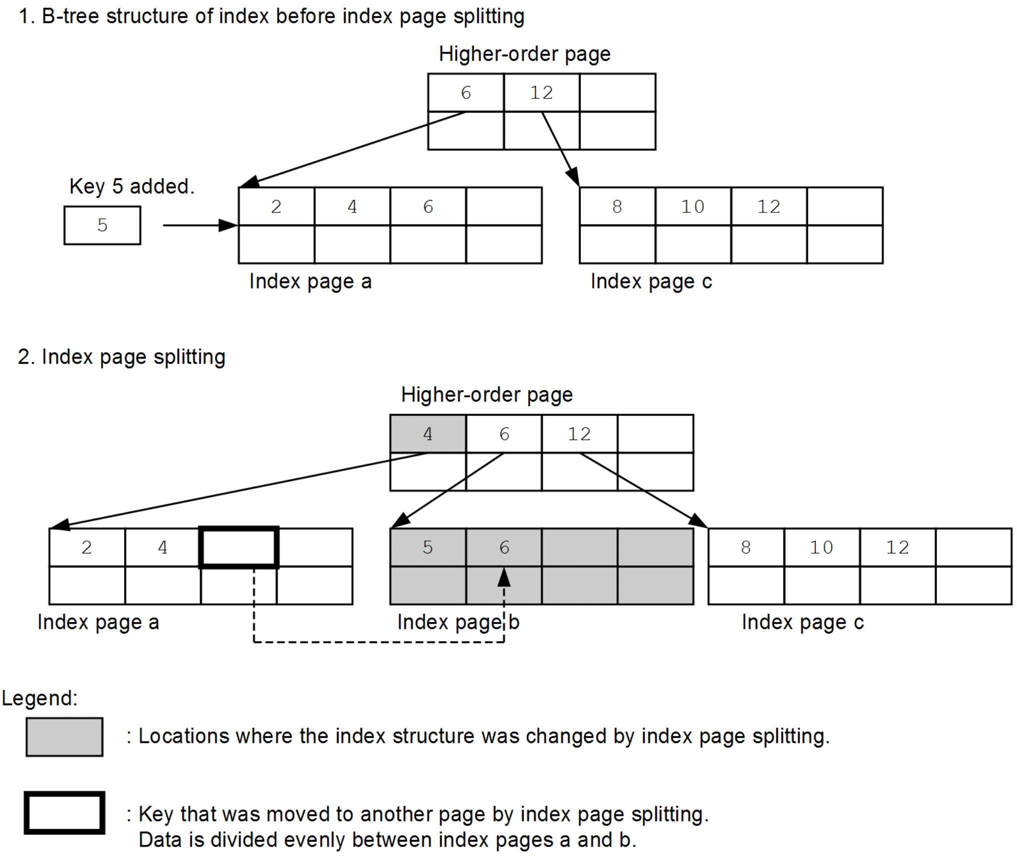
When an attempt is made to add a key to an index page that does not contain any more free space, HiRDB allocates space for a new index page and splits the index information on the existing page. It then transfers the second portion of the index information to the new page. This is called *index page splitting*.

An index page can be split so that the existing index information is divided evenly in half, or it can be split unevenly (unbalanced index splitting).

## (1) Even-split index page splitting

The following figure shows an example of even-split index page splitting.

Figure 3–31: Example of even-split index page splitting


1. B-tree structure of index before index page splitting
Higher-order page
6
12
Key 5 added.
5
2
4
6
8
10
12
Index page a
Index page c
2. Index page splitting
Higher-order page
4
6
12
2
4
5
6
8
10
12
Index page a
Index page b
Index page c
Legend:
: Locations where the index structure was changed by index page splitting.
: Key that was moved to another page by index page splitting.
Data is divided evenly between index pages a and b.

**Explanation**

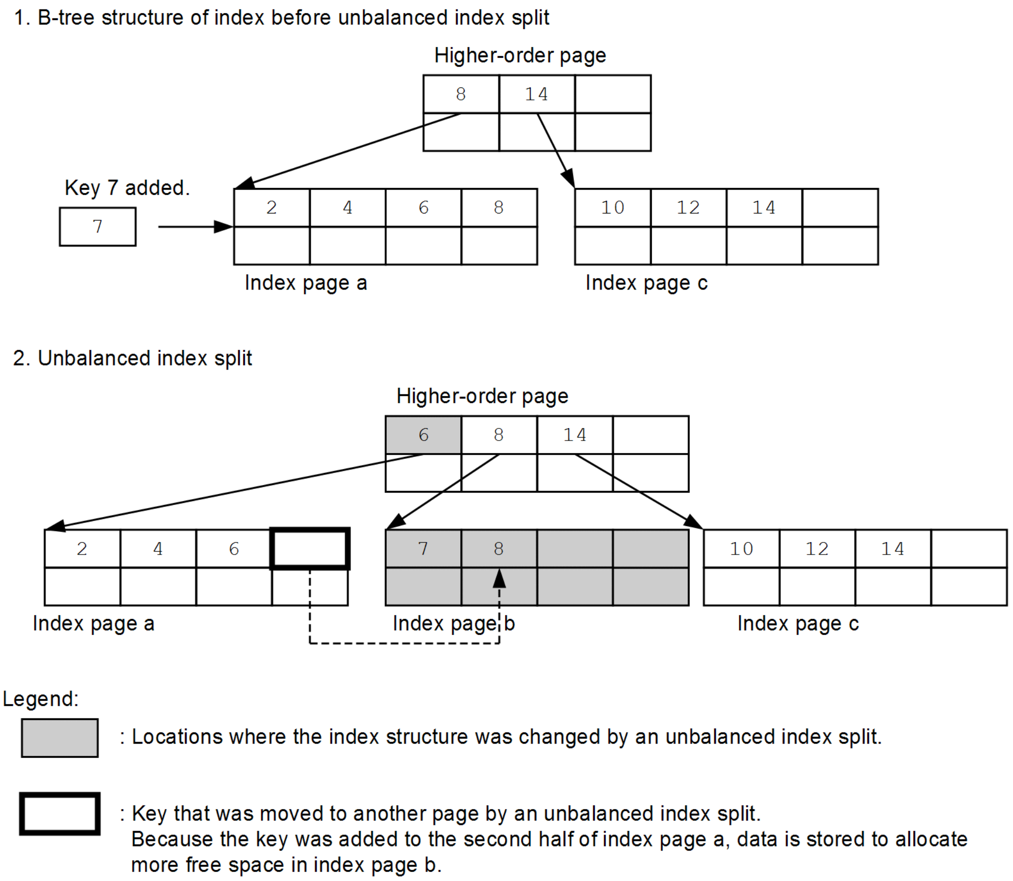
The addition of key 5 to index page a causes index page splitting. Key 5, as well as key 6 that follows, are moved to a new index page b so that the keys are distributed evenly.

## (2) Unbalanced index split

Adding many contiguous key values can degrade an index page's data storage efficiency. In such a case, it is advisable to split the index information on the index page unevenly; this is called *unbalanced index splitting*.

When unbalanced index splitting is used, the place in the index information at which the index page is split is determined by the location of the key value that is being added. If the location of the new key value is in the top part of the index page, the system assumes the subsequent keys will also be added in the top part. Therefore, it stores the top part in the left-hand page, using the key value that is one greater than the added key value as the partitioning position. If the location of the new key value is in the bottom part of the index page, the system assumes the subsequent keys will also be added in the bottom part. Therefore, it stores the bottom part in the right-hand page, using the key value of the added key as the partitioning position. The following figure shows an example of an unbalanced index split in which a key is added to the bottom part of an index page.

Figure 3–32: Example of unbalanced index split


1. B-tree structure of index before unbalanced index split
Higher-order page
8
14
Key 7 added.
7
2
4
6
8
10
12
14
Index page a
Index page c
2. Unbalanced index split
Higher-order page
6
8
14
2
4
6
7
8
10
12
14
Index page a
Index page b
Index page c
Legend:
: Locations where the index structure was changed by an unbalanced index split.
: Key that was moved to another page by an unbalanced index split.
Because the key was added to the second half of index page a, data is stored to allocate more free space in index page b.

**Explanation**

Key 7 is added to the second half of index page a, so key 7, as well as key 8 that follows, are moved to index page b on the right side.

**Application criteria**

Use of unbalanced index splitting is advisable in the following cases, because it can improve the data storage efficiency and reduce the number of times index splitting is necessary:

- The index has a uniform degree of duplication of key values
- The index's key values are roughly uniform in size
- Index is one to which consecutive intermediate key values are added frequently

**Specifying an unbalanced index split**

For an unbalanced index split, you specify the `UNBALANCED SPLIT` option in the `CREATE INDEX` or `CREATE TABLE` of the definition SQL. For details about the unbalanced index split option, see the *HiRDB Version 9 System Operation Guide*.

## 3.4.4 Exception key value

All data values in a column for which an index is defined, even null values, are incorporated into the index as index key values. However, null-value keys in an index serve no purpose, so their presence in the index is wasteful of space. Therefore, if an entire column in an index is redundant null values, the null value can be specified as an exception key value for the index. Assigning an exception key value to an index has the following benefits:

**Expected benefits**

1. Because null-value keys are not used in the index, the size of the index is reduced.

2. Overhead for index maintenance (CPU time, number of I/O operations, number of lock requests, frequency of deadlock, etc.) during row insertion, deletion, and updating operations is reduced.
3. If the only retrieval condition for a column of an index that has the null value as an exception key value is `IS NULL`, that index will not be used. Consequently, retrieval performance would be improved in the following situation:
   - When input/output processes have occurred on the same page because data pages were accessed randomly using an index that contained many redundant null values.

**Specifying an exception key value**

To set an exception key value, you specify the `EXCEPT VALUES` option in the `CREATE INDEX` of the definition SQL. For details about index exception key values, see the *HiRDB Version 9 Installation and Design Guide*.

## 3.4.5 Defining an index for a table that contains data

Creation of an index for a table that contains a large amount of data (execution of the `CREATE INDEX`) is a time-consuming process that delays execution of other definition SQL statements.

The `EMPTY` option in the `CREATE INDEX` enables you to define an index without actually creating the index entries. Such an index is called an *unfinished index*. Because no index entries are created, execution of the `CREATE INDEX` terminates immediately and other definition SQL statements can be executed.

Because the index entries are not created, it is not possible to perform searches using an unfinished index or to update the columns in the table that define the unfinished index (an SQL error might result). The index entries can be created by executing the index re-creation facility `(-k ixrc)` of the database reorganization utility (`pdrorg`). When the index entries are created, the index is released from unfinished index status. Deleting all data in the table with the `PURGE TABLE` statement also resets indexes to the table from unfinished index status.

For details about how to use the `EMPTY` option, see the *HiRDB Version 9 System Operation Guide*.

## 3.4.6 Index key value no-lock

*Index key value no-lock* is the process by which a table is accessed by locking only the table data without locking its index key values.

The index key value no-lock option prevents the following problems:

- Deadlock between data-updating and index-search processes
- Unreasonable delayed access to data with the same key
- Unreasonable delayed access to data with different keys

When the index key value no-lock option is used, search processing based on an index does not lock the index key values. In the case of table updating (inserting rows, deleting rows, updating column values), the index key values for any index that is defined on the basis of a column being updated are not locked.

### (1) Application criteria

The index key value no-lock option can be used for all operations. However, determine whether it is to be used by taking into consideration the operation of unique indexes, the presence of any residual entries, and the size of the index log. For an explanation of uniqueness constraint assurance processing for unique indexes and residual entries, see Section *(4) Notes*. For the implications of the size of the index log when the index key value no-lock option is used, see the explanation on estimating the size of the index log and the number of locked resources in the *HiRDB Version 9 Installation and Design Guide*.

### (2) Specification

To use the index key value no-lock option, specify `NONE` in the `pd_indexlock_mode` operand in the system common definition.

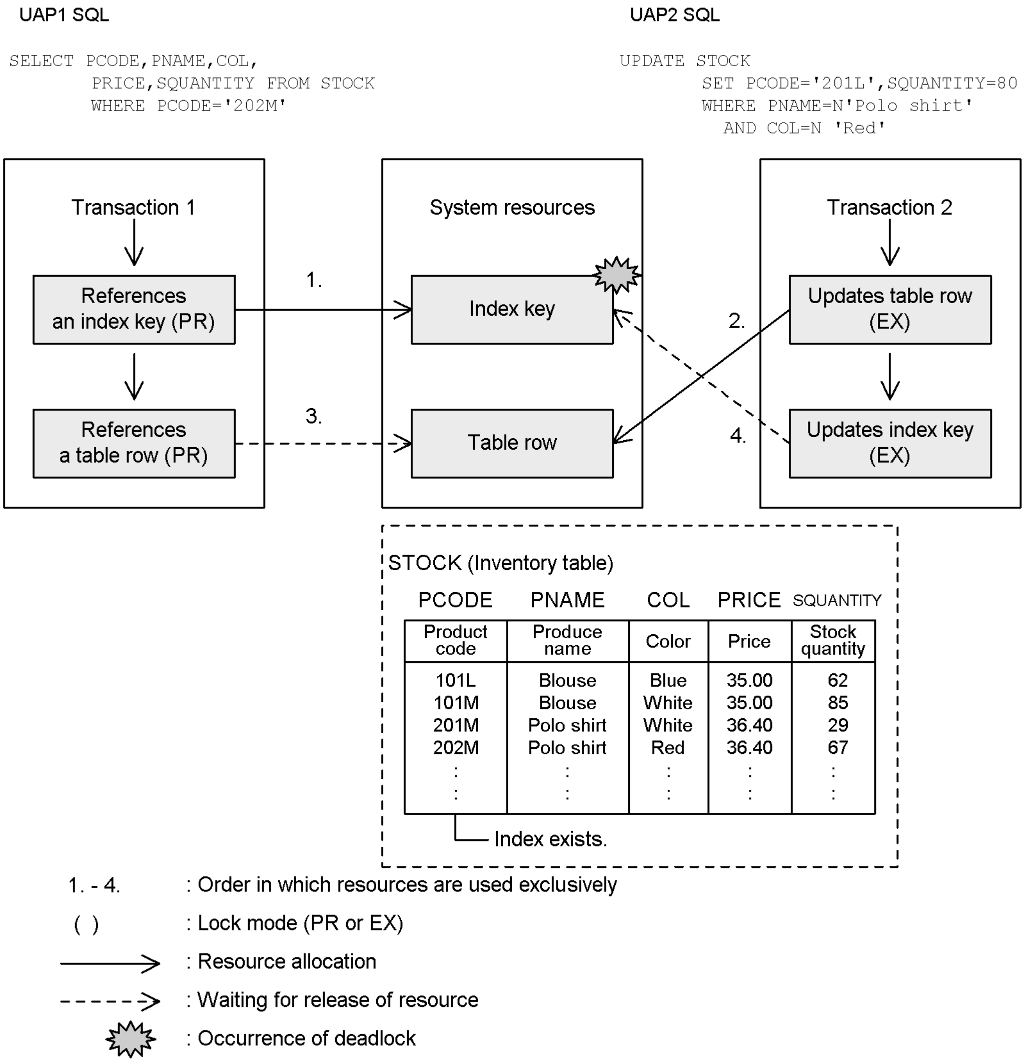
If you are using the UNIX edition, note the following:

> If a value greater than `1` is specified for the `pd_inner_replica_control` operand of the system definition, `NONE` is assumed for the `pd_indexlock_mode` operand of the system definition, regardless of the value specified for it.

## (3) Example of preventing deadlock by using the index key value no-lock option

This section explains how to prevent deadlock by using the index key value no-lock option. The following figure shows an example of a deadlock that is described in this section. For a description of the lock mode, see Section *6.10.2 Lock modes*.

Figure 3–33: Example of deadlock when the index key value no-lock option is not used



UAP1 SQL

```
SELECT PCODE,PNAME,COL,
       PRICE,SQUANTITY FROM STOCK
       WHERE PCODE='202M'
```

UAP2 SQL

```
UPDATE STOCK
       SET PCODE='201L',SQUANTITY=80
       WHERE PNAME=N'Polo shirt'
         AND COL=N 'Red'
```

STOCK (Inventory table)

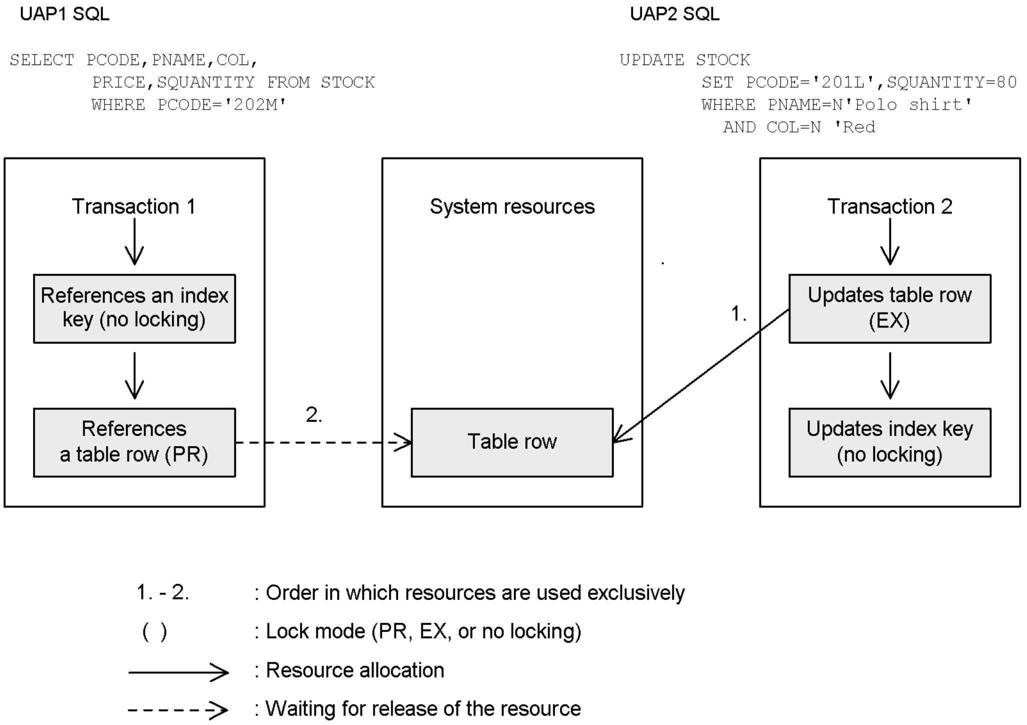
| PCODE<br>Product code | PNAME<br>Produce name | COL<br>Color | PRICE<br>Price | SQUANTITY<br>Stock quantity |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| : | : | : | : | : |
| : | : | : | : | : |

Index exists.

| | |
|---|---|
| 1. - 4. | : Order in which resources are used exclusively |
| ( ) | : Lock mode (PR or EX) |
| ——→ | : Resource allocation |
| - - - - -→ | : Waiting for release of resource |
| ✸ | : Occurrence of deadlock |

Specifying the index key value no-lock option prevents the type of deadlock shown in Figure 3-33. The following figure shows an example in which deadlock is prevented by using the index key value no-lock option.

Figure 3–34: Example of preventing deadlock by using the index key value no-lock option



UAP1 SQL

```
SELECT PCODE,PNAME,COL,
        PRICE,SQUANTITY FROM STOCK
        WHERE PCODE='202M'
```

UAP2 SQL

```
UPDATE STOCK
        SET PCODE='201L',SQUANTITY=80
        WHERE PNAME=N'Polo shirt'
          AND COL=N 'Red
```

Transaction 1
References an index key (no locking)
References a table row (PR)
2.
System resources
Table row
1.
Transaction 2
Updates table row (EX)
Updates index key (no locking)
1. - 2. : Order in which resources are used exclusively
( ) : Lock mode (PR, EX, or no locking)
: Resource allocation
: Waiting for release of the resource

## (4) Notes

### (a) Operation of uniqueness constraint assurance processing for unique indexes

In the case of a table for which the uniqueness constraint is specified, the operation of the uniqueness constraint assurance processing that is performed during addition or updating of rows depends on whether the index key value no-lock option is used. Uniqueness constraint assurance processing is the processing by which locking is applied during insertion of row data or updating of column values in order to ensure the uniqueness of a key value being added; key value uniqueness is determined by checking the index (unique index) to see whether the keyed data to be added already exists. If uniqueness constraint assurance processing locates an index key entry that uses the same key, an error occurs. Even if another party's transaction that is manipulating the index key is incomplete, and there is a possibility of a rollback being performed, an error occurs without a lock check being performed.

If you are performing table data insertion or updating processing with the uniqueness constraint specified and you want continuation of processing to take priority over waiting, apply the index key value no-lock option. If the insertion or updating processing must take priority, even if doing so will create delay, do not apply the index key value no-lock option.

### (b) Residual entries in a unique index

When the index key value no-lock option is used, either lock-wait or deadlock can arise in the unique index. With a unique index in the index key value no-lock option, any index key prior to the execution of the `DELETE` or `UPDATE` statement is left intact, instead of being deleted from the index. This is to maintain the uniqueness of the index. The remaining index key is called a residual entry. Although residual entries are deleted at an appropriate time when the transaction has been settled, if the `INSERT` or `UPDATE` statement is executed on the same key as a residual entry, it is possible that an unexpected delay or deadlock will result.

These problems can be prevented by creating the UAP so that it will not update any columns that are subject to the uniqueness constraint.

### (5) Deadlocks that cannot be prevented even with the index key value no-lock option

A deadlock between index keys can arise depending on the order in which accesses are made by a UAP. To prevent this, you must create UAPs so that they will not update columns that are subject to the uniqueness constraint.

# 3.5 Expansion into an object relational database

By applying object oriented concepts to a relational database model, you can build an *object relational database management system*.

HiRDB is capable of handling data with a complex structure, such as multimedia data, as well as operations performed on such data, by integrating this type of data as objects and managing them in a database. This means that SQL statements can be used to manipulate multimedia data, as in the case of a conventional relational database. For example, you can manage and manipulate the following types of multimedia data:

- SGML/XML structured text data

  You can perform operations such as text searches and highlighting of retrieval hit positions by means of structure specifications.
- Spatial data

  You can search for spatial data (two-dimensional data) such as map information.

Functions for manipulating SGML/XML structured text data and spatial data are provided by plug-ins. For details about how to use plug-ins, see *10. Plug-ins*.

## 3.5.1 Abstract data types

By using *abstract data types*, which are user-defined types, together with various routines, you can uniquely define and use data with complex structures and perform operations on such data. When you define a column as having an abstract data type, you can conceptualize and model its data based on object-oriented concepts. In addition, by applying object-oriented software development techniques, you can reduce the workload for database design, UAP development, and maintenance.

HiRDB makes it possible to use definition SQL statements to define unique abstract data types and their structures. An abstract data type can be handled as a column data type in the same way as HiRDB's predefined data types, such as the numeric type and the character type. Operations on the values of abstract data types can also be defined as routines by using definition SQL statements. In a UAP, you can use a routine to code complex operations on abstract data types in SQL.

Abstract data types, routines, and their characteristic concepts are explained below by means of examples.

### (1) Defining an abstract data type

An example of managing and manipulating employee information in a database using an abstract data type is explained below.

Let's assume that employee information consists of such items as name, sex, date of employment, position, and salary. Let's also assume that image information such as an ID photo is also part of the employee information. Calculation of service years is one possible operation on employee information.

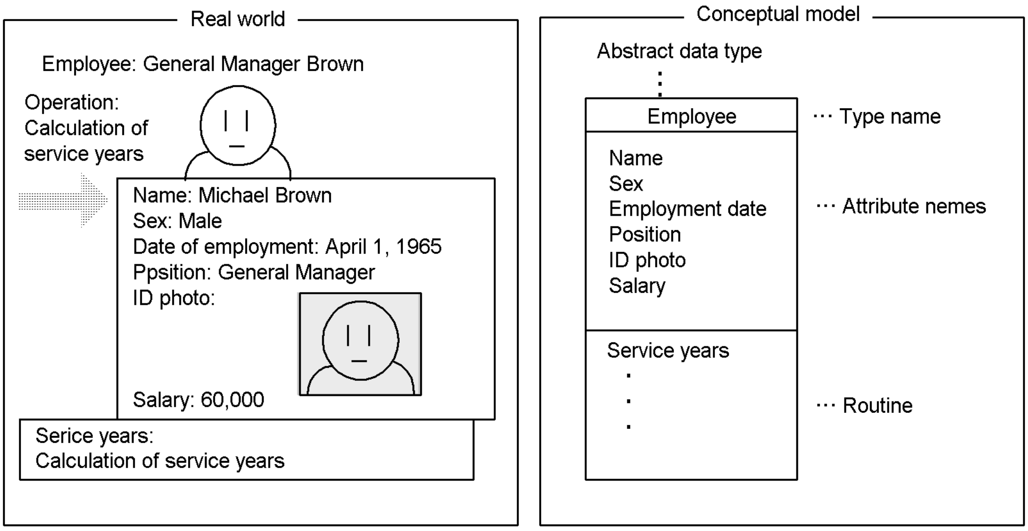
When this information is handled in a database, the abstract concept of *employee* in a data model can be considered to consist of attributes such as *name*, *sex*, *employment date*, *position*, *ID photo*, and *salary*, all of which are common to this concept. An operation such as *calculation of service years* for an employee can also be considered to be an operation that indicates an employee characteristic.

*Employee* can capture all these characteristics (attributes and operations) in a single concept.

In HiRDB, you can address as abstract concepts and handle in a database objects that exist in the real world, and then use abstract data types to define such concepts as data types.

The following figure shows a conceptual model based on real-world information and an abstract data type.

Figure 3–35: Conceptual model based on real-world information and an abstract data type



An abstract data type can be defined in a database using the CREATE TYPE definition SQL shown as follows.

```
CREATE TYPE t_employee (
    Name        CHAR(16),
    Sex         CHAR(1),
    employment_date   DATE,
    position    CHAR(10),
    id_photo    BLOB(64K),
    salary      INTEGER,
    ....
    FUNCTION    service-years ( p t_employee )
       RETURNS  INTEGER
     BEGIN
      DECLARE service_years INTERVAL YEAR TO DAY;
      SET service_years = CURRENT_DATE -
p..employment_date;
      RETURN YEAR(service_years);
     END,
    ....
 )
```

In this way, the user can use an abstract data type to define new data types by specifying attributes and operations.

## (2) Abstract data type as a data type

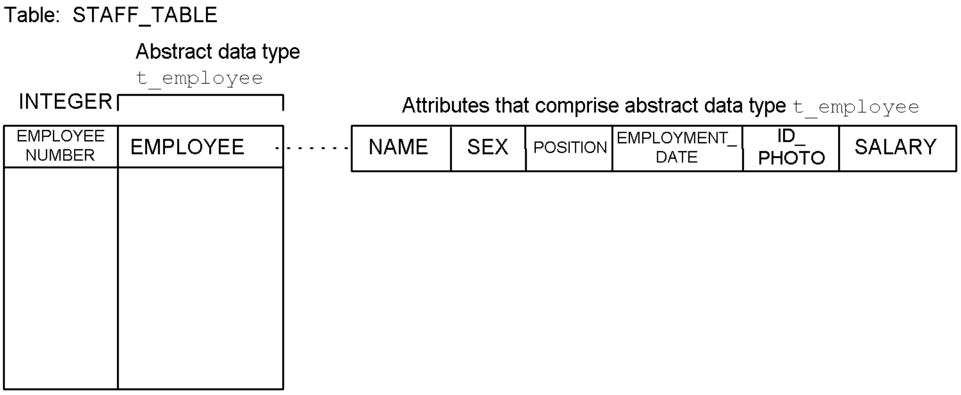
An abstract data type can be handled in the same way as the HiRDB system default data types, such as the numeric type and the character type. For example, the table STAFF_TABLE can be defined by the definition SQL shown below using the abstract data type t_employee as a column data type:

```
CREATE TABLE STAFF_TABLE (
     employee_number   INTEGER,
     employee          t_employee
                        ALLOCATE(id_photo IN(lobarea))
     )
```

When the BLOB type is to be used as one of the attributes of an abstract data type, the user LOB RDAREA in which the data is to be stored must be specified with the ALLOCATE option in CREATE TABLE. In the example above, id_photo, which is an attribute of t_employee, is of the BLOB type, so ALLOCATE is used to store it in user LOB RDAREA lobarea.

The following figure shows a table called STAFF_TABLE, for which an abstract data type is defined.

Figure 3–36: STAFF_TABLE for which an abstract data type is defined



## (3) Encapsulation

When an abstract data type is used, an application can handle the abstract data type values without knowing the detailed configuration of the individual attributes or the installed routines; it does this by using routines declared in that abstract data.

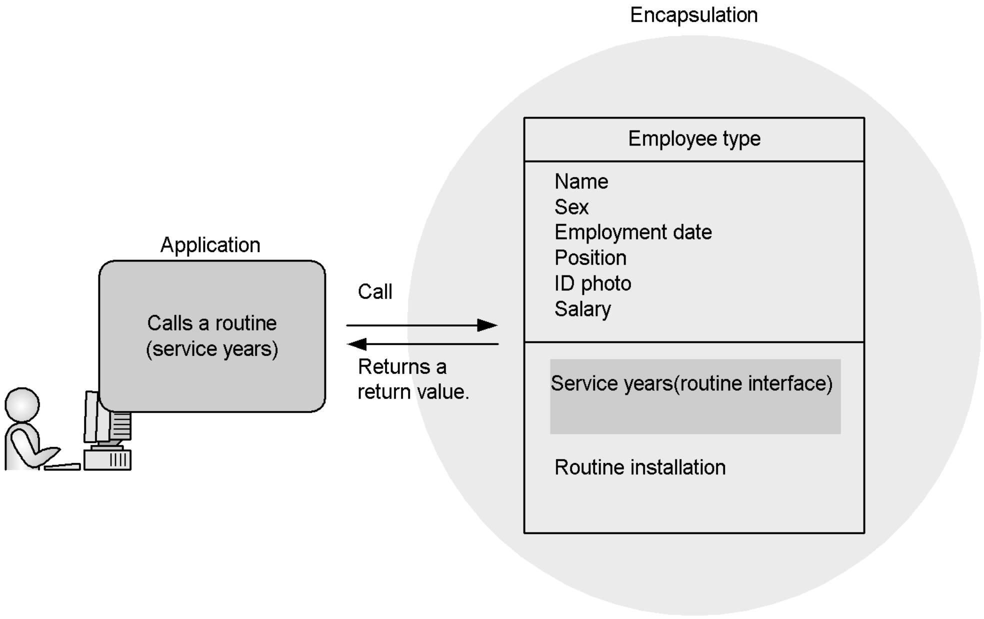
For example, it is possible to manipulate `t_employee` type values using the following data manipulation SQL:

```
SELECT employee_number, employee_name,
       service-years(employee)
  FROM  STAFF_TABLE
```

Handling of values based on an abstract data type and using only an external interface without being concerned about the information in the values is called encapsulation.

The following figure provides an overview of encapsulation.

Figure 3–37: Encapsulation



## (4) Abstract data type values

### (a) Value generation

By executing a function without arguments that is recognized by the same name as an abstract data type, HiRDB can generate values for that abstract data type.

For example, for the `t_employee` type, `t_employee` type values can be generated using the function `t_employee()`.

A function that generates abstract data type values is called a constructor function.

```
BEGIN                     ... Start of SQL procedure.
   DECLARE p t_employee;  ... Declares a t_employee type variable.
   SET p = t_employee();  ... Generates a t_employee type value
                            and substitutes in the variable.
   SET p..name = 'Michael Brown' ... Setting of attribute value
                                   through component
                                   specification.
   RETURN p;              ... Returns the function's return value
                            (returns a t_employee type value).
END                       ... End of SQL procedure.
```

When `CREATE TYPE` is used to define an abstract data type in the database, HiRDB automatically defines a function such as `t_employee()` that has the same name as the data type but has no argument. Such a function is called the default constructor function.

### (b) User-defined constructor function

The user can also define a constructor function.

A constructor function is defined by defining in a `CREATE TYPE` routine declaration a function that has the same name as the abstract data type to be defined and that uses the abstract data type as the return value type.

```
CREATE TYPE t_employee (
      name                CHAR(16),
      sex                 CHAR(1),
      employment_date     DATE,
      position            CHAR(10),
      id_photo            BLOB(64K),
      salary              INTEGER,

  FUNCTION t_employee(
      p_name              CHAR(16),
      p_sex               CHAR(1),
      p_employment_date   DATE,
      p_position          CHAR(10),
      p_id_photo          BLOB(64K),
      p_salary            INTEGER)
      RETURNS             t_employee
  BEGIN
      DECLARE    d_employee t_employee;
        SET      d_employee = t_employee();
        SET      d_employee..name       = p_name;
        SET      d_employee..sex        = p_sex;
        SET      d_employee..employment_date
                                        = p_employment_date;
        SET      d_employee..position   = p_position;
        SET      d_employee..id_photo   = p_id_photo;
        SET      d_employee..salary     = p_salary;

      RETURN d_employee;
  END,
  ...
  )
```

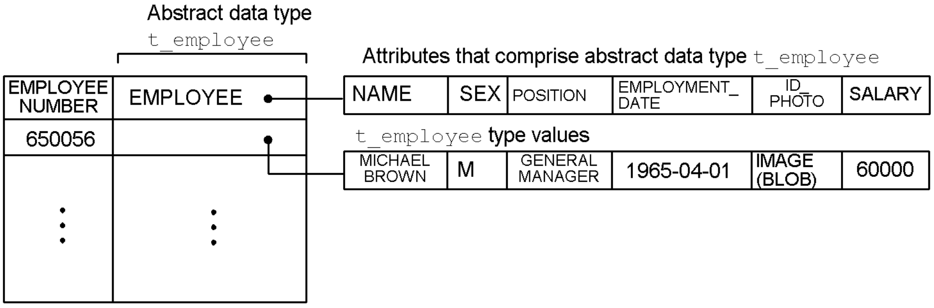
For example, using the user-defined constructor function t_employee() and the following data manipulation SQL, values can be generated and stored in a database:

```
INSERT INTO STAFF_TABLE
  VALUES (
          650056, t_employee(:name AS CHAR(16),
                                      :sex AS CHAR(1),
                                      :yrs AS DATE,
                                      :post AS CHAR(10),
                                      :picture AS BLOB(64K),
                                      :salary AS INTEGER)
          )
```

The following figure shows a table called `STAFF_TABLE`, for which values are generated by constructor function `t_employee()` and are inserted as column values.

Figure 3–38: STAFF_TABLE for which values are generated using a constructor function

Table: STAFF_TABLE



## (5) Abstract data type null values

Null values can also be applied to an abstract data type, in the same manner as with the HiRDB system default types. For example, executing the following data manipulation SQL for the aforementioned `STAFF_TABLE` places null values in the `EMPLOYEE` column:

```
INSERT INTO STAFF_TABLE(employee_number)
                  VALUES(650056)
```

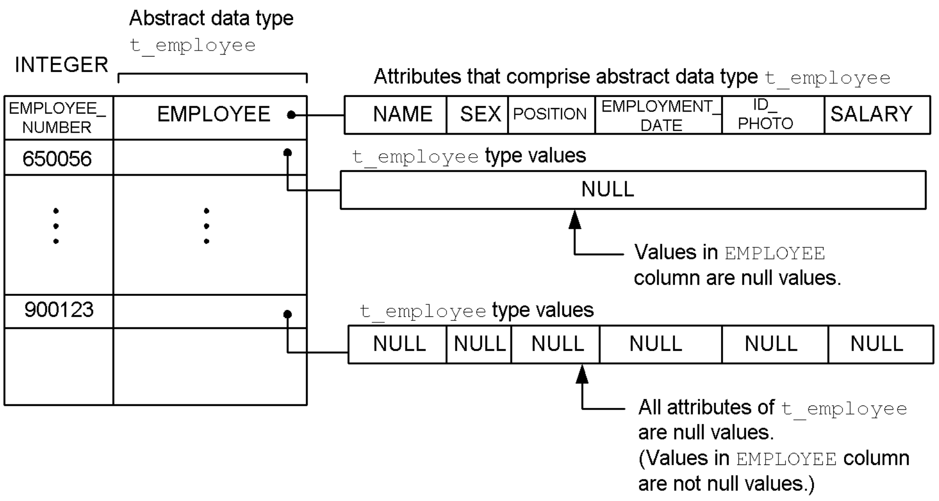
On the other hand, executing the following data manipulation SQL changes all `t_employee` type attribute values to null values (the column values of the abstract data type in which all attribute values are null values are regarded as values other than null values):

```
INSERT INTO STAFF_TABLE (900123, t_employee ())
```

The following figure shows the handling of null values in `STAFF_TABLE` for which an abstract data type is defined.

Figure 3–39: Handling of null values in STAFF_TABLE for which an abstract data type is defined

Table: STAFF_TABLE


Abstract data type
t_employee
INTEGER
Attributes that comprise abstract data type t_employee
EMPLOYEE_ NUMBER
EMPLOYEE
NAME
SEX
POSITION
EMPLOYMENT_ DATE
ID_ PHOTO
SALARY
650056
t_employee type values
NULL
Values in EMPLOYEE column are null values.
900123
t_employee type values
NULL
NULL
NULL
NULL
NULL
NULL
All attributes of t_employee are null values.
(Values in EMPLOYEE column are not null values.)

For example, executing the following data manipulation SQL retrieves the employee numbers of those employees whose values in the EMPLOYEE column are not null values (therefore, the employee numbers of those employees who have all of the t_employee type attributes as null values will not be retrieved):

```
SELECT employee_number FROM STAFF_TABLE
    WHERE employee IS NOT NULL
```

Retrieval result:

```
900123
```

## (6) Manipulation of abstract data type values

Let's plan an operation for calculating an employee's service years with the company.

Operations for abstract data type values can be defined in HiRDB by using a routine declaration in CREATE TYPE. For example, an operation such as for calculating service years and compensation rate can be defined with the following definition SQL:

```
CREATE TYPE t_employee (
        name              CHAR(16),
        sex               CHAR(1),
        employment date   DATE,
        position          CHAR(10),
        id-photo          BLOB(64K),
        salary            INTEGER,
      ........
        FUNCTION  service years ( p t_employee )
           RETURNS INTEGER
         BEGIN
          DECLARE service_years INTERVAL YEAR TO DAY;
           SET service_years
           = CURRENT_DATE - p..employment date;
          RETURN YEAR(service_years);
         END,
       .......
  )
```

In this way, routines defined for an abstract data type can be used for abstract data type values. For example, an SQL for retrieving the employee number and employee name of each employee with a service years value of 10 or more years can be described as follows:

```
SELECT employee number, employee..name, service years (employee)
   FROM  STAFF_TABLE
   WHERE service years(employee) >= 10
```

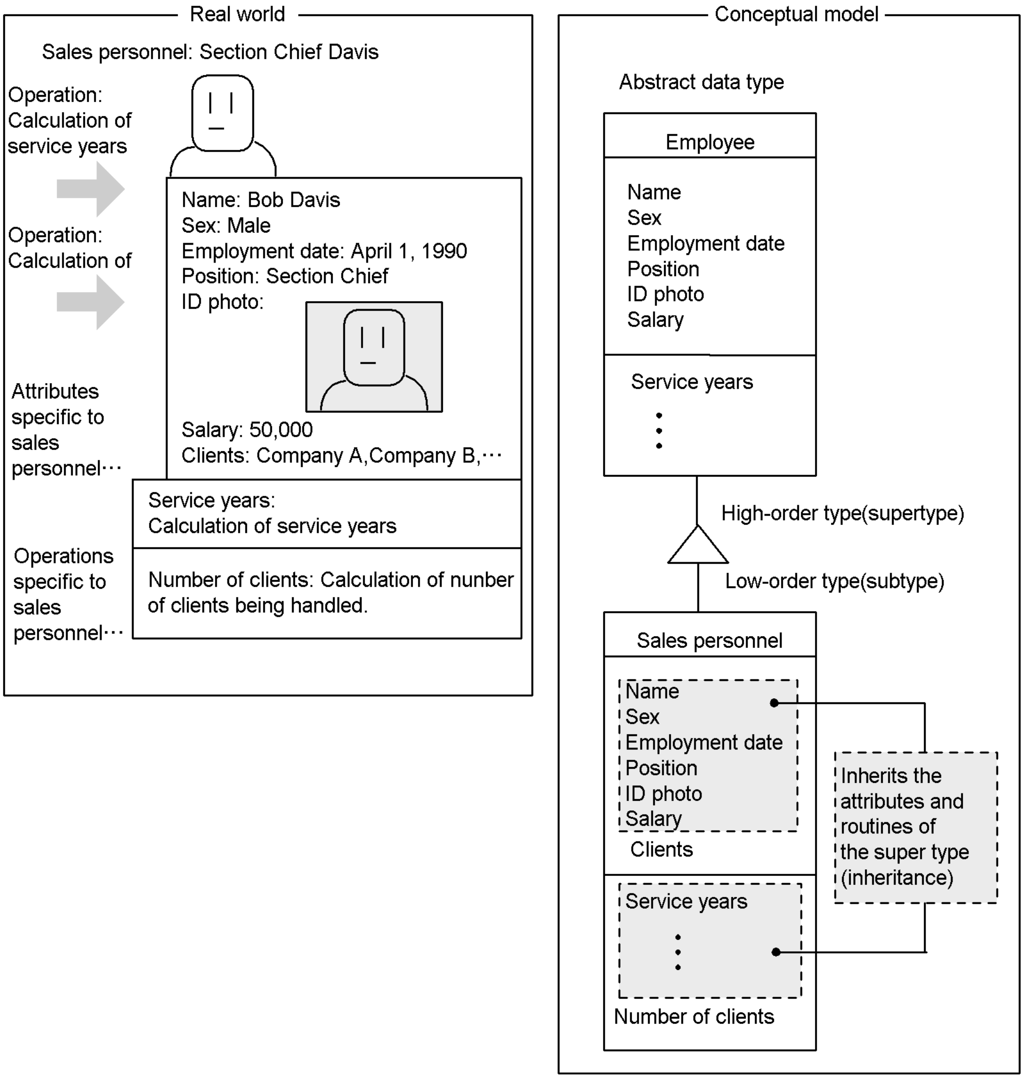
## 3.5.2 Subtypes and inheritance

### (1) Subtype

Let's consider managing the information for employees who are in sales. *Salesperson* can be considered a more specific and specialized (particular) concept in comparison to the abstract concept of *employee*. The information on a salesperson might contain information related to sales activities in addition to the information common to all employees. Therefore, attributes such as *charge client* and *number of clients* are assigned to a salesperson, in addition to being an employee.

The following figure shows a conceptual model for a salesperson based on both real-world information and an abstract data type.

Figure 3–40: Conceptual model based on real-world information and an abstract data type (for a salesperson)


Real world
Sales personnel: Section Chief Davis
Operation: Calculation of service years
Operation: Calculation of
Attributes specific to sales personnel···
Operations specific to sales personnel···
Name: Bob Davis
Sex: Male
Employment date: April 1, 1990
Position: Section Chief
ID photo:
Salary: 50,000
Clients: Company A,Company B,···
Service years: Calculation of service years
Number of clients: Calculation of nunber of clients being handled.
Conceptual model
Abstract data type
Employee
Name
Sex
Employment date
Position
ID photo
Salary
Service years
High-order type(supertype)
Low-order type(subtype)
Sales personnel
Name
Sex
Employment date
Position
ID photo
Salary
Clients
Service years
Number of clients
Inherits the attributes and routines of the super type (inheritance)

HiRDB makes it possible to take a particular data type and tailor it so as to define an abstract data type as a subtype.

For example, the subtype clause of a `CREATE TYPE` definition SQL statement can be used to define `t_salesperson` based on `t_employee`, as shown below:

```
CREATE TYPE t_salesperson UNDER t_employee(
      charge client  VARCHAR(3000),

      FUNCTION number of clients (..........)
RETURNS INTEGER
  ...........
)
```

Note that a high-order abstract data type, such as `t_employee`, is called a supertype.

## (2) Substitutability

*Salesperson* is also an *employee*. HiRDB can handle an abstract data type value that was specialized (low-order) in a subtype also as a high-order abstract data type value. For example, the SQL shown below can substitute the `t_salesperson` values as `t_employee` column values in `STAFF_TABLE`.

**Note**

To handle `t_salesperson` values in the same manner as `t_employee` values, it is necessary to re-create SQL objects by executing `ALTER ROUTINE` before executing this SQL.

```
INSERT INTO STAFF_TABLE VALUES ( 51, t_salesperson(:name AS
CHAR(16),.....) )
```

The capability of a low-order abstract data type value to be considered as a high-order abstract data type value in this way is called substitutability.

## (3) Inheritance

Because *salesperson* is also an *employee*, it also has attributes, such as *name* and *sex*, just like *employee*, which makes it possible to calculate *service years* for a salesperson.

Figure 3-40 shows that a lower-order abstract data type in a subtype inherits the attributes and routines defined for the higher-order abstract data type.

This manner in which a lower-order abstract data type inherits the attributes and routines of the higher-order abstract data type is called inheritance.

For example, the attribute `name` and the operation `service years` become available for `t_salesperson` values through inheritance. Consequently, as shown below, the SQL shown above can be executed without any changes for `STAFF_TABLE` into whose columns `t_salesperson` values have been inserted.

**Note**

Before this SQL can be executed, it is necessary to re-create SQL objects by executing `ALTER ROUTINE`.

```
SELECT employee number, employee..name, service years (employee)
  FROM  STAFF_TABLE
  WHERE service years(employee) >= 10
```

Subtype and inheritance can provide the following benefits:

- The concept of substitutability (for example, a salesperson is also an employee) can be expressed quickly and clearly.
- Attributes and routine definitions can be shared, and new definitions can be added based on existing definitions. Consequently, the overhead for database and application development can be reduced, and a system with expandability can be constructed.

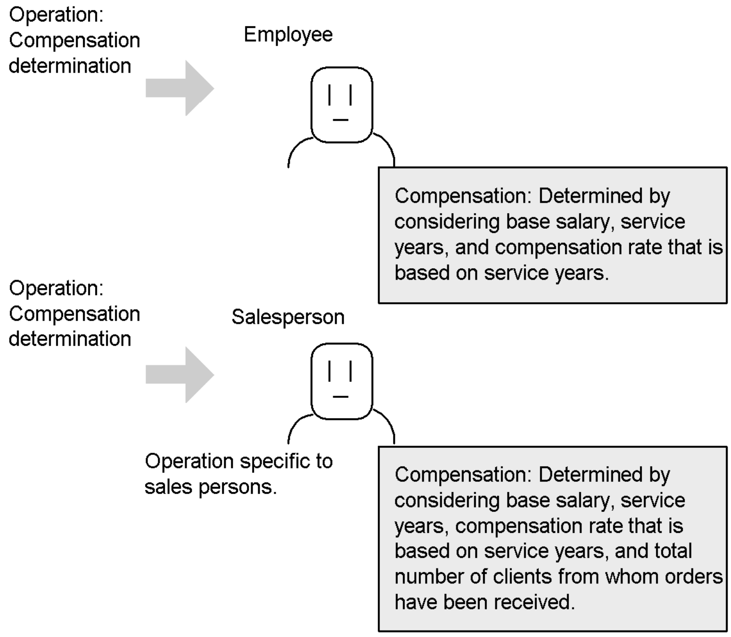
## (4) Override

Let's consider an operation for determining compensation for an employee. For example, compensation can be determined based on the information that is set up by salary, service years, and compensation rate based on service years and on work performance.

Meanwhile, the operation *compensation* to be defined for *employee* is inherited by *salesperson*, as explained above. However, the method of determining compensation for salespersons might be different from that used for ordinary employees, and might include such attributes as *number of clients* that are specific to salespersons.

The following figure shows operations related to employees and salespersons in the real world.

Figure 3–41: Operations related to employees and salespersons in the real world


Operation:
Compensation
determination
Employee
Compensation: Determined by considering base salary, service years, and compensation rate that is based on service years.
Operation:
Compensation
determination
Salesperson
Operation specific to sales persons.
Compensation: Determined by considering base salary, service years, compensation rate that is based on service years, and total number of clients from whom orders have been received.

Here, let's define routines, such as `employee compensation` and `salesperson compensation`, that have different names for each of the abstract data types.

In this case, despite the fact that *employee* values and *salesperson* values can be handled together because of substitutability, it will be necessary to have a different name for the routine to be called for each value type. Consequently, applications will not be able to execute the data manipulation SQL described as follows:

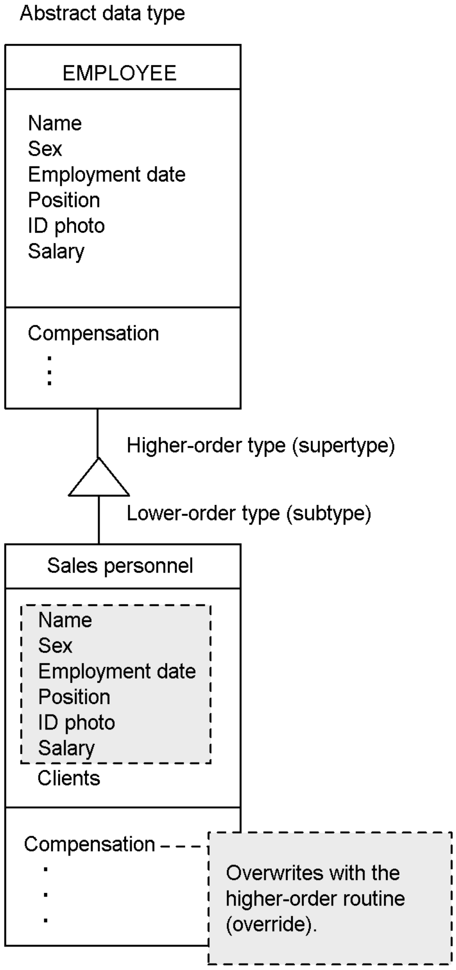
```
SELECT   employee number, employee..name,
         employee compensation(employee)
    FROM     STAFF_TABLE
... Determination of compensation specific to salesperson cannot be executed.

SELECT   employee number, employee..name,
         salesperson compensation (employee)
    FROM     STAFF_TABLE
... Determination of compensation specific to salesperson will be executed
also for employees who are not salespersons.
```

In HiRDB, a routine that has the same name as a routine defined in a higher-order abstract data type can be overwritten during the definition of a lower-order abstract data type. This process of overwriting is called override.

The following figure shows an example of override.

Figure 3–42: Override



HiRDB automatically executes an overridden routine based on the definition appropriate to the type of value that is used as the argument.

Override eliminates the need to change the name of the routine to be called, according to the value type. Therefore, it is possible to execute the data manipulation SQL shown as follows.

**Note**

Before this SQL can be executed, it is necessary to re-create SQL objects by executing ALTER ROUTINE.

```
SELECT   employee_number, employee..name,
         compensation (employee)
    FROM     STAFF_TABLE
```

*... The overriding routine compensation executes a routine that is appropriate to each argument value.*

During execution of this SQL, the routine compensation defined for the t_employee type is executed for the t_employee type values, and the routine compensation defined for the t_salesperson type is executed for the t_salesperson type values.

An application can call a routine without being concerned with whether that routine has been overridden, as in the above SQL. Even when routines are added through override, it is possible to execute an SQL without changing the application.

### 3.5.3 Encapsulation

It might be desirable to prevent applications from directly accessing some of the information related to employees, such as personal information (such as information to be processed in the compensation operation and the value of the employment date attribute). The reasons for this are listed as follows:

- Direct access from an application is not desirable in order to protect confidentiality.

  *Example*

  The value of the salary attribute can be referenced in internal processing of routines such as compensation, but is not directly accessible from the outside.
- Direct access from an application is meaningless.

  *Example*

  If job performance evaluation results are encoded and held as attribute values, accessing these values from an application will be of no use.
- It is necessary to prevent an application from directly altering internal information.

  *Example*

  It is necessary to prevent an application from changing the value of the employment date attribute to a different date.

#### (1) Encapsulation level

In HiRDB, it is possible to specify an encapsulation level during the declaration of abstract data types and routines.

The following three encapsulation levels are available:

**PRIVATE**

Attribute values can be accessed and routines can be used only within the definition of an abstract data type.

For example, PRIVATE might be specified for the t_employee type attribute employment date.

When PRIVATE is specified for an attribute, it will not be possible to access this attribute or to use a routine from inside a subtype definition or an application.

**PROTECTED**

Attribute values can be accessed and routines can be used only within the definition of an abstract data type and within the definition of subtypes of that abstract data type.

For example, PROTECTED might be specified for the t_employee type routine compensation. This will allow t_salesperson (a subtype of the t_employee type) to execute the compensation rate routine in order to calculate compensation rate, without directly referencing the attribute employment date or knowing the processing details of compensation rate (calculation of service years based on the employment date, and calculation of a compensation rate based on this value).

**PUBLIC**

No restrictions are placed, as with PRIVATE or PROTECTED, and attribute values can be accessed and routines can be used from within the definitions of other abstract data types and subtypes, as well as from applications.

For example, PUBLIC might be specified for the service years routine for which there is no access restriction.

The following table shows encapsulation levels and whether they can access abstract data type values and use routines.

Table 3–9: Encapsulation levels and access types

| Encapsulation level | Access source | | | |
|---|---|---|---|---|
| | Within the definition of an abstract data type | Within the definition of a subtype abstract data type | Within the definitions of abstract data types other than those listed on the left | Applications |
| PUBLIC | Y | Y | Y | Y |
| PROTECTED | Y | Y | N | N |

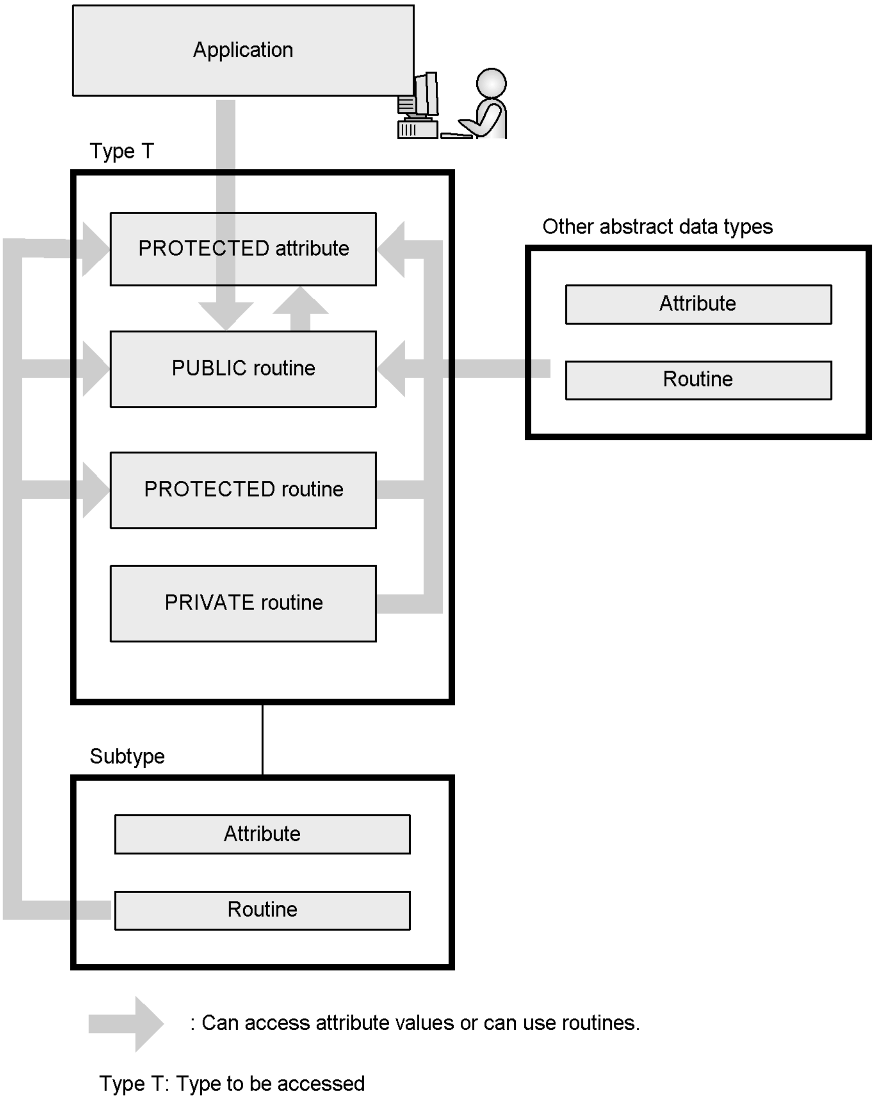
| Encapsulation level | Access source | | | |
|---|---|---|---|---|
| | Within the definition of an abstract data type | Within the definition of a subtype abstract data type | Within the definitions of abstract data types other than those listed on the left | Applications |
| PRIVATE | Y | N | N | N |

Y: Can access abstract data type values and can use routines.

N: Cannot access abstract data type values and cannot use routines (an SQL error occurs).

The following figure shows an example of the relationship between encapsulation levels and access types for abstract data type T.

Figure 3–43: Encapsulation levels and access types



For details about the procedures for designing and creating a table for which an abstract data type is defined, see the *HiRDB Version 9 Installation and Design Guide*. For details about the procedures for manipulating a table for which an abstract data type is defined, see the *HiRDB Version 9 UAP Development Guide*.

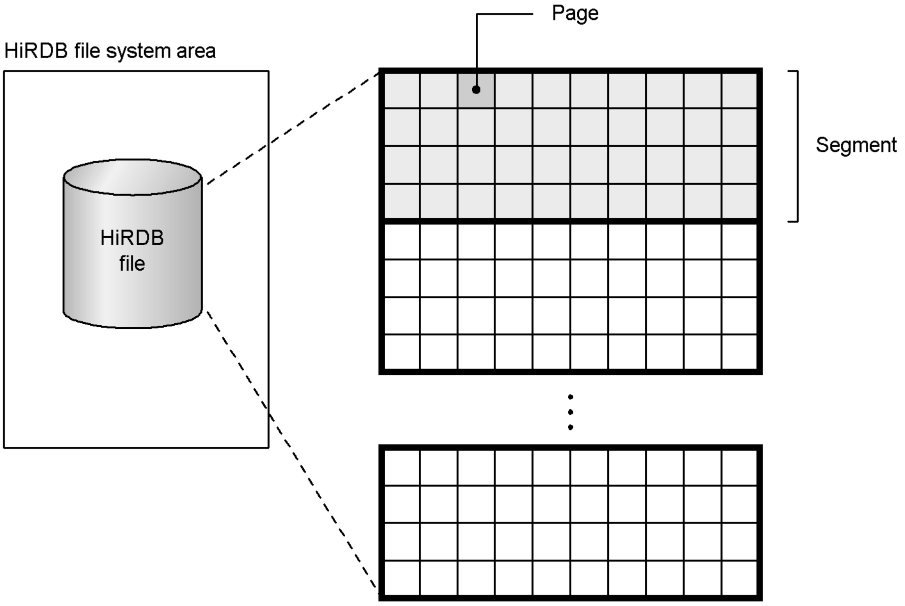
# 4 Database Physical Structure

This chapter explains the physical structure (segments and pages) of a database.

# 4.1 Database physical structure

The following figure shows the physical structure of a database.

Figure 4–1: Database physical structure



**Explanation**

- HiRDB file system area
  This is an area in which HiRDB files are created.
- HiRDB file
  This is a type of file unique to HiRDB that is used to store table and index data.
- Segment
  This is the smallest unit of table and index data storage. One segment stores data from only one table or one index. A segment is composed of multiple, consecutive pages.
- Page
  This is the smallest unit of database I/O operations. If the page size is large, one page can store several consecutive rows, thus reducing the number of I/O operations in situations where the data is processed in the same order in which it is stored. The following table lists the available types of pages.

Table 4–1: Types of pages

| Page type | Explanation |
|---|---|
| Data page | Stores row data for a table. |
| Index page | Stores index key values. |
| Directory page | Stores management information on the status of RDAREAs. |

**Specifying the physical structure of a database**

The physical structure of a database is specified when its RDAREAs are defined. Specifically, the physical structure is specified in the following control statements:

- `create rdarea` statement of the database initialization utility (`pdinit`)
- `create rdarea` statement of the database structure modification utility (`pdmod`)

Following is an example of specifying the `create rdarea` statement:

```
create rdarea USRRD01 for user used by PUBLIC       1
    page 4096 characters                            2
    storage control segment 20 pages                3
    file name "C:\rdarea01\file01"                  4
         initial 150 segments ;                     5
```

**Explanation**

1. Specifies a name (`USRRD01`) for an RDAREA.
2. Specifies 4,096 bytes as the page size.
3. Specifies 20 pages as the segment size.
4. Specifies the name of the HiRDB file system area where the RDAREA will be stored and a HiRDB file name:

   `C:\rdarea01`: Name of HiRDB file system area

   `file01`: HiRDB file name
5. Specifies the number of segments.

# 4.2 Segment design

## (1) Segment statuses

The following table lists and describes the statuses that are assigned to segments.

Table 4–2: Segment statuses

| Segment status | Description |
|---|---|
| Used segment# | A segment in which table or index data is stored.<br>A used segment that is completely filled with data, such that no more can be added, is called a *full segment*. A used segment that is not full is called a *space-available segment*.<br>A space-available segment from which data has been deleted so that only free pages remain (used free pages or unused pages) is called a *used free segment*. |
| Unused segment | A segment that is not being used. This segment can be used by any table or index in the RDAREA. |
| Free segment | A segment in which no data is stored. Both used free segments and unused segments are called free segments. |

#: A used segment can only be used by a table or an index that has stored data in that segment. Other tables or indexes cannot use the segment.

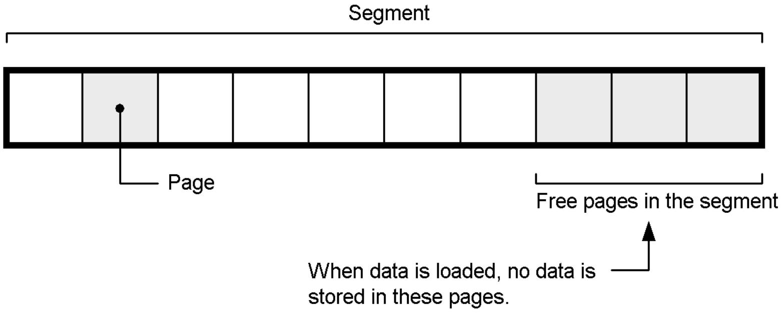
## (2) Segment design policy

Segments must be designed with care, because the segment size (number of pages per segment) affects I/O times and disk space requirements. The recommended segment size for most purposes is 10-20 pages. For details about segment design, see the *HiRDB Version 9 Installation and Design Guide*.

## (3) Ratio of free pages in a segment

When you define a table, you can specify the ratio of free pages in a segment. This concept is illustrated in the following figure.

Figure 4–2: Ratio of free pages in a segment


Segment
Page
Free pages in the segment
When data is loaded, no data is stored in these pages.

**Explanation**

- In this example, the segment free page ratio is set to 30%. Therefore, three out of every 10 pages are free pages.
- The segment free page ratio is specified with the `PCTFREE` option of the `CREATE TABLE`. This ratio can be specified in the 0-50% range; the default is 10%.
- When data is loaded (including when reloading and reorganizing), it is not stored in the free pages specified here.

The data storage efficiency improves as the segment free page ratio becomes smaller.

Increasing the segment free page ratio sometimes improves performance. For example, when data is added to a table for which a cluster key is defined and a segment free page ratio has been set, you can cause data to be stored in positions that are near a cluster key value, which minimizes the number of data I/O operations that will be needed.

For details about determining the segment free page ratio, see the *HiRDB Version 9 Installation and Design Guide*.

## (4) Allocating and releasing segments

Segments are not allocated when the table is defined. Instead, segments are allocated as needed when data is saved to the table. Once a segment is allocated (once it becomes a used segment), no other table or index can use that segment until it is released. Consequently, the RDAREA might run out of free space if data is added and deleted repeatedly, even though the amount of data has not increased. To prevent this from occurring, periodically perform the following operations to release segments:

- Reorganize tables or indexes using the database reorganization utility (`pdrorg` command).
- Release used free segments using the free page release utility (`pdreclaim` command).

For details about reorganizing tables and indexes, and about releasing used free segments, see the *HiRDB Version 9 System Operation Guide*. In addition to these operations, the following operations also release segments:

- Executing the `PURGE TABLE` statement
- Reinitializing the RDAREA
- Deleting the table definition
- Deleting the index definition
- Loading data in the creation mode (`-d` option specification)

## (5) Reusing free space

The free space reuse facility enables you to efficiently utilize free space made available by deleting data.

### (a) How searching is performed when data is saved

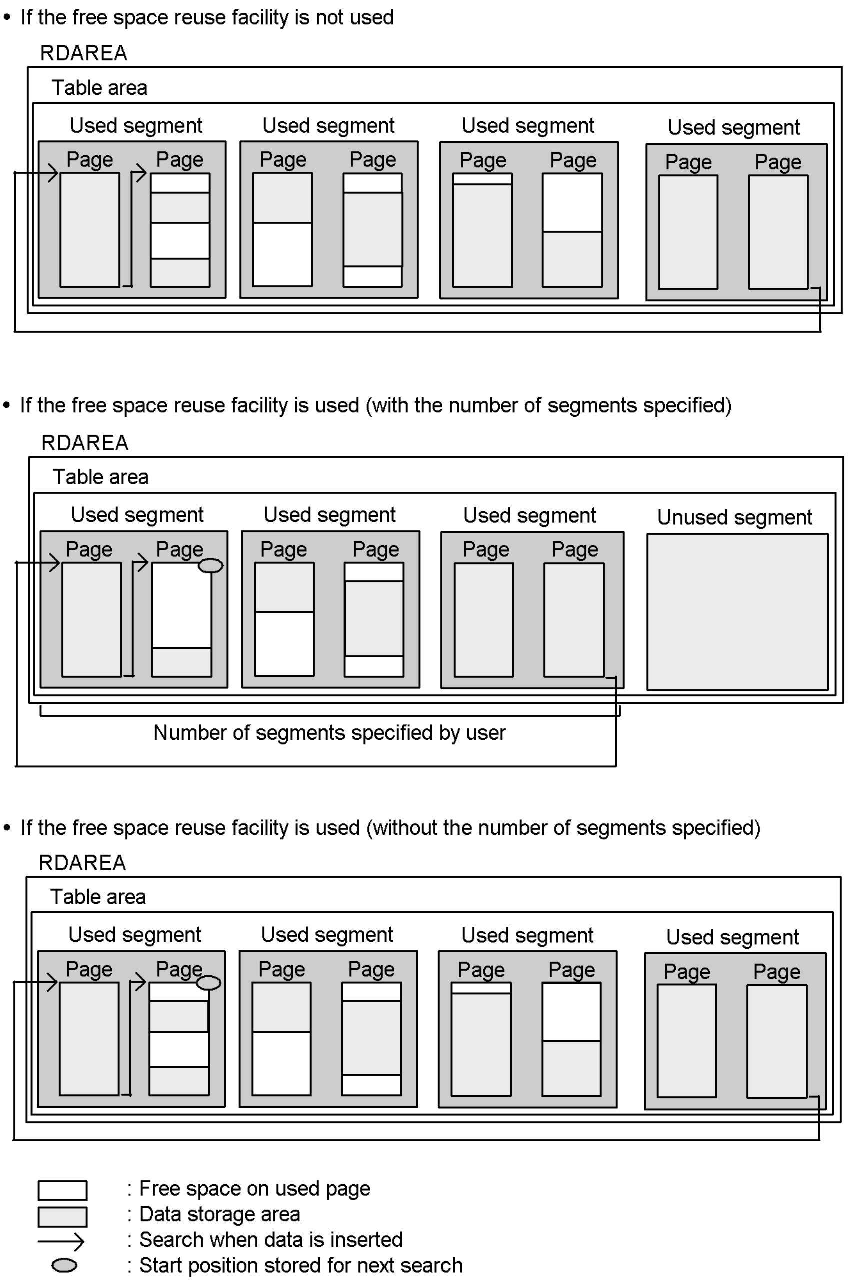
When data is saved to a table, there are two page search modes that are used to search for a storage area:

- New page allocate mode
  In this mode, when the last page in a used segment becomes full, a new unused segment is allocated. If the RDAREA runs out of unused pages, a search is then conducted to find free space in a used page, starting at the beginning of the first used segment. Once free space is found, the data is saved to that free space.
- Free page reuse mode
  In this mode, when the last page in a used segment becomes full, a search is conducted to find free space in used pages starting in the first used segment. If no free space is found, an unused segment is allocated. The next search start position is also remembered, so that, the next time, searching for free space begins at this position.

### (b) Overview of the free space reuse facility

The *free space reuse facility* is designed to enable the free space in used pages to be utilized by activating the free page reuse mode. This occurs when the number of used segments in a table reaches the number of segments specified by the user, and the last segment becomes full. Once the free space in every segment of the specified number of segments runs out, the free space reuse facility activates the new page allocate mode, and a new unused segment is allocated. If the number of segments is not specified, free space is not reused until there are no more unused pages in the RDAREA. If the free space reuse facility is not used, the system searches for free space from the beginning of the used segment every time a search is performed. If the free space reuse facility is used, the free page reuse mode is activated, the next search start position is remembered, and the next search is performed from there. As such, use of this facility enables searches to be performed more efficiently. The following figure provides an overview of the free space reuse facility.

Figure 4–3: Overview of the free space reuse facility


- If the free space reuse facility is not used
- If the free space reuse facility is used (with the number of segments specified)
- If the free space reuse facility is used (without the number of segments specified)

RDAREA
Table area
Used segment
Unused segment
Page
Number of segments specified by user
: Free space on used page
: Data storage area
: Search when data is inserted
: Start position stored for next search

**Explanation**

- If the free space reuse facility is not used

  Every time data is to be inserted after the RDAREA has run out of unused pages, the system searches for free space in a used page, starting at the beginning of the first used segment. Once the space is found, the data is saved to that free space.

- If the free space utility is used (with the number of segments specified)

  When an attempt is made to insert data into a table after the specified number of segments is reached, this facility searches for free space in a used page, starting at the beginning of the first used segment. Once the space is found, it saves the data to that free space. In addition, the facility remembers that location as the next search start position and, the next time, begins searching for free space from this position.

- If the free space utility is used (without the number of segments specified)

When an attempt is made to insert data after the RDAREA has run out of unused pages, this facility searches for free space in a used page, starting at the beginning of the first used segment. Once the space is found, it saves the data to that free space. In addition, the facility remembers that location as the next search start position and, the next time, begins searching for free space from this position.

For details about the free space reuse utility, see the *HiRDB Version 9 System Operation Guide*.

### (c) Application criteria

For operations requiring frequent reorganization, use the free space reuse facility to absolutely minimize the number of reorganizations you need to perform, especially when repeated deletions and insertions of data are using up large numbers of segments.

### (d) Environment settings

The following environment settings are used for the free space reuse facility. For details, see the *HiRDB Version 9 Installation and Design Guide*.

1. In the `pd_assurance_table_no` operand, specify the number of tables that can use the free space reuse facility.
2. Specify the number of segments in the `SEGMENT REUSE` option of the definition SQL `CREATE TABLE` statement. For tables that have already been created, specify this value in the `SEGMENT REUSE` option of the `ALTER TABLE` statement.

### (e) Notes

The free space reuse facility does not operate in the following cases:

- When data is being stored with the hash facility for hash row partitioning
- When a data dictionary table is being stored
- When data is being stored using the data load or database reorganization utility (`pdrorg`)
- When a user LOB RDAREA is being used

# 4.3 Page design

This section describes page statuses and explains page design policy.

## (1) Page statuses

The following table lists and describes the statuses that are assigned to pages.

Table 4–3: Page statuses

| Page status | Description |
|---|---|
| Unused page | A page that has not been assigned. |
| Used free page | A page in which no data is being stored, as a result of data deletion.# |
| Active page | A page on which data is being stored, and that has free space to which data can be added.<br>For a table on which the free space reusage facility is being used, its active pages include those pages to which data could not be added because the free space on the page could not be used due to data deletion.# |
| Used full page | A page on which data is being stored, but that has no free area to which data can be added.<br>For a table or index on which the free space reusage facility is not being used, its used full pages include those pages to which data could not be added because the free space on the page could not be used due to data deletion.# |

#

Free space that is generated as a result of data deletion cannot be used until the transaction that executed data deletion is committed.

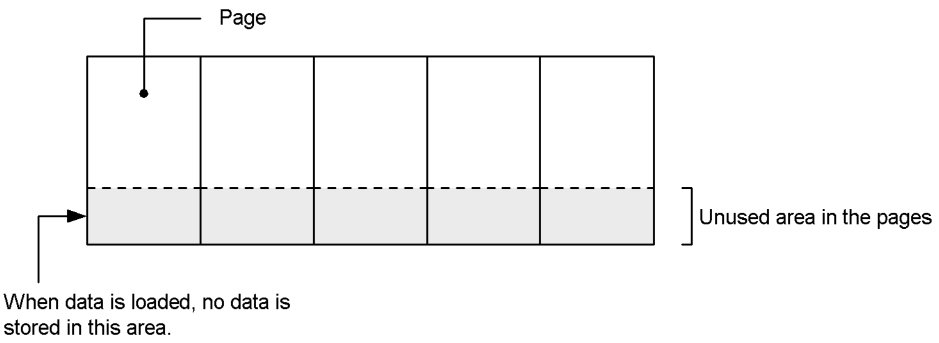
## (2) Page design policy

Pages must be designed with care, because the page size affects data I/O times. For details about page design, see the *HiRDB Version 9 Installation and Design Guide*.

## (3) Ratio of unused area in a page

When you define a table or index, you can specify the ratio of unused area in a page. This concept is illustrated in the following figure.

Figure 4–4: Ratio of unused area in a page



**Explanation**

- In this example, the page unused area ratio is set to 30%.
- The page unused area ratio is specified with the `PCTFREE` option of the `CREATE TABLE` or `CREATE INDEX`. This ratio can be specified in the 0-99% range; the default is 30%.
- When data is loaded into the table, no data is stored in the unused area.

The data storage efficiency improves as the page unused area ratio becomes smaller. Increasing the page unused area ratio sometimes improves performance. For example, when data is updated, you can reduce the number of data I/O operations, provided that either of the following occurs:

- The row size is increased as a result of the updating.
- A row is added to a table for which a cluster key is specified.

For details about determining the page unused area ratio, see the *HiRDB Version 9 Installation and Design Guide*.

## (4) Allocating and releasing pages

### (a) Allocating pages

Pages are not allocated when the table is defined. Instead, pages are allocated as needed when data is saved to the table. Once a page is allocated (once it becomes a used page), it cannot be reused until it is released.

When an index is defined, the system allocates pages according to the number of data items to be included in the index. If there are no data items, only one page (*root page*) is allocated. However, if the `EMPTY` option is specified in the `CREATE INDEX` (index entries not to be created), no pages will be allocated.

Reference note

- If data in a non-FIX table is updated and as a result there is a change in the row size, any area freed up by a reduction in the row size cannot be reused.
- An index page cannot be reused until a key value that is identical to a key value that was stored in the deleted page is added.

! Important note

Reusing a page freed up by deletion of data is subject to the following restrictions:

- The page cannot be used by VARCHARs of 256 bytes or greater, BINARY types, abstract data types, and branch rows of repetition columns.
- The page cannot be used when data is inserted until the segment utilization rate reaches 100%.
- A transaction that executes DELETE cannot use the space freed by the DELETE until COMMIT is issued.

### (b) Releasing pages

- When a segment is released, the pages in that segment are released.
- When a UAP deletes all rows in a page in a table that is locked with the `LOCK` statement in the `EXCLUSIVE` specification, that page is released.
- Because the used segments in a table or index are released when `PURGE TABLE` is executed, all pages in that segment other than the index root page are also released.
- You can use the free page release utility (`pdreclaim` command) to release used pages. For details about releasing used free pages, see the *HiRDB Version 9 System Operation Guide*.

# 5 Database Access Using SQL

This chapter explains the use of the SQL database manipulation language to manipulate a database.

# 5.1 Use of SQL in HiRDB

This section provides an overview of using SQL for database access. For details about the SQL syntax, see the manual *HiRDB Version 9 SQL Reference*; For details about creating and designing a UAP, see the *HiRDB Version 9 UAP Development Guide*.

## 5.1.1 HiRDB SQL functions

Database manipulation language SQL statements are used to manipulate data in tables. The following functions are provided in the HiRDB SQL:

**Basic data manipulation**

- Data retrieval
- Data updating
- Data deletion
- Data insertion
- Search for specific data
- Data operations
- Data processing
- Manipulation of data in a table containing an abstract data type

**Reduction in number of steps required for UAP development and reduction of communication and analysis overhead**

- Stored procedures
- Stored functions

**Improvement in database access performance**

- Block transfer facility
- Rapid grouping facility
- FETCH facility using arrays
- Holdable cursor
- SQL optimization option

## 5.1.2 SQL execution methods

There are three methods of using SQL to access a HiRDB database:

- By coding SQL statements in a UAP and executing the executable-format file of the UAP (a UAP in which SQL statements are directly coded is called an *embedded SQL UAP*)
- By using the HiRDB SQL Executer to execute SQL statements interactively
- By using the *database definition utility* (`pddef`) to execute SQL statements

You can create a UAP to reference table data when complex operations or computational processing are required, or you can execute SQL statements interactively in order to perform simple data operations.

The following lists the high-level languages that can be used to code embedded UAPs:

- C
- C++
- COBOL
- OOCOBOL

# 5.2 Basic data manipulation

## 5.2.1 Cursor

Table retrieval results usually span multiple rows. A cursor is used by a UAP to indicate the position from which the newest result is to be extracted one row at a time out of retrieval results that span multiple rows. A cursor can be used for retrieving, updating, and deleting data.

`DECLARE CURSOR` is specified in order to use a cursor. The `OPEN` statement is used to open a cursor, and the `CLOSE` statement is used to close a cursor. The `FETCH` statement advances the position of the cursor.

## 5.2.2 Data retrieval

Data retrieval is the process of selecting the rows from a table that satisfy conditions that are specified with respect to a column. The data retrieval methods and an example of specifying SQL statements are shown as follows.

**Data retrieval methods**

The `SELECT` statement is used to retrieve data. The following three methods are available for data retrieval:

- Retrieval using a cursor
- Retrieval of multiple tables (specify the `FROM` clause in the `SELECT` statement)
- Retrieval by row (specify `ROW` in the selection expression of the `SELECT` statement)

**Data retrieval SQL specification example**

An example of retrieval using a cursor is explained as follows.

**Example**

1. Define cursor.

   In this example, a cursor named `CUR1` is used to retrieve from the stock table (`STOCK`) the product names (`PNAME`), colors (`COLOR`), and prices (`PRICE`) of products whose product name (`PNAME`) is `skirt`:

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

2. Open cursor.

   Cursor `CUR1` is opened:

   ```
   OPEN CUR1
   ```

3. Extract data.

   While cursor `CUR1` is open, it is advanced by one row and the contents of that row are stored in the UAP in specified areas (`:XPNAME`, `:XCOLOR`, `:XPRICE`):

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

4. Close cursor.

   Cursor `CUR1` is closed:

   ```
   CLOSE CUR1
   ```

By specifying `LIMIT` following the `ORDER BY` clause, you can retrieve search results from the first *n* rows. Specifying `LIMIT` might also improve SQL search performance. For details about retrieving search results from the first *n* rows, see the *HiRDB Version 9 UAP Development Guide*.

## 5.2.3 Data updating

Data updating is the process of changing information in a table. The data updating methods and an example of specifying SQL statements are shown as follows.

**Data updating methods**

The UPDATE statement is used to update data. The following three methods are available for data updating:

- Updating the row being pointed to by the cursor
- Updating only those rows that satisfy a condition (specify the WHERE clause in the UPDATE statement)
- Updating by row (specify ROW in the SET clause)

**Data updating SQL specification example**

An example of updating only those rows that satisfy a condition is explained as follows.

**Example**

In this example, the UPDATE statement updates to 20 the stock quantity (SQUANTITY) of products whose product code (PCODE) in the stock table (STOCK) is 411M:

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

## 5.2.4 Data deletion

Data deletion is the process of removing either the rows from a table that satisfy conditions that are specified with respect to a column or all rows that constitute the table. The data deletion methods and an example of specifying SQL statements are shown as follows.

**Data deletion methods**

The DELETE or PURGE TABLE statement is used to delete data. The following three methods are available for deleting data:

- Deleting the row being pointed to by the cursor
- Deleting only those rows that satisfy a condition (specify the WHERE clause in the DELETE statement)
- Deleting all rows (PURGE TABLE statement)

**Data deletion SQL specification example**

An example of deleting only those rows that satisfy a condition is explained as follows.

**Example**

In this example, the DELETE statement deletes from the stock table (STOCK) only the data whose product name (PNAME) is skirt:

```
DELETE FROM STOCK
  WHERE PNAME='skirt'
```

## 5.2.5 Data insertion

Data insertion is the process of inserting rows into a table. The data insertion methods and an example of specifying SQL statements are shown as follows.

**Data insertion methods**

The INSERT statement is used to insert rows. The following two methods are available for inserting rows into a table:

- Inserting rows by column
- Inserting rows by row (specify ROW in the INSERT statement)

**Data insertion SQL specification example**

An example of inserting rows by column is explained as follows.

**Example**

In this example, the INSERT statement inserts in each column of the stock table (STOCK) the values set in embedded variables (:ZPCODE, :ZPNAME, :ZCOLOR, :ZPRICE, and :ZSQUANTITY) that are used for transferring values between a table and UAPs:

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

## 5.2.6 Searching for specific data

A search condition is specified to manipulate specific data in a table. A search condition is a condition for selecting rows. For example, a search condition can specify a specific range of data or all data that is not the null value. It is also possible to use Boolean operators to combine multiple conditions. The methods of searching for specific data and SQL specification examples follow.

### (1) Methods of searching for specific data

The following methods can be used to search for data in a table:

- Searching for data within a specific range
- Searching for a specific character string
- Searching for data that is not the null value
- Searching for data that satisfies multiple conditions
- Search using a subquery

### (2) SQL examples for searching for specific data

#### (a) SQL specification example for searching for data within a specific range

The following three methods are available for searching for data within a specific range:

- Comparison predicate (used for equivalence and size comparison)
- `BETWEEN` predicate (used for extracting data within a specific range)
- `IN` predicate (used for extracting only the data that matches multiple values that are specified)

An example of using a comparison predicate for a data search is explained as follows.

**Example**

In this example, the `SELECT` statement searches the stock table (`STOCK`) for the product codes (`PCODE`) and product names (`PNAME`) of products whose stock quantity (`SQUANTITY`) is `50` or less:

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

#### (b) SQL specification example for searching for a specific character string

An example of conducting a search for rows in which there is a column that contains a specific character string is explained as follows.

**Example**

The `LIKE` predicate is used in this example. The `SELECT` statement searches the stock table (`STOCK`) for the product names (`PNAME`) and stock quantities (`SQUANTITY`) of products whose product code (`PCODE`) has `L` as its second character:

```
SELECT PNAME,SQUANTITY FROM STOCK
  WHERE PCODE LIKE '_L%'
```

#### (c) SQL specification example for searching for data that is not the null value

An example of conducting a search for rows in which a specified column does not contain the null value is explained as follows.

**Example**

NOT of the NULL predicate is used in combination in this example. The SELECT statement searches the stock table (STOCK) for the product codes (PCODE) of products whose product name (PNAME) is not empty (not the null value):

```
SELECT PCODE FROM STOCK
  WHERE PNAME IS NOT NULL
```

### (d) SQL specification example for searching for data that satisfies multiple conditions

An example of conducting a search for rows that contain data that satisfies a combination of multiple conditions is explained as follows.

**Example**

Boolean operators (AND, OR, and NOT) are used in this example. The SELECT statement searches the stock table (STOCK) for the product codes (PCODE) of products whose product name (PNAME) is blouse or polo shirt and whose stock quantity (SQUANTITY) is 50 or greater:

```
SELECT PCODE, SQUANTITY FROM STOCK
  WHERE (PNAME='blouse'
        OR PNAME='polo shirt')
        AND SQUANTITY=>50
```

### (e) SQL specification example for a search using a subquery

You can code a complex query by specifying a retrieval result of a search as a condition in the SELECT statement. This is called a *subquery*. The following two subquery methods are available:

- Quantified predicate
  A quantified predicate is used to narrow the results of a subquery by determining whether the results of the subquery satisfy a specified comparison condition.
- EXISTS predicate
  The EXISTS predicate is used to determine whether the results of a subquery constitute an empty set.

An example of conducting a search that uses a subquery that contains a quantified predicate is explained as follows:

**Example**

In this example that uses a quantified predicate, the SELECT statement searches the stock table (STOCK) for the product codes (PCODE) and product names (PNAME) of products that have a greater stock quantity value than the stock quantity (SQUANTITY) value for blouse:

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

## 5.2.7 Data operations

It is possible to retrieve numerical values or dates from a table's columns, to perform operations on such values, and to extract the results. The data operation methods and an SQL specification example follow.

**Data operation methods**

The following methods can be used to perform data operations:

- Concatenation operations on character string data
- Arithmetic operations on numeric data
- Operations on date and time data
- Operations using scalar functions
- Specification of conditional values using the CASE expression
- Explicit type conversion using a CAST specification

**Data operation SQL specification example**

An example of performing an arithmetic operation on numeric data is explained as follows.

**Example**

In this example, the SELECT statement uses the price (PRICE) and stock quantity (SQUANTITY) to compute the projected revenue for products with sweat pants as the product name (PNAME) and extracts the product codes (PCODE) and the computation results (in units of $10):

```
SELECT PCODE,PRICE*SQUANTITY/10 FROM STOCK
   WHERE PNAME='sweat pants'
```

## 5.2.8 Data processing

When data is extracted from a table, it is possible to process that data by grouping it or by sorting it in ascending or descending order. The data processing methods and an SQL specification example follow.

**Data processing methods**

The following operations can be used to process data in a table:

- Data grouping (GROUP BY clause specification and the set function)
- Sorting data in ascending or descending order (ORDER BY clause specification)
- Excluding duplicate data (DISTINCT specification)
- Set operations between sets of rows (derived tables) (UNION or EXCEPT specification)

**Data processing SQL specification example**

An example of rearranging (sorting) data into ascending order is explained as follows.

**Example**

In this example, the SELECT statement retrieves from the stock table (STOCK) the product codes (PCODE) and stock quantities (SQUANTITY) of products and sorts so that the retrieved data is in ascending order of the product codes (PCODE):

```
SELECT PCODE,SQUANTITY FROM STOCK
  ORDER BY PCODE
```

## 5.2.9 Manipulation of data in a table containing an abstract data type

This section describes how to manipulate data in a table containing abstract data types.

### (1) Abstract data type provided by a plug-in

When you use a plug-in, you can create a UAP by specifying the facility provided by the plug-in. This makes it possible to manipulate multimedia data such as documents and spatial data.

The following examples use the HiRDB Text Search Plug-in.

#### (a) Data retrieval

An example of using a Boolean predicate for data retrieval is explained as follows.

**Example**

In this example, the SELECT statement retrieves medicine IDs that contain the keyword headache in the efficacy section of the operation manual column of the MEDICINE_MANAGEMENT_TABLE (the SQL uses the facility contains in order to extract those documents that match the text search condition provided by the plug-in):

```
SELECT medicine-ID FROM MEDICINE_MANAGEMENT_TABLE
  WHERE contains(operation-manual,'attached-document-
data[efficacy{"headache"}]') IS TRUE
```

#### (b) Data updating

An example of data updating is explained as follows.

**Example**

In this example, the UPDATE statement updates the data in the operation manual for those columns in the MEDICINE_MANAGEMENT_TABLE that have medicine 2 as the medicine ID (the SQL uses the SGMLTEXT facility provided by the plug-in):

```
UPDATE MEDICINE_MANAGEMENT_TABLE SET operation-manual = SGMLTEXT(:sgml)
  WHERE medicine-ID = 'medicine-2'
```

The following sgml BLOB type embedded variable must be defined before the UPDATE statement:

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

**Explanation**

1. Defines the sgml BLOB type embedded variable.
2. Stores the new data for updating in the sgml embedded variable.
3. Sets the attribute value sgml_length of the created BLOB data to the length of the stored data.

### (c) Data deletion

An example of data deletion is explained as follows:

**Example**

In this example, the DELETE statement deletes from the MEDICINE_MANAGEMENT_TABLE those rows that have medicine 2 in the medicine ID column:

```
DELETE FROM MEDICINE_MANAGEMENT_TABLE
  WHERE medicine-ID = 'medicine-2'
```

### (d) Data insertion

An example of data insertion is explained as follows.

**Example**

In this example, the INSERT statement inserts into the MEDICINE_MANAGEMENT_TABLE rows that have medicine 25 in the medicine ID column (the SGMLTEXT facility provided by a plug-in is used in the SQL):

```
INSERT INTO MEDICINE_MANAGEMENT_TABLE (medicine-ID,operation-manual)
  VALUES('medicine-25',SGMLTEXT(:sgml))
```

The following sgml BLOB type embedded variable must be defined before the UPDATE statement:

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

**Explanation**

1. Defines the sgml BLOB type embedded variable.
2. Stores the new data for updating in the sgml embedded variable.
3. Sets the attribute value sgml_length of the created BLOB data to the length of the stored data.

## (2) Abstract data type provided by HiRDB XML Extension

Creating a UAP by specifying the functions provided by HiRDB XML Extension allows you to manipulate XML data.

### (a) Data retrieval

Examples of XML data retrieval follow.

**Example 1**

In this example, book information containing the book ID `126513592` is extracted from the book control table as a `VARCHAR` type value. The SQL statement can be written as follows:

```
SELECT book-ID, XMLSERIALIZE(BOOK INFORMATION AS VARCHAR(32000))
    FROM Book Control Table
    WHERE book-ID = 126513592
```

**Example 2**

In this example, the titles of books that fall in the `database` category are extracted from the book control table. To extract evaluation results of an XQuery expression, the `XMLQUERY` function is used. Furthermore, in order not to output rows in which the evaluation result of the XQuery expression is an empty sequence, the `XMLEXISTS` predicate is used. The SQL statement can be written as follows:

```
  SELECT book-ID,
         XMLSERIALIZE(
           XMLQUERY('/Book Information/Title'
                     PASSING BY VALUE BOOK INFORMATION
                     RETURNING SEQUENCE EMPTY ON EMPTY)
           AS VARCHAR(32000))
    FROM Book Control Table
    WHERE XMLEXISTS('/Book Information[category="database"]'
                     PASSING BY VALUE BOOK Information)
```

**Example 3**

In this example, the titles of books that fall in the `database` category are combined and extracted from the book control table. To output the XML value from each row as a single XML type value, the `XMLAGG` set function is used. The SQL statement can be written as follows:

```
SELECT XMLSERIALIZE(
         XMLAGG(
           XMLQUERY('/Book Information/Title'
                      PASSING BY VALUEBOOK INFORMATION
                      RETURNING SEQUENCE EMPTY ON EMPTY)
         )
         AS VARCHAR(32000))
       FROM Book Control Table
       WHERE XMLEXISTS('/Book Information[category="database"]'
                       PASSING BY VALUE BOOK INFORMATION)
```

**Example 4**

In this example, information on books whose title is the *Complete SQL Guide* and the titles of books that fall in the same category are extracted from the book control table. In order to use the XML value from each row as a single XML type value and evaluate the XQuery expression on this value, the `XMLAGG` set function has been specified as the argument of the XML query of the `XMLQUERY` function. The SQL statement can be written as follows:

```
SELECT
  XMLSERIALIZE(
    XMLQUERY(
      '$BOOKS/Book information [category=$BOOKS/book
information[title="Complete SQL Guide"]/category]'
      PASSING BY VALUE XMLAGG(BOOK INFORMATION) AS BOOKS
      RETURNING SEQUENCE EMPTY ON EMPTY)
    AS VARCHAR(32000))
  FROM Book Control Table
```

## (3) User-defined abstract data type

A routine or a component specification is used to manipulate data of a table containing a user-defined abstract data type. A component specification is used to manipulate the attribute of a column that comprises an abstract data type. The following examples manipulate the data of a table containing a user-defined abstract data type.

### (a) Retrieving an abstract data type column

An example of retrieving a column of a table containing a user-defined abstract data type is explained as follows.

**Example**

In this example, the `SELECT` statement uses the user-defined facility `service-years` to retrieve from `STAFF_TABLE` the employee numbers of employees whose service years are `20` years or more:

```
SELECT employee-no
  FROM STAFF_TABLE
  WHERE service-years (employee)>=20
```

### (b) Updating an abstract data type column

An example of updating a column of a table containing a user-defined abstract data type is explained as follows.

**Example**

In this example, the `UPDATE` statement updates to `CHIEF` the `position` employee attribute of the employee whose `EMPLOYEE_NO` column in `STAFF_TABLE` is `9001230`; this `UPDATE` statement uses the `employee..position` component specification for this updating:

```
UPDATE STAFF_TABLE
SET employee.position ='CHIEF'
WHERE employee_no = '900123'
```

### (c) Deleting an abstract data type column

An example of deleting a column of a table containing a user-defined abstract data type is explained as follows:

**Example**

In this example, the `DELETE` statement uses the `employee..position` component specification to delete the data in which the `position` attribute of the `employee` column in `STAFF_TABLE` is `GENERAL`:

```
DELETE FROM STAFF_TABLE
  WHERE employee..position='general'
```

### (d) Data insertion

An example of insertion of data into a table containing a user-defined abstract data type is explained as follows.

**Example**

In this example, the `INSERT` statement uses the `t_employee` constructor facility to insert a row whose `EMPLOYEE_NO` column is `990070` into `STAFF_TABLE` (`:xidphoto` is a BLOB type embedded variable in which the ID_photo image has been set):

```
INSERT INTO STAFF_TABLE
  VALUES ('990070',t_employee('Mary Moore',
                              'F',
                              'GENERAL',
                              '1999-04-01',
                              :xidphoto AS BLOB,
                              140000
                              ))
```

# 5.3 Stored procedures and stored functions

When a set of operations on a database is defined as a *procedure*, it is called a *stored procedure*; when a set of operations on a database is defined as a *function*, it is called a *stored function*.

Defining a stored procedure or a stored function generates an SQL object that codes an access procedure. The resulting stored procedure or stored function, together with its definition information, is stored in the database. Processing instructions for stored procedures or stored functions can be coded in SQL, Java, or C. Instructions coded in SQL are called an *SQL stored procedure* or an *SQL stored function*; instructions coded in Java are called a *Java stored procedure* or a *Java stored function*; and instructions coded in C are called a *C stored procedure* or a *C stored function*.

Stored procedures and stored functions are referred to collectively as *stored routines*. Furthermore, a stored routine whose owner is defined as `PUBLIC`, indicating all users, is called a *public routine*. Once a stored routine is defined as a public routine, there is no need to specify the owner's authorization identifier for a routine defined by another user to be invoked from a UAP (only the routine identifier need be specified). You can define a public routine by using the `CREATE PUBLIC PROCEDURE` or `CREATE PUBLIC FUNCTION` statement.

For details about Java stored procedures and Java stored functions, see Section *5.4 Java stored procedures and Java stored functions*. For details about C stored procedures and C stored functions, see *5.5 C stored procedures and C stored functions*.

A stored procedure might or might not have input, output, or input/output parameters; it is called by an SQL `CALL` statement. A stored function might or might not have input parameters; it is able to return a return value, and thus can be called as a value expression in an SQL. Note, however, that a stored function can be used only for processing data; it cannot be used for accessing tables in the database.

## (1) Application of a stored procedure to a job

Explained as follows are the types of jobs for which a stored procedure might be useful. For example, a product management job might involve the processing described as follows for analyzing the sales status of a product.

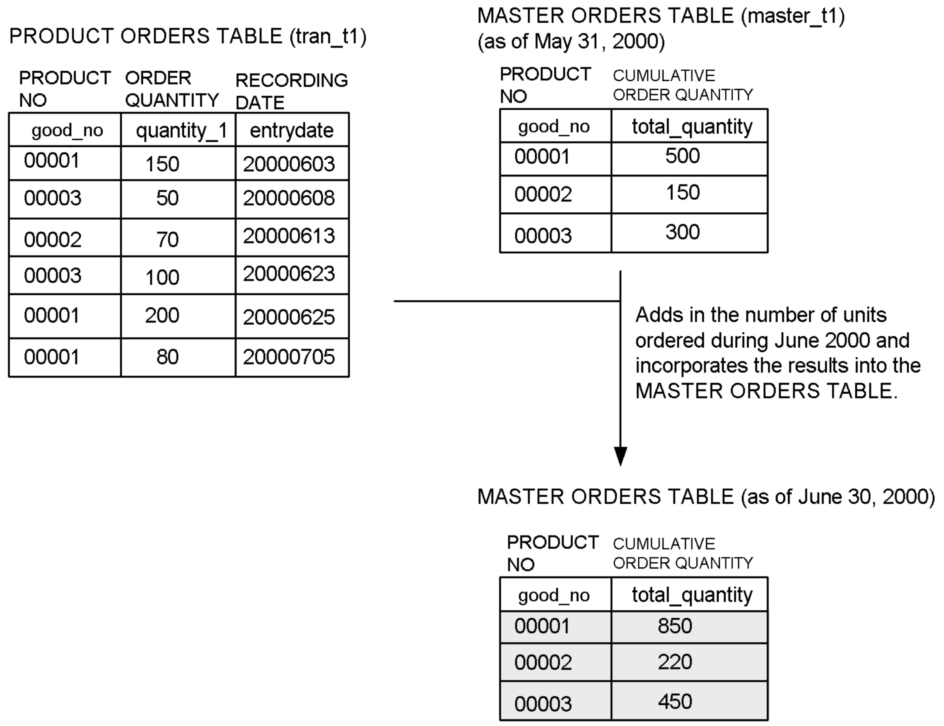
- For each product, the total number of units ordered each month is computed and the result is incorporated into a table that shows the cumulative number of units ordered since the product was introduced.

This is accomplished through the multiple database access processes described as follows.

1. Use a cursor to retrieve from the product orders table the product numbers, order quantities, and order recording dates for products for which orders were received during the month (`SELECT` statement).
2. Use a cursor to retrieve from the master order table the cumulative total number of units ordered for each product whose product number is contained in both the master orders table and the product orders table (`SELECT` statement).
3. For each applicable product, compute the sum of the total number of units ordered during the month and the cumulative total number of units ordered from the time the product was introduced through the end of the previous month, and update the master orders table with the result (`INSERT` and `UPDATE` statements).

When a series of database accessing processes such as the job shown here is registered at the database side rather than at the UAP side, the processes can be called for use from multiple UAPs. The following figure shows jobs to which a stored procedure can be applied.

Figure 5–1: Jobs to which a stored procedure can be applied



PRODUCT ORDERS TABLE (tran_t1)

| PRODUCT NO | ORDER QUANTITY | RECORDING DATE |
|---|---|---|
| good_no | quantity_1 | entrydate |
| 00001 | 150 | 20000603 |
| 00003 | 50 | 20000608 |
| 00002 | 70 | 20000613 |
| 00003 | 100 | 20000623 |
| 00001 | 200 | 20000625 |
| 00001 | 80 | 20000705 |

MASTER ORDERS TABLE (master_t1)
(as of May 31, 2000)

| PRODUCT NO | CUMULATIVE ORDER QUANTITY |
|---|---|
| good_no | total_quantity |
| 00001 | 500 |
| 00002 | 150 |
| 00003 | 300 |

Adds in the number of units ordered during June 2000 and incorporates the results into the MASTER ORDERS TABLE.

MASTER ORDERS TABLE (as of June 30, 2000)

| PRODUCT NO | CUMULATIVE ORDER QUANTITY |
|---|---|
| good_no | total_quantity |
| 00001 | 850 |
| 00002 | 220 |
| 00003 | 450 |

In the database access process shown in Figure 5-1, the total number of units ordered during June 2000 is determined for each product from the PRODUCT ORDERS TABLE, which has PRODUCT NO, ORDER QUANTITY, and RECORDING DATE as its columns. The results are incorporated into the MASTER ORDERS TABLE, which has PRODUCT NO and CUMULATIVE ORDER QUANTITY as its columns. In this case, the database access process described as follows can be defined as a stored procedure and registered in the database.

**Processing contents of stored procedure**

The number of units ordered for a specified period is determined, and the result is incorporated into the master orders table.

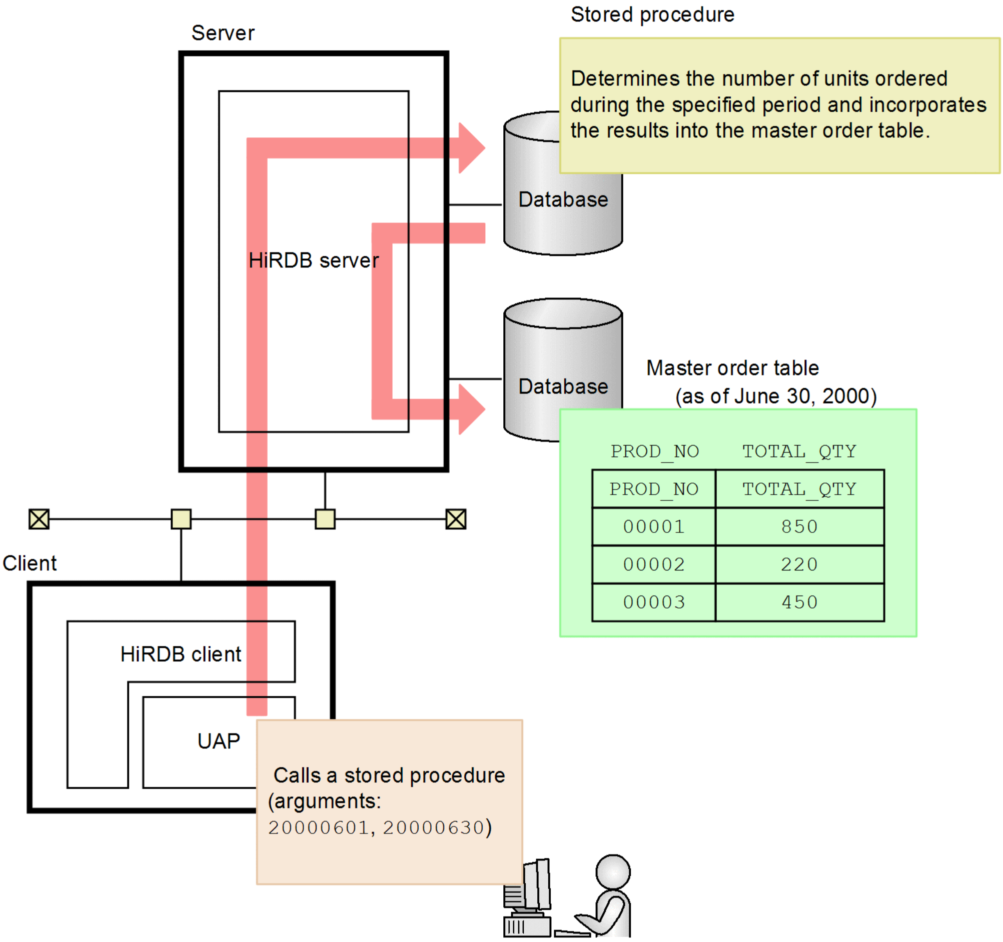
A UAP can perform the product management job shown in Figure 5-1 by simply calling the stored procedure with the argument described as follows specified.

**Argument specified by UAP**

Period for which number of units ordered is to be computed: (begin-date 20000601, end-date 20000630)

The following figure shows how a stored procedure is used.

Figure 5–2: Stored procedure usage



Determines the number of units ordered during the specified period and incorporates the results into the master order table.

PROD_NO TOTAL_QTY

| PROD_NO | TOTAL_QTY |
|---|---|
| 00001 | 850 |
| 00002 | 220 |
| 00003 | 450 |

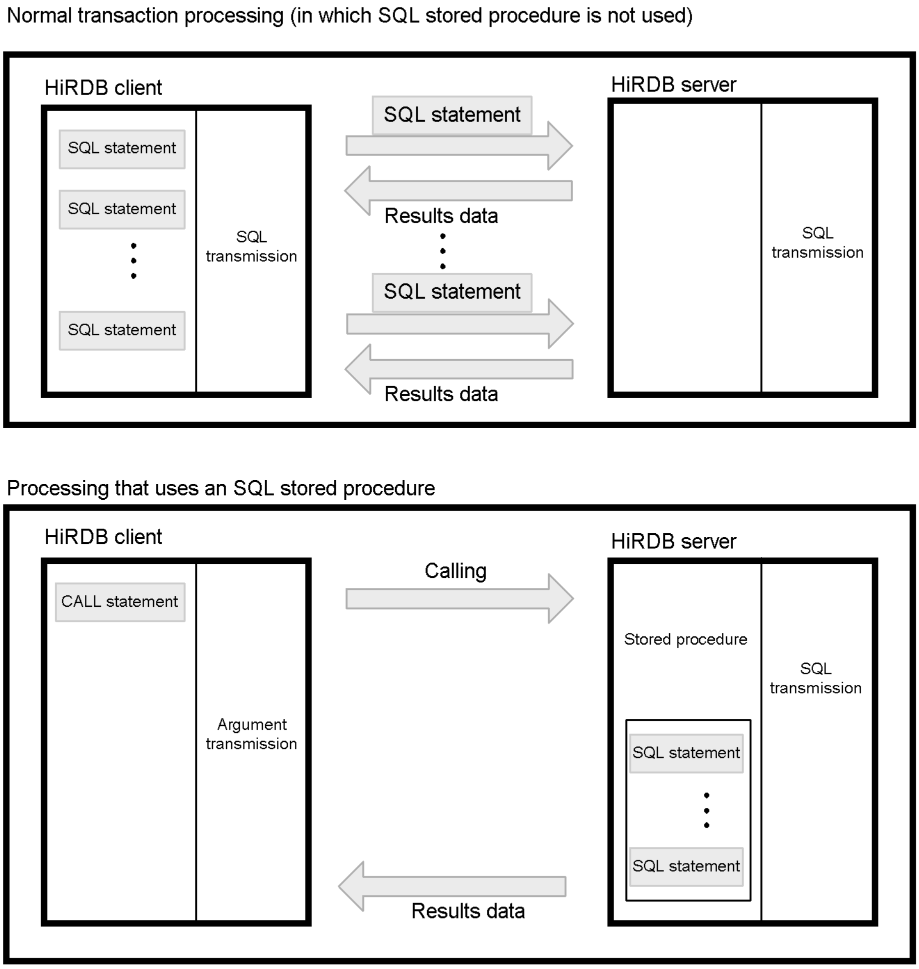
Calls a stored procedure (arguments: 20000601, 20000630)

This illustrates how, when a stored procedure is used, it is possible to register a database access process at the database side, thus turning the database access process into a component.

Even when the database access process is changed, all that has to be done is to change the stored procedure; there is no need to change the UAP, thus reducing the number of steps required for UAP development.

In order to execute multiple SQL statements, an application typically has to access the database as many times as there are SQL statements to be executed. By contrast, storing multiple SQL statements to be executed in the database as a stored procedure means that the multiple SQL statements can be executed with one call of the stored procedure and only one access to the database. This greatly reduces communication processing associated with the passing of data between the HiRDB server and HiRDB client applications, as well as the overhead associated with the parsing of SQL statements by the front-end server. The following figure shows the communications processing for an SQL stored procedure.

Figure 5–3: Communications processing for an SQL stored procedure



## (2) Application of a stored function

With a stored function, the user can define data processing in a database as an arbitrary function, using conditional branching (`IF` statement) and a routine control SQL such as SQL repetition (`WHILE` statement). Consequently, the user can turn data processing into a component. When a plug-in is used, the function provided by the plug-in is registered as a stored function in the database.

## (3) Creating RDAREAs for storing stored procedures and stored functions

To use stored procedures or stored functions, the following types of RDAREAs must be created:

- Data dictionary LOB RDAREAs
- Data dictionary RDAREAs

For details about creating RDAREAs for storing stored procedures and stored functions, see the *HiRDB Version 9 System Operation Guide*.

## (4) Creating a UAP for calling stored procedures and stored functions

Explained as follows are the methods of calling a stored procedure or a stored function. For details about creating a UAP for calling stored procedures and stored functions, see the *HiRDB Version 9 UAP Development Guide*.

**Calling a stored procedure**

You call a stored procedure by specifying, in the UAP, a `CALL` statement written in SQL.

**Invoking a stored function**

You call a stored function by specifying a *function call* as a value in an SQL statement. You can specify arguments when the function is called. The value will be returned in the `RETURN` statement of the SQL.

## (5) Overloading of stored functions

Multiple stored functions with the same name can be defined, as long as they have different numbers of parameters and their data types are different. Stored functions with the same name are *mutually overloaded*. Because the overload function can be used to assign the same name to multiple functions with different parameter data types, it is possible to standardize the names of functions that have the same function. When a stored function is invoked, those facilities described as follows become the candidates for execution from among the facilities with the specified name and the same number of parameters as the number of specified arguments:

- Facilities in which the data type of each argument exactly matches the data type of the corresponding parameter
- If the data type of each argument does not exactly match the data type of the corresponding parameter, arguments and parameter data types are compared sequentially from left to right, and the facility with a parameter with the highest priority among the lower-priority parameters becomes the candidate.

For details about the rules for selecting the function to be invoked, see the manual *HiRDB Version 9 SQL Reference*.

## (6) SQL procedures and definition of user-defined functions

Explained as follows are the SQL procedures and the definition of user-defined functions. For details about SQL procedures and the definition of user-defined functions, see the *HiRDB Version 9 UAP Development Guide*.

**Defining a procedure for creating a stored procedure**

To create a stored procedure, one of the following SQLs must be used first to define a *procedure*:

- `CREATE PROCEDURE`
- *Procedure body* specified by `CREATE TYPE` (for specifying a procedure for an abstract data type)

**Defining a user-defined function for creating a stored function**

To create a stored function, one of the following SQLs must be used first to define a *user-defined function*:

- `CREATE FUNCTION`
- *Function body* specified by `CREATE TYPE` (for specifying a function for an abstract data type)

Procedures, user-defined functions, and system-defined functions are referred to collectively as routines.

## (7) Registering stored procedures and stored functions into the database

When an SQL that defines a procedure or a user-defined function is executed by database definition (`pddef`), the procedure or user-defined function is compiled automatically, thus creating an SQL object. Moreover, the definition source of the SQL and the SQL object are stored in a data dictionary LOB RDAREA. This registers the stored procedure or stored function into the database. For details about registration of stored procedures and stored functions into the database, see the *HiRDB Version 9 System Operation Guide*.

## (8) Re-creating stored procedures and stored functions

If the definition of a table or index is changed or a stored procedure is deleted by the `DROP PROCEDURE` statement, you need to re-create the voided stored procedure by executing the `ALTER PROCEDURE` statement. Similarly, if a stored function is deleted by the `DROP FUNCTION` statement, you need to re-create the voided stored function by executing the `ALTER ROUTINE` statement. For details about re-creating a stored procedure or a stored function, see the *HiRDB Version 9 System Operation Guide*.

# 5.4 Java stored procedures and Java stored functions

Stored procedures and stored functions in which processing instructions are coded in Java are called *Java stored procedures* and *Java stored functions*. Java stored procedures and Java stored functions are referred to collectively as *external Java stored routines*. External Java stored routines are explained below.

## 5.4.1 Environments in which Java stored procedures and Java stored functions can be used

The following table lists the environments in which Java stored procedures and Java stored functions can be used (by OS used on the HiRDB server).

Table 5–1: Environments in which Java stored procedures and Java stored functions can be used

| Applicable OS on the HiRDB server | | Can it be used? | |
|---|---|---|---|
| | | When the Type2 JDBC driver is used | When the Type4 JDBC driver is used |
| HP-UX | | Y | Y |
| Solaris | | N | Y |
| AIX | | N | Y |
| Linux | | N | Y |
| Windows | 32-bit | Y | Y |
| | x64 | N | Y |
| | IPF | Y | Y |

Legend:

Y: Stored routines can be used.

N: Stored routines cannot be used.

## 5.4.2 Characteristics of an external Java stored routine

An external Java stored routine has the following characteristics:

1. Absence of communication overhead between server and client

   As in the case of SQL stored procedures and SQL stored functions, external Java stored routines are processed at the server, which eliminates communication overhead between server and client.

2. Ability to code the procedure or function itself in Java

   The Java coding language provides more sophisticated control than is possible with SQL.

3. Ability to operate between dissimilar DBMSs

   Because Java is a platform-independent language, programs coded in Java can operate on dissimilar DBMSs that support external Java stored routines.

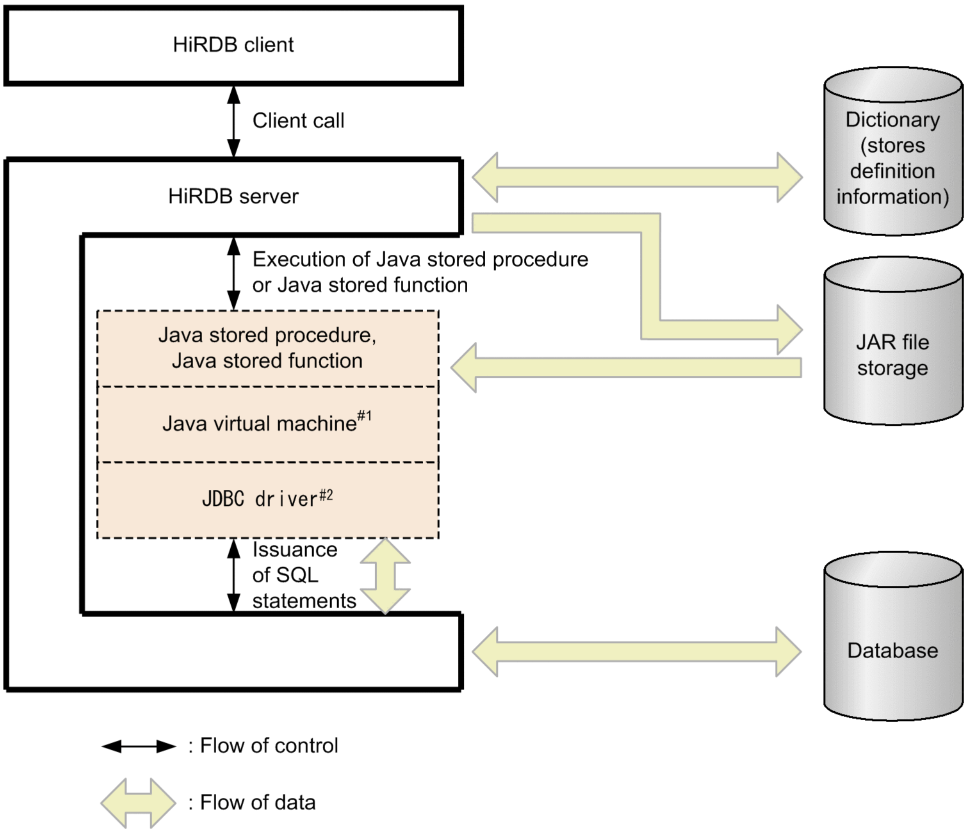
4. Simple debugging

   The debugging of an SQL stored procedure or an SQL stored function requires actual execution at the server of the procedure or function. By contrast, external Java stored routines can be debugged by providing a Java debugger at the client; moreover, this debugging process can address database access operations as well.

## 5.4.3 System configuration (position of a Java virtual machine)

The following figure shows the position of a Java virtual machine in a HiRDB system.

Figure 5–4: Position of Java virtual machine in a HiRDB system


HiRDB client
Client call
HiRDB server
Execution of Java stored procedure or Java stored function
Java stored procedure, Java stored function
Java virtual machine#1
JDBC driver#2
Issuance of SQL statements
Dictionary (stores definition information)
JAR file storage
Database
: Flow of control
: Flow of data

#1: The Java virtual machine is included in the JRE (Java Runtime Environment). You can find information about and obtain the JRE from the Web sites of various platform vendors.

#2: The JDBC driver is a standard interface provided with HiRDB.

For details about how to obtain the JRE and system configuration examples, see the *HiRDB Version 9 System Operation Guide*.

## 5.4.4 Executing external Java stored routines

Before you can execute Java stored routines, you must install the JDBC driver. To do so, select the JDBC driver when you perform a HiRDB client installation.

For details about setting an environment for using Java stored procedures or Java stored functions, see the *HiRDB Version 9 System Operation Guide*.

## 5.4.5 External Java stored routine creation and execution procedure

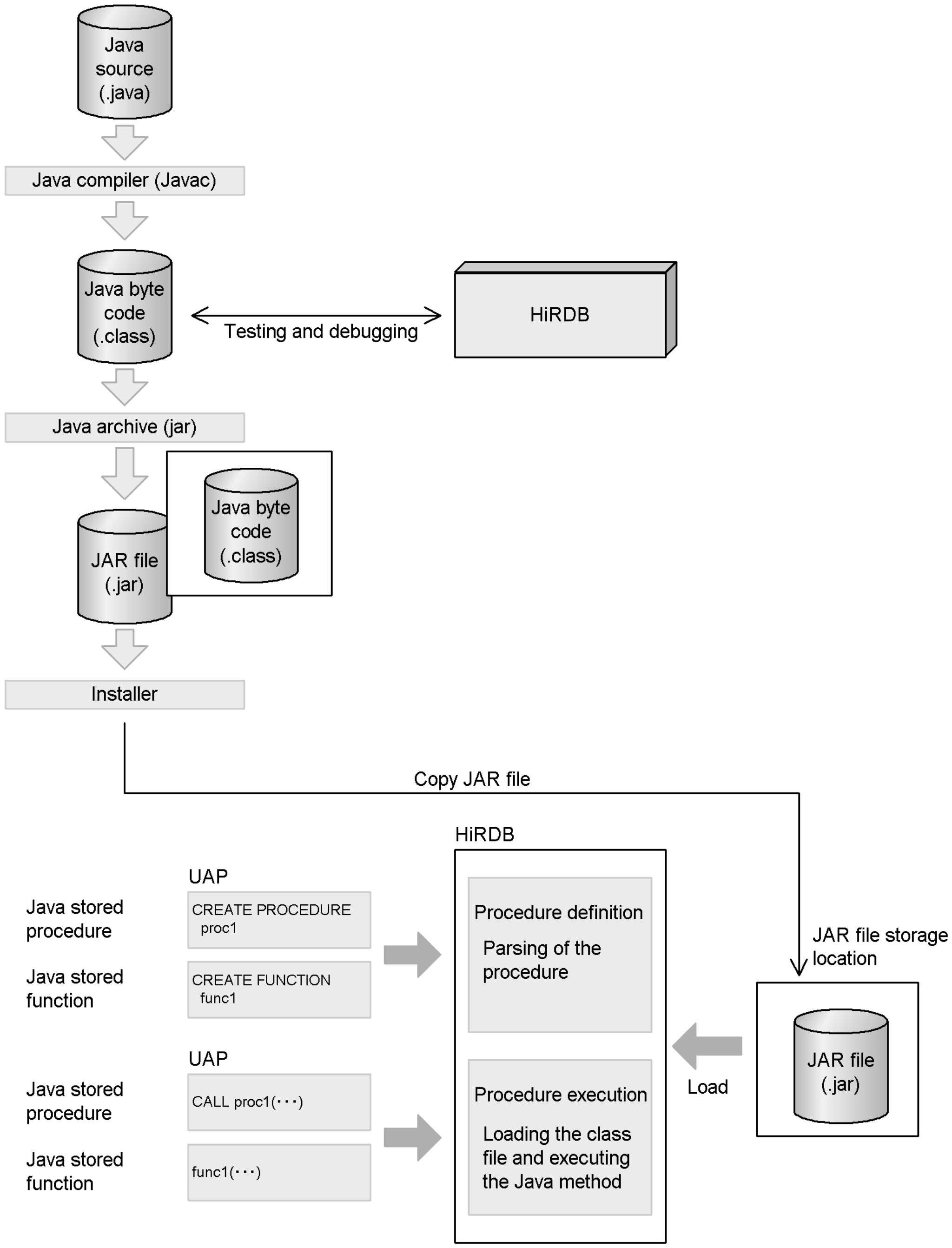
The following is the general procedure for creating and executing an external Java stored routine.

**Procedure**

1. Create an external Java stored routine.
2. Register a new JAR file.
3. Define the external Java stored routine.
4. Execute the external Java stored routine.

The following figure shows the flow of events from the creation of an external Java stored routine to its execution.

Figure 5–5: Flow from creation to execution of an external Java stored routine



## (1) Coding an external Java stored routine

In this step, you code either a procedure or a function in Java, then you compile the resulting Java program. Compilation produces a Class file. You can then test and debug the Class file, using the Java virtual machine at the client, and create a JAR file from the Class file.

## (2) Registering a new JAR file

In this step, you register the new JAR file in HiRDB.

**Registration by a HiRDB administrator:**

The HiRDB administrator uses the `pdjarsync` command.

**Registration by the UAP developer:**

The programmer uses the INSTALL JAR or REPLACE JAR statement in the embedded language. These SQL statements can be coded in either the pddef file or the UAP, and then executed.

## (3) Defining the external Java stored routine

You use the CREATE PROCEDURE or CREATE FUNCTION statement to define an external Java stored routine from the JAR file.

## (4) Executing the external Java stored routine

As in the case of executing a stored procedure or a stored function, you execute the SQL by specifying either a CALL statement or a function call. The CALL statement causes a Java method to be executed as a Java stored procedure. The function call causes a Java method to be executed as a Java stored function. The external Java stored routine is executed on the Java virtual machine at the server.

# 5.5 C stored procedures and C stored functions

Stored procedures and stored functions whose processing instructions are coded in C are called *C stored procedures* and *C stored functions*. C stored procedures and C stored functions are referred to collectively as *external C stored routines*. External C stored routines are explained below.

## 5.5.1 Characteristics of an external C stored routine

An external C stored routine has the following characteristics:

- Absence of communication overhead between server and client

  As in the case of SQL stored procedures and SQL stored functions, external C stored routines are processed at the server, which eliminates communication overhead between server and client.
- Ability to code the procedure or function itself in C

  Because the C language instructions can be directly coded, processing can be coded more flexibly than with SQL control statements.
- Simple debugging

  The debugging of an SQL stored procedure or an SQL stored function requires actual execution at the server of the procedure or function. By contrast, external C stored routines can be debugged by providing a C language debugger at the client.

## 5.5.2 External C stored routine creation and execution procedure

The following is the general procedure for creating and executing an external C stored routine:

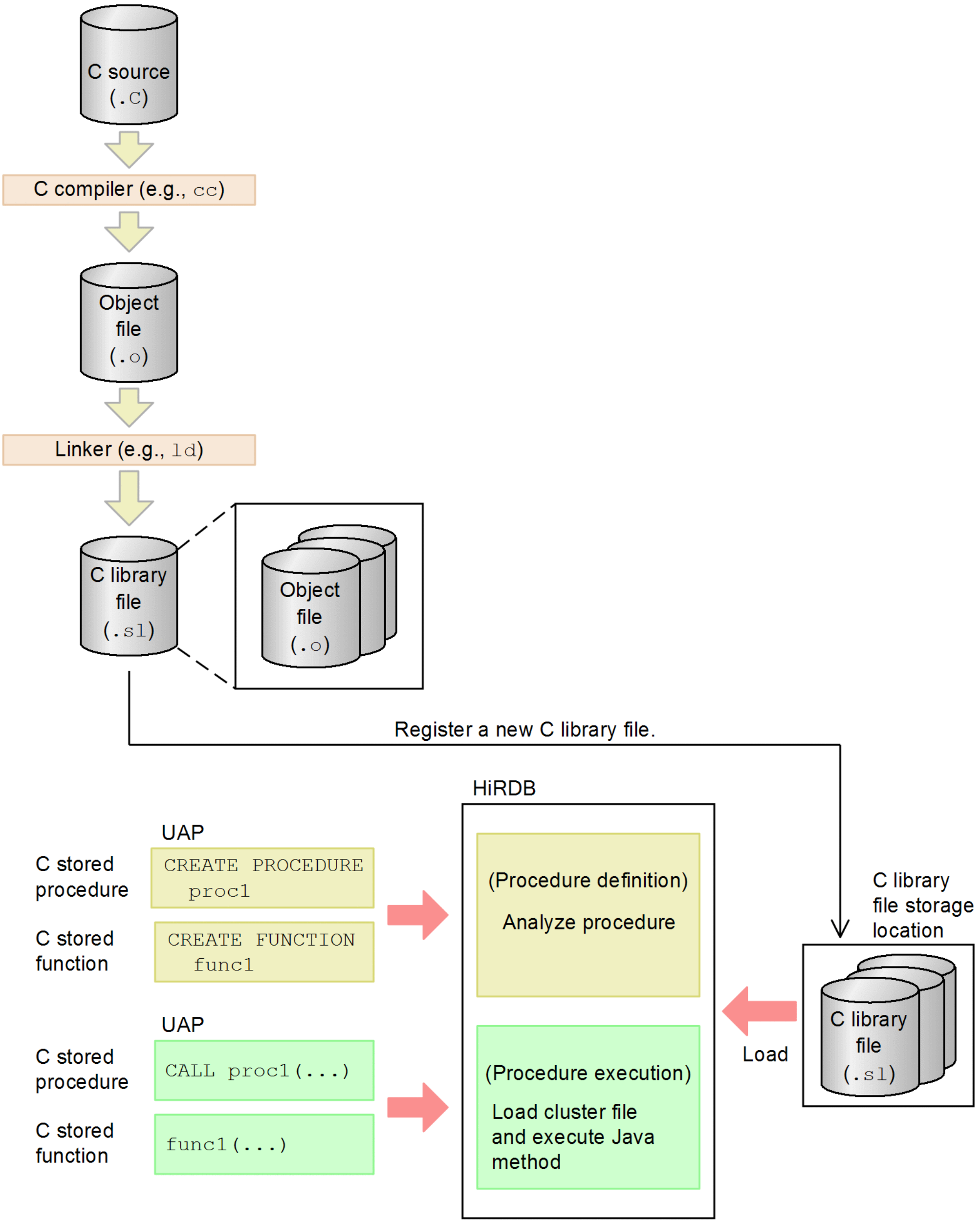
**General procedure**

1. Create an external C stored routine.
2. Register a new C library file.
3. Define the external C stored routine.
4. Execute the external C stored routine.

For details about executing external C stored routines, see the *HiRDB Version 9 UAP Development Guide*.

The following figure shows the flow of events from creation of an external C stored routine to its execution.

Figure 5–6: Flow from creation to execution of an external C stored routine



**Note**

The extension of the C library file varies according to the OS.

## (1) Coding an external C stored routine

In this step, you code either a procedure or a function in C, and then you compile the resulting C program. Compilation produces an object file. You then link multiple object files to create a C library file.

## (2) Registering a new C library file

In this step, you register the new C library file in HiRDB.

**Registration by a HiRDB administrator:**

The HiRDB administrator uses the pdclibsync command.

**Registration by the UAP developer:**

The programmer codes the `INSTALL CLIB` or `REPLACE CLIB` statement of the SQL in the UAP, and then executes it.

## (3) Defining the external C stored routine

You use the `CREATE PROCEDURE` or `CREATE FUNCTION` statement to define an external C stored routine.

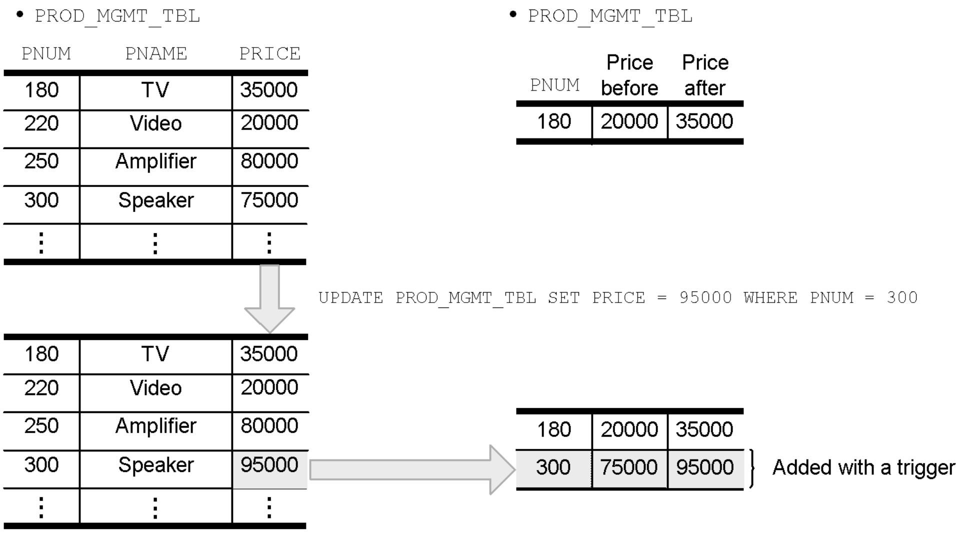
## (4) Executing the external C stored routine

As in the case of executing a stored procedure or a stored function, you execute the SQL by specifying either a `CALL` statement or a function call. The `CALL` statement causes a C function to be executed as a C stored procedure. The function call causes a C function to be executed as a C stored function.

# 5.6 Triggers

By defining a trigger, you can have SQL statements automatically execute when an operation (updating, insertion, deletion) is performed on a particular table. A trigger defines such specifications as the table to which the trigger applies, the SQL statement that activates the trigger (trigger event SQL), the SQL statements that are to be executed automatically (trigger SQL statements), and conditions for execution of the action (trigger action search conditions). When an SQL statement that satisfies the trigger action search conditions is executed on a table for which a trigger is defined, the trigger SQL statements are automatically executed. The following figure provides an overview of triggers.

Figure 5–7: Overview of triggers



- PROD_MGMT_TBL

| PNUM | PNAME | PRICE |
|---|---|---|
| 180 | TV | 35000 |
| 220 | Video | 20000 |
| 250 | Amplifier | 80000 |
| 300 | Speaker | 75000 |
| ⋮ | ⋮ | ⋮ |

- PROD_MGMT_TBL

| PNUM | Price before | Price after |
|---|---|---|
| 180 | 20000 | 35000 |

```
UPDATE PROD_MGMT_TBL SET PRICE = 95000 WHERE PNUM = 300
```

| | | |
|---|---|---|
| 180 | TV | 35000 |
| 220 | Video | 20000 |
| 250 | Amplifier | 80000 |
| 300 | Speaker | 95000 |
| ⋮ | ⋮ | ⋮ |

| | | |
|---|---|---|
| 180 | 20000 | 35000 |
| 300 | 75000 | 95000 |

**Explanation**

A trigger is defined to provide cumulative updating of data in the product management historical table on price changes that occur in the product management table. In this example, when the trigger event SQL statement (`UPDATE` in this case) is executed from the UAP, the product number, the price before the change, and the price after the change in the row containing the changed price are added to the product management historical table.

Note, however, that when you define a trigger for a table, you must re-generate any function, procedure, or trigger SQL object that uses that table, because any SQL object that uses that table becomes invalid. Similarly, if you define, change, or delete any resources that are used by the trigger (tables, indexes, and so on), the trigger SQL object becomes invalid, so you must re-generate it as well. For details, see the *HiRDB Version 9 Installation and Design Guide*.

## (1) Application criteria

Consider using a trigger to perform the following actions when an associated UAP operation occurs:

- To always update one table whenever a particular table is updated
- To always update particular columns associated with the updated row whenever a particular table is updated (relating columns)

## (2) Before you define a trigger

When you define a trigger, an SQL object that codes the specified trigger action procedure is automatically generated and stored in the corresponding data dictionary LOB RDAREA. Thus, before you define a trigger, you must make sure that the data dictionary LOB RDAREA has sufficient capacity. For details about how to estimate the capacity of data dictionary LOB RDAREAs, see the *HiRDB Version 9 Installation and Design Guide*.

In addition, if you plan to execute trigger event SQL statements, you must also take into consideration the trigger SQL object size when you specify the buffer length for SQL objects. For details about how to estimate the buffer length for SQL objects, see the manual *HiRDB Version 9 System Definition*.

### (3) Trigger definition statements

The following definition SQL statements are used to define a trigger, to re-generate an SQL object, and to delete a trigger.

`CREATE TRIGGER`

This statement defines a trigger. You can define triggers only for tables that you own. You cannot define triggers for tables owned by other users.

`ALTER TRIGGER`

This statement re-generates the SQL object of a trigger that has already been defined. You can also use the `ALTER ROUTINE` statement to re-generate SQL objects.

`DROP TRIGGER`

This statement deletes a trigger.

# 5.7 Integrity constraints

A constraint that guarantees that data in a database is valid is called an integrity constraint. Integrity constraints are set up in order to prevent accesses to the database from causing the data to lose its integrity. HiRDB supports two integrity constraints:

- NOT NULL constraint
- Uniqueness constraint

## 5.7.1 NOT NULL constraint

You can define the *NOT NULL constraint* for a column, and thereby prohibit the null value as data in the column. The NOT NULL constraint is specified on a column-by-column basis. When the NOT NULL constraint is defined for a column, a row containing the null value in that column cannot be added, nor can a row be updated so as to set the null value in that column. A column for which the NOT NULL constraint is defined must have a definite value in every row. An attempt to set the null value in such a column results in a constraint violation.

The NOT NULL constraint is set by specifying the `NOT NULL` option in the `CREATE TABLE` statement. For details about the NOT NULL constraint, see the *HiRDB Version 9 UAP Development Guide*.

**Comment**

The NOT NULL constraint is applied to columns for which the primary key is defined.

## 5.7.2 Uniqueness constraint

You can define the *uniqueness constraint* for a column, and thereby prohibit duplication of data in the column (each data entry in the column must be unique). When the uniqueness constraint is defined for a column, a row cannot be added if it contains a nonunique value in that column, nor can a row be updated so as to set a nonunique value in that column. Because columns for which the uniqueness constraint is defined are always required to have unique values, an attempt to assign a non-unique value to such a column results in a constraint violation. The uniqueness constraint can be specified for the following columns:

- Columns for which a cluster key is defined
- Columns for which an index key is defined

The uniqueness constraint is set for a column for which a cluster key is defined by specifying the `UNIQUE CLUSTER KEY` option in the `CREATE TABLE` statement. Similarly, the uniqueness constraint is set for a column for which an index key is defined by specifying the `UNIQUE` option in the `CREATE INDEX` statement. For details about the uniqueness constraint, see the *HiRDB Version 9 UAP Development Guide*.

**Comment**

The uniqueness constraint is applied to columns for which the primary key is defined.

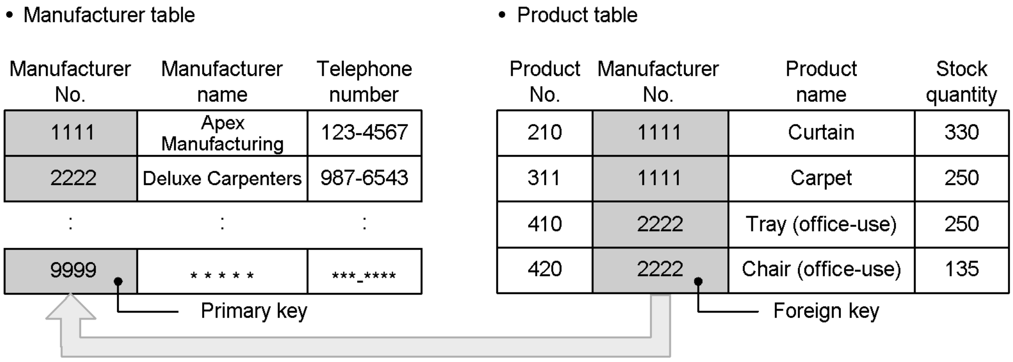
# 5.8 Referential constraints

Tables in a database are normally not independent entities, because typically there are links between tables. The data in a table that is not linked is probably of no value to other tables. To maintain referential integrity of data that is linked across tables, the concept of *referential constraints* is introduced. A referential constraint is defined for a specific column (called a *foreign key*) when a table is defined. A table in which a referential constraint and a foreign key are defined is called a *referencing table*, while a table that is referenced from a referencing table with a foreign key is called a *referenced table*. In a referenced table, a primary key that is referenced by the foreign key must be pre-defined.

Note that execution of SQL code or a utility might cause a loss of referential integrity between a referenced table and a referencing table. When this occurs, the referencing table is placed in check pending status. For details about check pending status, see *5.10 Check pending status*.

The following figure shows an example of a referenced table and a referencing table. In this example, the product table is the referencing table, and the manufacturer table is the referenced table. The foreign key in the referencing table references the primary key to determine the name of the manufacturer.

Figure 5–8: Example of referenced table and referencing table



- Manufacturer table

| Manufacturer No. | Manufacturer name | Telephone number |
|---|---|---|
| 1111 | Apex Manufacturing | 123-4567 |
| 2222 | Deluxe Carpenters | 987-6543 |
| : | : | : |
| 9999 | * * * * * | ***-**** |

- Product table

| Product No. | Manufacturer No. | Product name | Stock quantity |
|---|---|---|---|
| 210 | 1111 | Curtain | 330 |
| 311 | 1111 | Carpet | 250 |
| 410 | 2222 | Tray (office-use) | 250 |
| 420 | 2222 | Chair (office-use) | 135 |

For details about referential constraints, see the *HiRDB Version 9 Installation and Design Guide*.

**Advantages of referential constraints**

When you define a referential constraint, the workload associated with UAP creation can be reduced because checking of data integrity on tables and data manipulation can be automated. The downside is that updating referenced tables and referencing tables takes longer because of the processing time required to check the integrity of the data.

## (1) Defining a referential constraint

To enable a referential constraint, you must first define in the referenced table the primary key that is referenced by the foreign key. To do so, use the `CREATE TABLE` definition SQL statement to specify `PRIMARY KEY` in the referenced table. To apply check pending status, specify `USE` in the `pd_check_pending` operand or omit specification of the `pd_check_pending` operand.

To define a referential constraint, you first specify `FOREIGN KEY` in the referencing table. In the `FOREIGN KEY` clause, you specify the following:

- Column to be referenced
- Referenced table
- Referential constraint action

  A referential constraint action is an insertion, update, or deletion action in the referenced table; it is specified with `CASCADE` or `RESTRICT`.

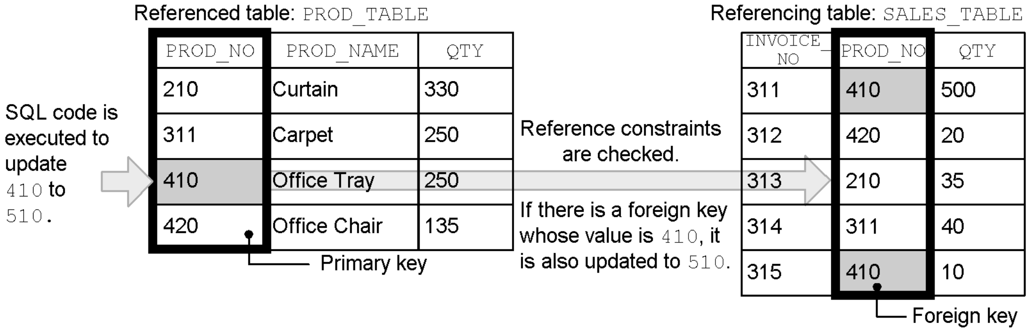
The following subsections explain the actions in the referenced table and the referencing table when `CASCADE` and `RESTRICT` are specified.

### (a) When CASCADE is specified

When `CASCADE` is specified and a primary key of a referenced table is changed, the foreign key is also changed. When a foreign key of a referencing table is changed, a check is performed to determine if there is a row containing a primary key whose value is the same as the value of the foreign key after the change; the foreign key is not changed if such a change would result in a referential constraint violation.

The figures below show examples of the actions that occur when `CASCADE` is specified when SQL code is executed for a referenced table and a referencing table.

Figure 5–9: Example of the action when update SQL code is executed for a referenced table (with CASCADE specified)



Referenced table: `PROD_TABLE`

| PROD_NO | PROD_NAME | QTY |
|---|---|---|
| 210 | Curtain | 330 |
| 311 | Carpet | 250 |
| 410 | Office Tray | 250 |
| 420 | Office Chair | 135 |

Referencing table: `SALES_TABLE`

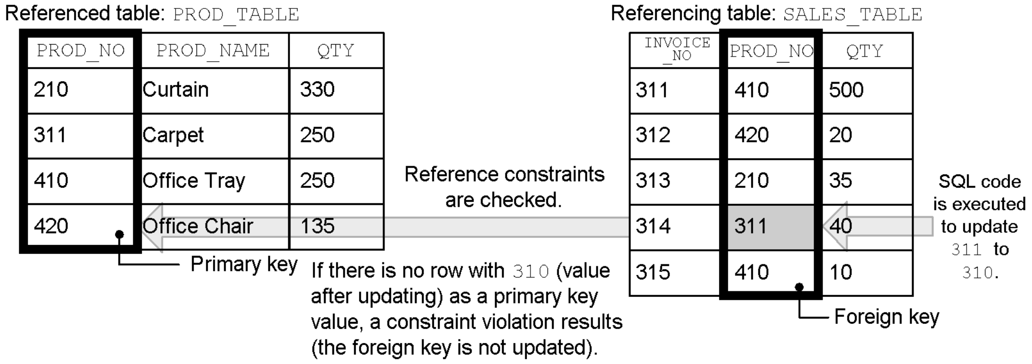
| INVOICE_NO | PROD_NO | QTY |
|---|---|---|
| 311 | 410 | 500 |
| 312 | 420 | 20 |
| 313 | 210 | 35 |
| 314 | 311 | 40 |
| 315 | 410 | 10 |

SQL code is executed to update 410 to 510.
Primary key
Reference constraints are checked.
If there is a foreign key whose value is 410, it is also updated to 510.
Foreign key

**Explanation**

If a row contains a foreign key with the same value as the primary key, then in order to maintain constraints the foreign key is changed in the same way that the primary key is changed. In this case, updating is performed on the referenced table; insertion and deletion would be handled in the same manner.

Figure 5–10: Example of the action when update SQL code is executed for a referencing table (with CASCADE specified)



Referenced table: `PROD_TABLE`

| PROD_NO | PROD_NAME | QTY |
|---|---|---|
| 210 | Curtain | 330 |
| 311 | Carpet | 250 |
| 410 | Office Tray | 250 |
| 420 | Office Chair | 135 |

Referencing table: `SALES_TABLE`

| INVOICE_NO | PROD_NO | QTY |
|---|---|---|
| 311 | 410 | 500 |
| 312 | 420 | 20 |
| 313 | 210 | 35 |
| 314 | 311 | 40 |
| 315 | 410 | 10 |

Primary key
Reference constraints are checked.
If there is no row with 310 (value after updating) as a primary key value, a constraint violation results (the foreign key is not updated).
SQL code is executed to update 311 to 310.
Foreign key

**Explanation**

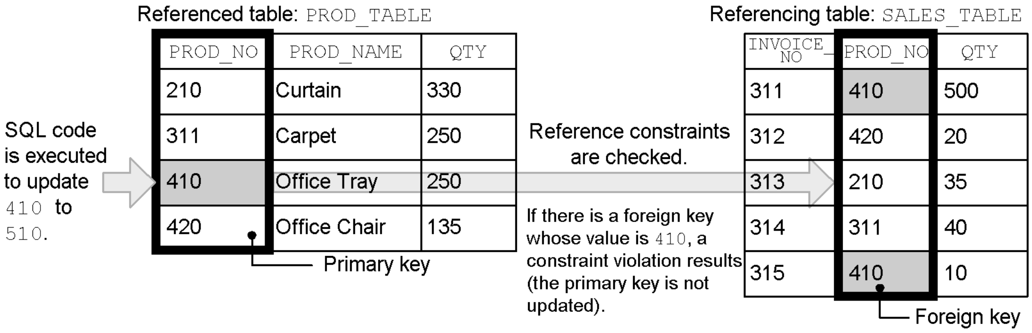
If a row contains a primary key with the same value as that of the foreign key after updating, the update is executed on the foreign key. The update is also executed if the foreign key contains a null value, even if there is no row containing a primary key with the same value. If there is no null value, a referential constraint violation results. If this occurs, there is no effect on the referenced table. Insertion and deletion would be handled in the same manner.

### (b) When RESTRICT is specified

When `RESTRICT` is specified, then when a primary key of a referenced table is changed, a referential constraint violation occurs if there is a row that contains a foreign key with the same value. If the foreign key can be changed, a check is performed to determine if a row contains a primary key with the same value and, if a referential constraint violation would result, the foreign key is not changed.

The following figure shows the actions that occur when `RESTRICT` is specified when SQL code is executed for a referenced table. The action that occurs in a referencing table is the same as when `CASCADE` is specified (see *Figure 5-10*).

Figure 5–11: 11Example of the action when update SQL code is executed for a referenced table (with RESTRICT specified)


Referenced table: PROD_TABLE
Referencing table: SALES_TABLE
PROD_NO
PROD_NAME
QTY
210
Curtain
330
311
Carpet
250
410
Office Tray
250
420
Office Chair
135
Primary key
SQL code is executed to update 410 to 510.
Reference constraints are checked.
If there is a foreign key whose value is 410, a constraint violation results (the primary key is not updated).
INVOICE_NO
PROD_NO
QTY
311
410
500
312
420
20
313
210
35
314
311
40
315
410
10
Foreign key

## (2) Data manipulation and integrity

When you use a data manipulation SQL (other than PURGE TABLE) to manipulate data in a referenced table or referencing table, HiRDB performs checking when the SQL code is executed in order to guarantee data integrity. However, if you perform the data operations shown below, HiRDB might no longer be able to guarantee data integrity.

- Loading data with the database load utility (pdload)
- Reorganization or reloading with the database reorganization utility (pdrorg)
- Re-initialization of an RDAREA (initialize rdarea) with the database structure modification utility (pdmod)
- Updating with updatable online reorganization (pdorend) (UNIX edition only)
- Execution of the PURGE TABLE statement
- Changing the partition storage conditions of a table with the ALTER TABLE statement

If you do perform any of these data operations, you must verify the integrity of the data. For details about how to verify data integrity, see the *HiRDB Version 9 Installation and Design Guide*. If USE is specified in the pd_check_pending operand, the referencing table is placed in check pending status if you perform any of these data operations.

# 5.9 Check constraints

A characteristic of databases is that value ranges, conditions, and other limitations are often imposed on the data contained in the tables. For example, when product information is stored in a database, the product cost cannot be a negative value. This means that there must not be negative product cost values in the database, which suggests that these values must be checked when they are inserted or updated. A check constraint is a function that maintains the integrity of table data by checking a constraint condition when data is inserted or updated and by suppressing operations on data that does not satisfy the condition. In this manual, a table for which a check constraint has been defined is called a check constraint table.

Note that execution of a utility might make it impossible to guarantee the integrity of data in a check constraint table. When this occurs, the check constraint table is placed in check pending status. For details about check pending status, see *5.10 Check pending status*.

For details about check constraints, see the *HiRDB Version 9 Installation and Design Guide*.

**Advantages of check constraints**

Defining a check constraint reduces the workload required to create a UAP, because the checking of data when it is inserted or updated is automated. The downside is that updating check constraint tables takes longer because of the processing time required to check the integrity of the data.

## (1) Defining a check constraint

To define a check constraint, you first specify `CHECK` with `CREATE TABLE` of a definition SQL, and then use a search condition to specify the constraint condition for the table value To use the check pending status, specify `USE` in the `pd_check_pending` operand or omit specification of the `pd_check_pending` operand.

## (2) Data manipulation and integrity

When you use a data manipulation SQL to update, add, or delete data in a check constraint table, HiRDB performs checking when the SQL is executed in order to guarantee data integrity. However, if you perform certain data operations with the utilities listed below, HiRDB might no longer be able to guarantee data integrity.

- Loading data with the database load utility (`pdload`)
- Reloading data with the database reorganization utility (`pdrorg`)
- Updating with updatable online reorganization (`pdorend`) (UNIX edition only)

If you do perform any of these data operations, you must verify the integrity of the data. For details about how to verify data integrity, see the *HiRDB Version 9 Installation and Design Guide*. When `USE` is specified in the `pd_check_pending` operand, the referencing table is placed in check pending status if you perform any of these data operations.

# 5.10 Check pending status

When the integrity of table data can no longer be guaranteed, such as when SQL code or a utility has been executed on a table for which a referential constraint or a check constraint is defined, HiRDB limits the data operations that can be performed on the referencing table or the check constraint table. The status in which data operations are limited in this way because integrity cannot be guaranteed is called *check pending status*. To place a referencing table or a check constraint table in check pending status in order to limit data operations, either you must specify `USE` in the `pd_check_pending` operand or you must omit specification of the `pd_check_pending` operand. You can remove check pending status from a table by executing the *integrity check utility* (`pdconstck`). You can also use the integrity check utility to forcibly place a table in check pending status.

If you specify `NOUSE` in the `pd_check_pending` operand, data operations are not limited even if referential integrity between tables can no longer be guaranteed. In such a case, when you execute SQL code or a utility that might cause integrity to be lost, it is advisable to use the integrity check utility to forcibly place the table in check pending status and then check its integrity.

For details about check pending status and how to check the integrity of data on which the integrity check utility has been applied, see the *HiRDB Version 9 Installation and Design Guide*.

## (1) Events that set or remove check pending status

In addition to the integrity check utility, the following utilities, commands, and SQL code can assign or remove check pending status for a referencing table.

- Specification with the `constraint` statement of the database load utility (`pdload`)
- Specification with the `constraint` statement of the database reorganization utility (`pdrorg`) (reload or reorganize)
- Use of the database structure modification utility (`pdmod`) (re-initialization of an RDAREA)
- Updating with updatable online reorganization (`pdorend -p` command) (UNIX edition only)
- `PURGE TABLE` statement
- `ALTER TABLE` (`CHANGE RDAREA`)

For details about these utilities and commands, see the manual *HiRDB Version 9 Command Reference*. For details about SQL, see the manual *HiRDB Version 9 SQL Reference*.

## (2) Operations that are limited for a check pending status table

The following table lists the operations that cannot be performed on a table that has been placed in check pending status.

Table 5–2: Operations that cannot be performed on a check pending status table

| Operation attempted on check pending status table | Result |
|---|---|
| Rebalancing utility (`pdrbal`) | Cannot be performed. |
| `SELECT` statement (searching a target table or searching a list created from a target table)<br>`INSERT` statement (insertion of data into a target table)<br>`UPDATE` statement (updating a target table)<br>`DELETE` statement (deleting rows from a target table)<br>`ASSIGN LIST` statement (creating a list from a target table) | Each of these operations can be performed if either of the following conditions is satisfied; these operations cannot be performed in any other case:<br>• The operation is performed on a partitioned table whose storage condition is either key range partitioning or FIX hash partitioning<br>• The RDAREA on which the operation is performed is not in check pending status |
| `pdrorg` (reorganizing) | You might not be able to perform reorganization on a partitioned table that has been reorganized with flexible hash partitioning. For details, see the section on the *Database Reorganization Utility (pdrorg)* in the manual *HiRDB Version 9 Command Reference*. |

# 5.11 Improving database access performance

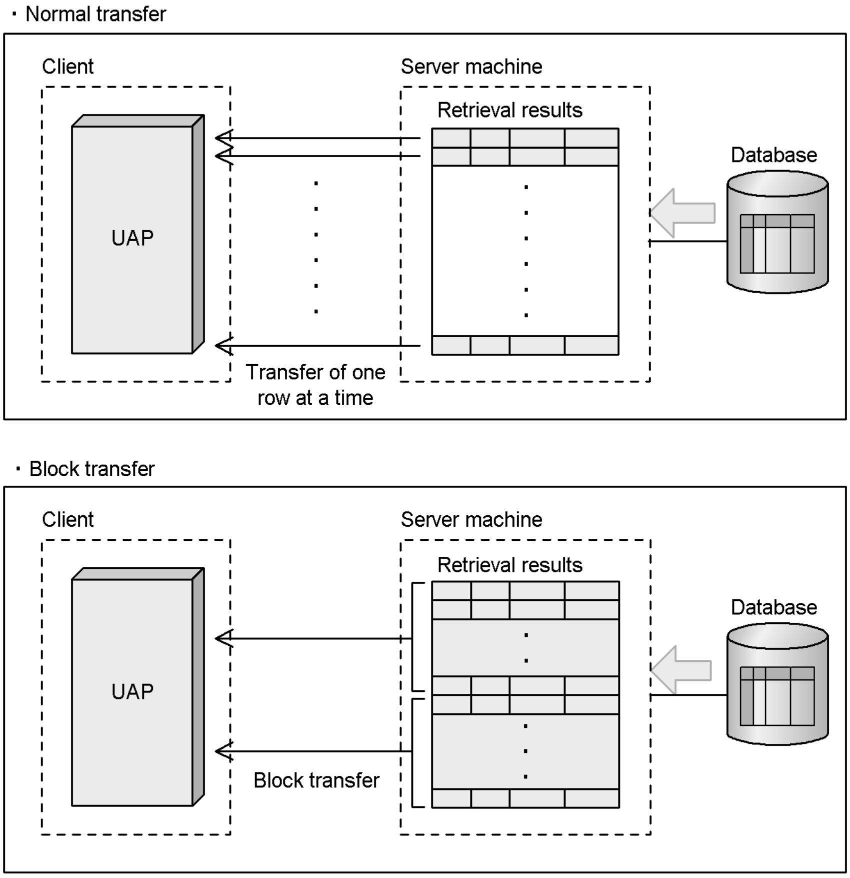
HiRDB provides the following functions that are designed to improve database access performance:

- Block transfer facility
- Rapid grouping facility
- Functions that use arrays
- Holdable cursor
- SQL optimization

## 5.11.1 Block transfer facility

The facility for transferring data from a HiRDB server to a HiRDB client in blocks of any number of rows is called the block transfer facility. The block transfer facility can improve retrieval performance when it is used for accessing a HiRDB server from a HiRDB client to retrieve a large volume of data. Although increasing the number of rows to be transferred can decrease the communications overhead and reduce retrieval time, it also increases the amount of memory required. Moreover, if the number of rows to be transferred is too large, resend will occur during communication, which requires additional time. The larger the number of rows to be transferred, the more frequently resend occurs, negating the benefit of the block transfer facility. For this reason, make sure that the number of rows specified for block transfer is not too large. The following figure provides an overview of the block transfer facility.

Figure 5–12: Overview of block transfer facility



**Using the block transfer facility**

The block transfer facility can be used when the following conditions are satisfied:

1. A value of 2 or greater is specified for PDBLKF in the client environment definition or a value of 1 or greater is specified in PDBLKBUFFSIZE.
2. The FETCH statement is specified, except under any of the following conditions:
   - An update uses a cursor
   - A search includes a BLOB-type selection expression
   - A search includes a BINARY-type selection expression whose definition length is more than 32,000 bytes, with the PDBINARYBLKF client environment definition set to NO (or omitted)
   - A search accepts a result that uses a BLOB locator type or BINARY locator type variable, and uses a holdable cursor

For details about client environment definition, see the *HiRDB Version 9 UAP Development Guide*.

## 5.11.2 Rapid grouping facility

When the GROUP BY clause of an SQL is used for grouping, sorting is performed before grouping. The facility for combining this grouping with hashing to achieve rapid grouping is called the rapid grouping facility. This facility can reduce the amount of time required for grouping when the number of groups is small and there is a large number of rows to be grouped.

**Specifying the rapid grouping facility**

To specify the rapid grouping facility, use an SQL optimization option, such as one of the following operands:

- pd_optimize_level system definition operand
- PDSQLOPTLVL client environment definition
- CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE, ALTER PROCEDURE, ALTER ROUTINE, or ALTER TRIGGER SQL optimization options

For details about the pd_optimize_level operand, see the manual *HiRDB Version 9 System Definition*. For specification of the PDSQLOPTLVL statement and details of the rapid grouping facility, see the *HiRDB Version 9 UAP Development Guide*.

## 5.11.3 Functions that use arrays

### (1) FETCH facility using arrays

When an arrayed embedded variable is specified for the INTO clause of a FETCH statement, it becomes possible to obtain multiple rows of retrieval results at the same time. The FETCH facility using arrays can improve retrieval performance when it is used for accessing a HiRDB system from a HiRDB client to retrieve a large volume of data. Unlike the block transfer facility, the FETCH facility using arrays requires the fact that multiple rows of retrieval results will be obtained to be described in a program. The FETCH facility using arrays is valid only when the embedded variables and indicator variables specified in the INTO clause are all arrayed. For details about the FETCH facility using arrays, see the *HiRDB Version 9 UAP Development Guide*.

### (2) INSERT facility using arrays

The INSERT facility using arrays makes it possible to execute a single SQL INSERT statement to insert multiple rows of data by specifying an array type variable in which the multiple rows of data are set. By using the INSERT facility using arrays, you can greatly reduce the frequency of communications between the HiRDB client and the HiRDB server. If you are using a HiRDB parallel server configuration, you can further reduce the frequency of communications among server applications on the HiRDB server as well. Accordingly, this facility works well for cases in which a HiRDB client accesses a HiRDB server to quickly insert a large amount of data.

**How to use the INSERT facility using arrays**

**Static execution**

With the `INSERT` statement, specify an embedded variable in the `FOR` clause, and change all embedded variables and indicator variables specified in the `VALUES` clause to be array type variables. Use the embedded variable specified in the `FOR` clause to control the number of rows that are inserted at one time.

**Dynamic execution**

Using the following procedure to execute:

1. Using the `PREPARE` statement, preprocess the `INSERT` statement (specify the `?` parameter for all values in the `VALUES` clause that are to be inserted).
2. In the `USING` clause of the `EXECUTE` statement, use an array to specify the values to be assigned to the `?` parameters entered in the preprocessed `INSERT` statement, and specify the embedded variable in the `BY` clause. Use the embedded variable specified in the `BY` clause to control the numbers of rows that are inserted at one time.

For details about the INSERT facility using arrays, see the *HiRDB Version 9 UAP Development Guide*.

## (3) UPDATE facility using arrays

The UPDATE facility using arrays enables you to execute a single SQL `UPDATE` statement to update multiple table columns by specifying an array-type variable in which multiple data items are set. Because the frequency of communication between the HiRDB client and the HiRDB server is reduced, this facility works well in cases where a HiRDB client accesses a HiRDB server to perform high-speed updating of a large amount of data.

**How to use the UPDATE facility using arrays**

**Static execution**

Specify an embedded variable in the `FOR` clause of the `UPDATE` statement, and then change all of the embedded variables and indicator variables specified in the search conditions so that they are array-type variables. Use the embedded variable specified in the `FOR` statement to control the number of data items that are updated at one time.

**Dynamic execution**

To perform dynamic execution:

1. Use a `PREPARE` statement to preprocess the `UPDATE` statement (specify the `?` parameter for the update values or in the search condition).
2. In the `USING` clause of the `EXECUTE` statement, use an array to specify the values to be assigned to the `?` parameter entered in the preprocessed `UPDATE` statement, and specify an embedded variable in the `BY` clause. Use the embedded variable specified in the `BY` clause to control the number of data items that are updated at one time.

For details about the UPDATE facility using arrays, see the *HiRDB Version 9 UAP Development Guide*.

## (4) DELETE facility using arrays

The DELETE facility using arrays enables you to execute a single SQL `DELETE` statement to delete multiple rows by specifying an array-type variable in which multiple data items are set. Because the frequency of communication between HiRDB client and HiRDB server is reduced, this facility works well in cases where a HiRDB client accesses a HiRDB server to perform high-speed deletion of a large amount of data.

**How to use the DELETE facility using arrays**

**Static execution**

Specify an embedded variable in the `FOR` clause of the `DELETE` statement, and then change all of the embedded variables and indicator variables specified in the search conditions so that they are array-type variables. Use the embedded variable specified in the `FOR` statement to control the number of data items that are deleted at one time.

**Dynamic execution**

To perform dynamic execution:

1. Use the PREPARE statement to preprocess the DELETE statement (specify the ? parameter in the search conditions).
2. In the USING clause of the EXECUTE statement, use an array to specify the values to be assigned to the ? parameter entered in the preprocessed DELETE statement, and specify the embedded variable in the BY clause. Use the embedded variable specified in the BY clause to control the number of data items that are deleted at one time.

For details about the DELETE facility using arrays, see the *HiRDB Version 9 UAP Development Guide*.

## 5.11.4 Holdable cursor

A cursor that is not to be closed even when a COMMIT statement is executed is called a holdable cursor. When a holdable cursor is specified, it can be kept open until one of the following SQL statements is executed:

- CLOSE statement
- DISCONNECT statement (including the DISCONNECT statement that is executed automatically by the system after an error occurs)
- ROLLBACK statement (including the ROLLBACK statement that is executed automatically by the system after an error occurs)

A holdable cursor can reduce the incidence of locked resources, because it is possible that a COMMIT statement will be executed in the middle of a retrieval or in the midst of updating a large volume of data. Furthermore, because the COMMIT statement can be executed while the cursor is open, synchronization point dumps can be enabled to reduce restart time even when a large volume of data must be retrieved or updated. For details about holdable cursors, see the *HiRDB Version 9 UAP Development Guide*.

**Specifying a holdable cursor**

To use a holdable cursor, specify UNTIL DISCONNECT or WITH HOLD in DECLARE CURSOR.

## 5.11.5 SQL optimization

Taking into account the database status to determine the most efficient access paths is called *SQL optimization*. SQL optimization includes the *SQL optimization specifications*, the *SQL optimization options*, and the *SQL extension optimizing options*. Table 5-3 lists and describes the SQL optimization specification facilities, Table 5-4 lists and describes the SQL optimization option facilities, and Table 5-5 lists and describes the SQL extension optimizing option facilities. Note, however, that there are additional SQL optimization facilities that are not described here. Because these facilities invariably improve performance, they are not provided as options, but are always used. For details about the SQL optimization specifications, SQL optimization options, and SQL extension optimizing options, see the *HiRDB Version 9 UAP Development Guide*.

Table 5–3: SQL optimization specification facilities

| SQL optimization specification facility | Description |
|---|---|
| SQL optimization specification for index utilization | Enables you to use table searches to specify which index to utilize. It also enables you to use table searches to specify not to utilize an index (by using a table scan).<br>You can use the following SQL methods to specify an SQL optimization specification for index utilization:<br>• Table expressions<br>• DELETE statement<br>• UPDATE statement, format 1 |
| SQL optimization specification for the join method | Enables you to specify a join method (nest-loop-join, hash join, or merge join) for join tables.<br>You can use the following SQL method to specify an SQL optimization specification for the join method:<br>• Table expressions |

| SQL optimization specification facility | Description |
|---|---|
| SQL optimization specification for the subquery execution method | Enables you to specify whether to set hash execution as the method of executing subqueries in predicates.<br>You can use the following SQL method to specify the SQL optimization specification for the subquery execution method:<br>• Subqueries |

Table 5–4: Facilities of SQL optimization options

| SQL optimization option facility | Explanation |
|---|---|
| Forced nest-loop-join | If an index is defined in the joining condition column, only nest loop join is used for joining. |
| Creating multiple SQL objects | Creates multiple SQL objects in advance and selects the optimal SQL object based on an embedded variable or ? parameter value during execution. |
| Increasing the target floatable servers (back-end servers for fetching data) | Normally, a back-end server not used for fetching data is used as a floating server. When this optimization method is used, a back-end server used for fetching data is also used as a floating server. However, because the number of back-end servers that can be used as floating servers is computed by HiRDB, not all back-end servers can be used as floating servers. To use all back-end servers, specify both this facility and the increasing the number of floatable server candidates facility. |
| Prioritized nest-loop-join | If an index is defined for the joining condition column, nest-loop-join is used for the joining process with a higher priority. |
| Increasing the number of floatable server candidates | Normally, the needed number of floating servers is computed by HiRDB from the available floating servers and allocated. When this optimization method is applied, all available floating servers are used. However, back-end servers used for fetching data cannot be used as floating servers. To also use the back-end servers used for fetching data as floating servers, specify this facility together with the increasing the target floatable servers (back-end servers for fetching data) facility. |
| Priority of OR multiple index use | Specify this facility to use a retrieval method that uses OR multiple indexes with a higher priority. OR multiple index use is a method that evaluates a search condition by using an index to retrieve each condition from multiple conditions joined with ORs in a search condition and obtains a sum set of the retrieval results. |
| Group processing, ORDER BY processing, and DISTINCT set function processing at the local back-end server | Specify this facility to perform group processing, ORDER BY processing, and DISTINCT set function processing at the back-end server that defines the table (local back-end server), without having to use a floating server. |
| Suppressing use of AND multiple indexes | Ensures that an access path using AND multiple indexes is never used.<br>• When the search condition contains multiple conditions linked with AND and different indexes are defined for the individual columns (such as in SELECT ROW FROM T1 WHERE C1=100 AND C2=200), use of AND multiple indexes means that individual indexes are used to create work tables for the rows that satisfy the condition and the product set of these rows is obtained. |
| Rapid grouping facility | Uses hashing to rapidly process the grouping specified by the GROUP BY clause of an SQL. |
| Limiting the target floatable servers (back-end servers for fetching data) | Normally, a back-end server not used for fetching data is used as a floating server. When this optimization method is used, only those back-end servers used for fetching data are used as floating servers. |
| Separating data collecting servers | If increasing the target floatable servers (back-end servers for fetching data) and limiting the target floatable servers (back-end servers for fetching data) are both specified, they will function as the facility for separating data collecting servers. If the facility for separating data collecting servers is applied, a back-end server other than the data transfer source is allocated for data collection to an SQL that needs to collect data from multiple back-end servers into a single back-end server. Back-end servers other than those used for collecting data (including back-end servers for fetching data) are allocated as front-end servers for other purposes. |
| Suppressing index use (forced table scan) | Normally, HiRDB determines whether indexes are used. When this optimization method is applied, methods that do not use indexes are given higher priority. However, index use cannot be suppressed in the following cases: joining results in a nest-loop-join, a structured repetition predicate is specified as the search condition, or a condition includes the index type plug-in dedicated facility. |

| SQL optimization option facility | Explanation |
|---|---|
| Forcing use of multiple indexes | Specify this optimization method to retrieve tables by forcibly selecting use of AND multiple indexes. When this optimization method is not specified and multiple conditions linked with AND have been specified, two indexes at most are used even if use of AND multiple indexes is selected. The number of indexes to be used will vary slightly depending on the table definition, index definition, and search condition. When this optimization is specified, all conditions that can narrow the search range using indexes will be used. When AND multiple indexes are used, search candidates can be narrowed down to a certain degree using these indexes. Moreover, AND multiple indexes are effective when there is little overlap in the product set. |
| Suppressing creation of update-SQL work tables | If this optimization is specified when index key value no locking is being applied, HiRDB does not create work tables for internal processing even if an index is used for a retrieval with the `FOR UPDATE` clause specified, an `UPDATE` statement, or a `DELETE` statement. Therefore, SQL statements can be processed at high speed. Work tables will be created if the index key value no-locking option is not being applied. Whether indexes are being used can be determined from the access path display utility. |
| Derivation of rapid search conditions | If this optimization facility is specified, rapid search conditions are derived. Rapid search conditions refer to conditions ranging from search conditions in the `WHERE` clause and `ON` search conditions in the `FROM` clause, to new conditions derived from CNF conversions or condition transitions. Deriving rapid search conditions improves search performance because it allows you to narrow down the rows that are searched at an early stage. However, generating and executing rapid search conditions takes time, or unexpected access paths might be used. In general, we recommend that you avoid specifying this optimization option. Instead, specify rapid search conditions directly in an SQL statement. |
| Application of scalar operation-included key conditions | If this optimization facility is specified, within the limitations that specify scalar operations, if all columns included in the scalar operation are index configuration columns, searching is narrowed by evaluating these columns by one index key value at a time. These conditions are evaluated as key conditions. |
| Facility for batch acquisition from functions provided by plug-ins | If you specify a function provided by a plug-in as a search condition and HiRDB performs the search using a plug-in index, HiRDB normally acquires the results (row location information and the passed value if needed) returned from the function provided by the plug-in one row at a time. By applying this optimization, HiRDB can acquire in a batch multiple rows of results returned from the function provided by the plug-in, which reduces the number of times that HiRDB has to call the function. |
| Facility for moving search conditions into derived tables | Normally, a work table for a derived table also stores rows that do not match the search condition for the derived table's columns. By applying this optimization option, HiRDB creates a work table for the derived table after removing the rows that do not match the search condition for the derived table's columns. As a result, the size of the work table and the number of input/output operations involving the work table can be reduced. Furthermore, using such an access path sometimes makes it possible to use an index that can more effectively narrow the rows to be searched. We recommend that you use this function when you install HiRDB Version 08-02 or newer for the first time. |

Table 5–5: Facilities of SQL extension optimizing options

| SQL extension optimizing option facility | Explanation |
|---|---|
| Application of optimizing mode 2 based on cost | Optimizes using optimizing mode 2 based on cost. |
| Hash-execution of a hash join or a subquery | Optimizes by applying hash join for a joined retrieval. If the retrieval is accompanied by a subquery, the subquery is processed by hashing. Determine whether hash-execution of a hash join or a subquery is to be applied by taking into consideration the join method and external reference. |
| Facility for applying join conditions that include value expressions | Creates a join condition for conditions that include value expressions. |
| Applying search conditions that include scalar operations to an index search | Improves execution performance of scalar operations that can be evaluated before a search by using an index to search with search conditions that are included in condition values for columns. |
| Enabling the substructure index to the `XMLEXISTS` predicate, including the parameters | Makes a substructure index usable when the following two conditions are satisfied: |

| SQL extension optimizing option facility | Explanation |
|---|---|
| | • CAST specification is specified for the XML query variable of the XMLEXISTS predicate, and the ? parameter, an SQL parameter, or an SQL variable is specified for the value expression of the CAST specification.<br>• The substructure specification of the defined index matches the substructure path specified as a condition in the XML query of the XMLEXISTS predicate. |
| Applying merge to derived tables in a FROM clause | HiRDB does not need to create a work table when it executes an SQL statement that specifies only simple derived tables in the FROM clause, such as row selections or column projections. This can improve the execution performance of SQL statements that specify derived tables in the FROM clause.<br>For the conditions under which HiRDB does not create a work table, see *Rules on derived tables in a FROM clause* under *Table reference* in the manual *HiRDB Version 9 SQL Reference*. |
| Facility for converting outer joins to inner joins | Converts the outer join to an inner join in cases where an outer join was specified for a query, and it is obvious that the same results would be obtained if an inner join were specified.<br>This optimization can be expected to improve execution performance, not only by avoiding unnecessary processing, but also by making it possible to apply an index to the search conditions for the columns of the internal table specified in the WHERE clause. |

**Setting SQL optimization options and SQL extension optimizing options**

The following methods can be used to specify SQL optimization options and SQL extension optimizing options:

1. Specifying the pd_optimize_level or pd_additional_optimize_level operand of the system common definition or front-end server definition
2. Specifying PDSQLOPTLVL and PDADDITIONALOPTLVL in the client environment definition
3. Specifying an SQL optimization option or SQL extension optimizing option in an SQL statement of a stored routine or trigger.

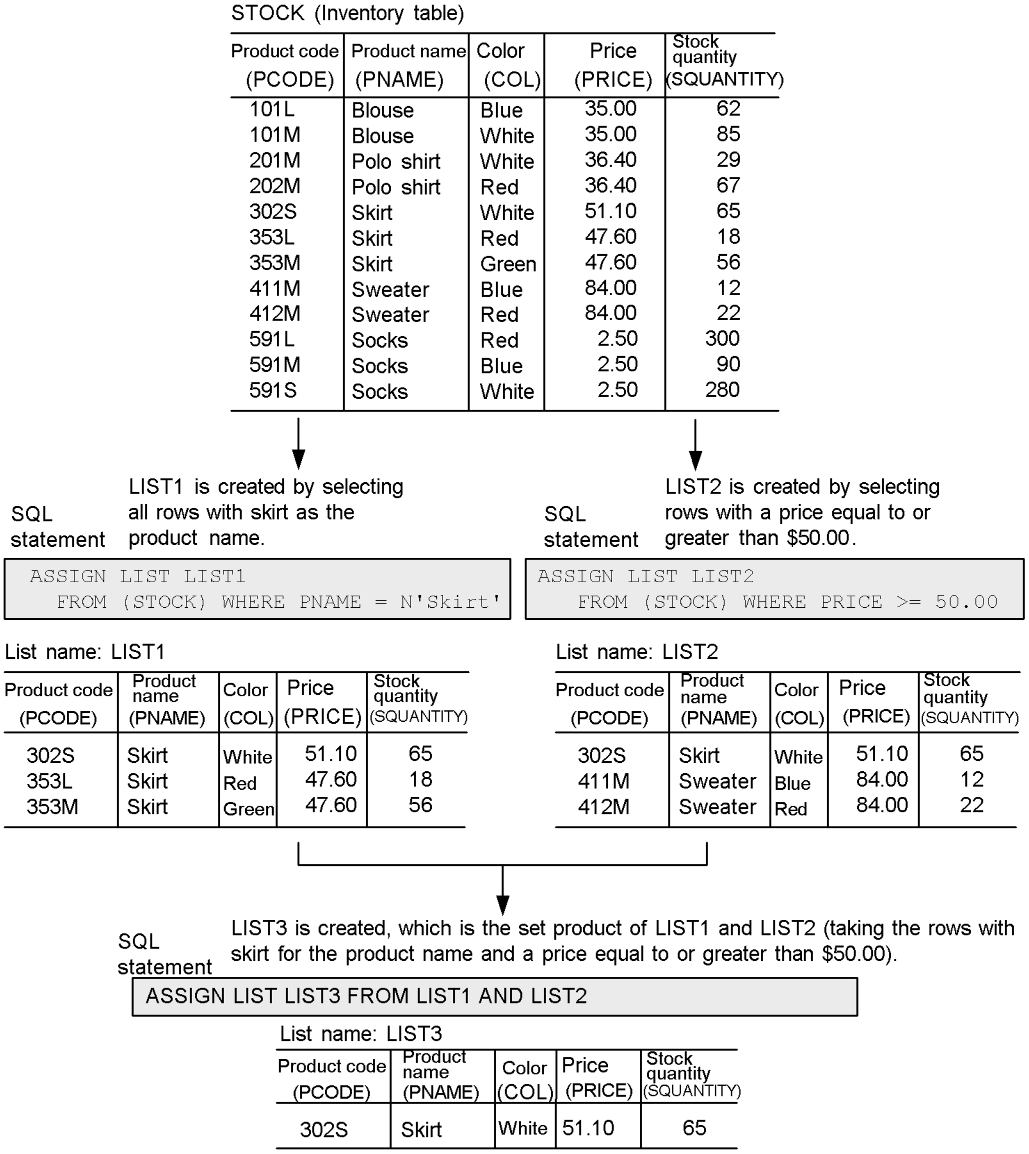
If more than one of these methods is specified concurrently, the order of precedence is 3, 2, 1. Furthermore, if you also specify an SQL optimization specification at the same time, the SQL optimization specification has precedence over any SQL optimization option or SQL extension optimizing option.

# 5.12 Narrowed search

*Narrowed search* is the process of retrieving records by gradually narrowing the search processing. In a narrowed search, the `ASSIGN LIST` statement generates a *list*, which refers to either saved data or a set of saved data with a temporary name (list name). The list is created during the process so that information can be retrieved by gradually narrowing the search for data by specifying conditions until an appropriate number of data items can be identified.

If an existing list created using known identical or substantially similar conditions is available, processing performance can be improved by using it. When multiple conditions are specified, a search can be performed by combining multiple lists. For details about narrowed searches, see the *HiRDB Version 9 UAP Development Guide*. The following figure shows an example of a search using a list.

Figure 5–13: Example of a search using a list



STOCK (Inventory table)

| Product code<br>(PCODE) | Product name<br>(PNAME) | Color<br>(COL) | Price<br>(PRICE) | Stock quantity<br>(SQUANTITY) |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |
| 202M | Polo shirt | Red | 36.40 | 67 |
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |
| 591L | Socks | Red | 2.50 | 300 |
| 591M | Socks | Blue | 2.50 | 90 |
| 591S | Socks | White | 2.50 | 280 |

SQL statement

LIST1 is created by selecting all rows with skirt as the product name.

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

SQL statement

LIST2 is created by selecting rows with a price equal to or greater than $50.00.

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

List name: LIST1

| Product code<br>(PCODE) | Product name<br>(PNAME) | Color<br>(COL) | Price<br>(PRICE) | Stock quantity<br>(SQUANTITY) |
|---|---|---|---|---|
| 302S | Skirt | White | 51.10 | 65 |
| 353L | Skirt | Red | 47.60 | 18 |
| 353M | Skirt | Green | 47.60 | 56 |

List name: LIST2

| Product code<br>(PCODE) | Product name<br>(PNAME) | Color<br>(COL) | Price<br>(PRICE) | Stock quantity<br>(SQUANTITY) |
|---|---|---|---|---|
| 302S | Skirt | White | 51.10 | 65 |
| 411M | Sweater | Blue | 84.00 | 12 |
| 412M | Sweater | Red | 84.00 | 22 |

SQL statement

LIST3 is created, which is the set product of LIST1 and LIST2 (taking the rows with skirt for the product name and a price equal to or greater than $50.00).

```
ASSIGN LIST LIST3 FROM LIST1 AND LIST2
```

List name: LIST3

| Product code<br>(PCODE) | Product name<br>(PNAME) | Color<br>(COL) | Price<br>(PRICE) | Stock quantity<br>(SQUANTITY) |
|---|---|---|---|---|
| 302S | Skirt | White | 51.10 | 65 |

To retrieve the data whose product name is skirt and whose unit price is equal to or greater than $50.00 from the inventory table (STOCK), you should search list LIST3, which is faster than searching the inventory table with conditions specified.

SQL statement

```
SELECT * FROM LIST LIST3
```

↓

| Product code<br>(PCODE) | Product name<br>(PNAME) | Color<br>(COL) | Price<br>(PRICE) | Stock quantity<br>(SQUANTITY) |
|---|---|---|---|---|
| 302S | Skirt | White | 51.10 | 65 |

### Preparations for performing a narrowed search

Following are the prerequisites for conducting narrowed searches:

#### Specifications in a system definition

It is necessary that the following two operands related to narrowed searches be specified in the system common definition:

- `pd_max_list_users`: Specifies the maximum number of users who are to be permitted to create lists.
- `pd_max_list_count`: Specifies the maximum number of lists that one user will be permitted to create.

For details about the narrowed search operands in the system common definition, see the manual *HiRDB Version 9 System Definition*.

#### Creating a list RDAREA

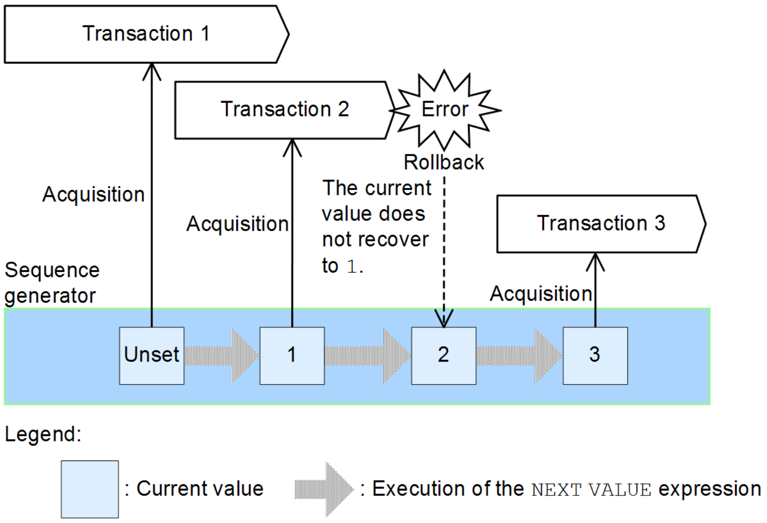
A list RDAREA can be created with either the database initialization utility (`pdinit`) or the database structure modification utility (`pdmod`). For details about list RDAREAs, see the *HiRDB Version 9 Installation and Design Guide*.

# 5.13 Automatic numbering facility

The automatic numbering facility returns a serial integer every time data is retrieved from a database. This facility can be used when you define a sequence generator. Using the automatic numbering facility improves the development efficiency of the UAP that performs numbering. Furthermore, the portability from UAPs created by other DBMSs that support a sequence generator also improves. Therefore, we recommend that you use the automatic numbering facility to number jobs.

A sequence generator generates a contiguous number (sequential number) each time regardless of the status of the user or transaction. The following figure provides an overview of a sequence generator.

Figure 5–14: Overview of a sequence generator



Explanation

To acquire the sequential numbers generated by the sequence generator, you use the `NEXT VALUE` expression. This expression acquires the value that comes after the latest value (current value) generated by the sequence generator and updates the current value to the acquired value.

If the `NEXT VALUE` expression is not used at all after the sequence generator is defined, the current value remains unset. If the `NEXT VALUE` expression is used while the current value is unset, the start value of the sequence generator is returned and this value is stored as the current value.

**Note**

The current value is not recovered even if a rollback occurs. Contiguous numbers are generated regardless of the transaction status.

For details about the automatic numbering facility, see the *HiRDB Version 9 UAP Development Guide*.

# 5.14 Accessing databases using DB access products

You can also use DB access products to access your databases. The following DB access products are discussed in this section:

- ODBC Driver
- HiRDB OLE DB Provider
- HiRDB.NET Data Provider
- JDBC Driver
- SQLJ

These DB access products are included in HiRDB/Run Time and HiRDB/Developer's Kit.

For details about DB access products, see the *HiRDB Version 9 UAP Development Guide*.

## 5.14.1 ODBC Driver

To access HiRDB from an ODBC-compatible application program, you use ODBC Driver. ODBC-compatible application programs include Microsoft Access, Microsoft Excel, and others.

ODBC Driver is also used when a UAP that uses ODBC functions accesses HiRDB. Note that the only ODBC functions that can be used are ones provided by HiRDB.

## 5.14.2 HiRDB OLE DB Provider

To access HiRDB from an OLE DB-compatible application program, you use HiRDB OLE DB Provider.

As with ODBC, OLE DB is an API designed for accessing a wide range of data sources. Interfaces suitable for accessing non-SQL data sources are defined as well.

## 5.14.3 HiRDB.NET Data Provider

To access HiRDB using ADO.NET, you use HiRDB.NET Data Provider. HiRDB.NET Data Provider complies with the ADO.NET specifications.

HiRDB.NET Data Provider bundles a set of common basic interfaces that are provided in the System.Data space of .Net Framework. In addition, it provides as extensions the INSERT facility using arrays and access to repetition columns.

## 5.14.4 JDBC Driver

To access HiRDB from a program written in Java, you use JDBC Driver.

## 5.14.5 SQLJ

SQLJ is a language specification for using Java to write and execute static SQL statements as embedded SQL statements. SQLJ is made up of SQLJ Translator and SQLJ runtime libraries.

SQLJ Translator interprets an SQLJ source program, and generates a profile in which the Java source file and SQL information are stored. The user then runs a Java compiler to compile the Java source file and create a class file (executable file).

The SQLJ runtime libraries are used when the generated profile and class file are executed.

# 6 HiRDB Architecture

This chapter explains the HiRDB architecture.

# 6.1 HiRDB environment setup

Hitachi provides the following tools designed to assist you in setting up a HiRDB environment.

- Simple setup tool
- Automatic generation using the batch file (`SPsetup.bat`) (Windows edition only)

For normal cases, use the simple setup tool to set up a HiRDB environment.

The table below shows the advantages and disadvantages of the various environment setup methods. For details about each of these setup methods, see the *HiRDB Version 9 Installation and Design Guide*.

Table 6–1: Advantages and disadvantages of the environment setup methods

| Environment setup method | Overview | Advantage | Disadvantage |
|---|---|---|---|
| Simple setup tool | You enter HiRDB environment setup information as instructed on windows that are displayed. The HiRDB environment is then set up based on the information you provide. | This is the easiest method to use to set up a HiRDB environment. The simple setup tool assures you of being able to start HiRDB operations. It also allows you to change values already specified in the system definition. | There is a limited range of HiRDB system configurations that can be set up with the simple setup tool. |
| Commands[#1] | You use HiRDB commands to set up a HiRDB environment. | This method gives you complete flexibility to set up a HiRDB system configuration in any way you wish. | A certain amount of knowledge of HiRDB is required in order to use this method. Basically, an understanding of the facilities and settings described in this manual is required. This environment setup method is more difficult than the other methods. |
| Batch file (`SPsetup.bat`) (Windows edition only)[#2] | You execute a batch file to set up a HiRDB environment. | This is an easier method for setting up a HiRDB environment than the commands method. | There is a limited range of HiRDB system configurations that can be set up with a batch file. |

#1

If you are using the Windows edition:

Before you attempt to set up an actual system for production use, try performing a simple installation. Once you become familiar with the procedures for configuring a HiRDB system by using a sample file to set up a test system, you will be able to set up a production-use system that is more appropriate to your needs. For details about how to perform a simple installation, see the *HiRDB Version 9 Installation and Design Guide*.

#2

The batch file method can be used only for a HiRDB single server configuration. For details about how to set up a HiRDB parallel server configuration environment, see *%PDDIR%\HiRDEF\readme.txt*.

! Important note

- You cannot use the simple setup tool to set up a plug-in environment.
- A batch file automatically configures HiRDB environment setting information. The HiRDB system administrator can then make changes to the values to configure a more appropriate environment.

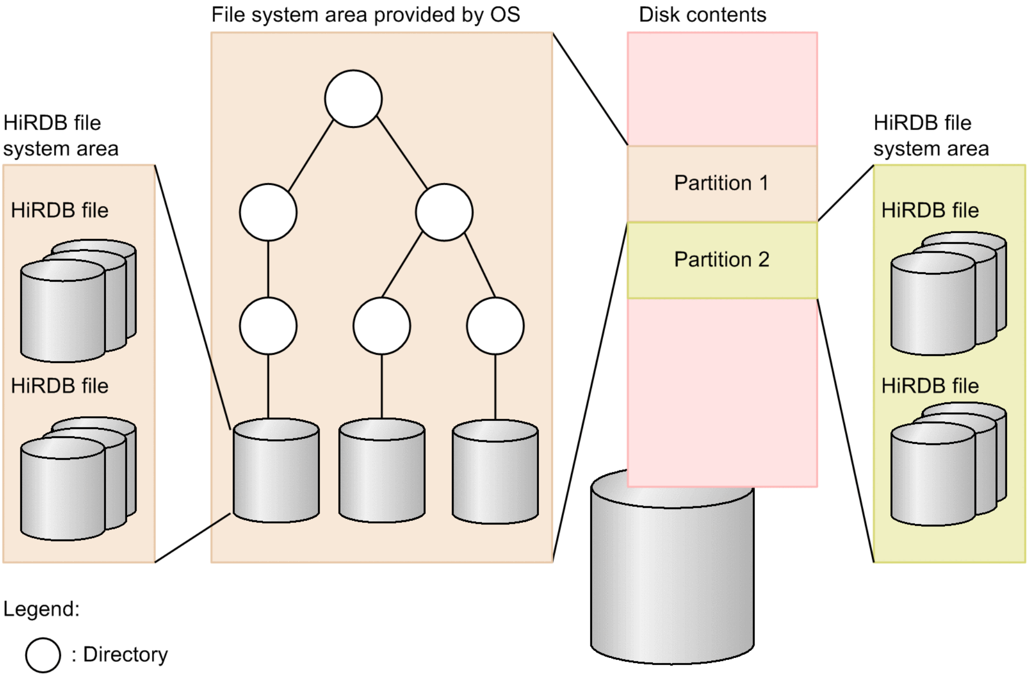
# 6.2 HiRDB file system areas

A special file in HiRDB that stores various types of HiRDB information, such as the information needed to restore the system status in the event of a table or index error, is called a *HiRDB file*. An area in which HiRDB files are created is called a *HiRDB file system area*. A HiRDB file system area must be provided before the special HiRDB files that constitute the system files and RDAREAs are created.

## (1) Relationship between a HiRDB file system area and a file system area provided by the operating system

A disk used by the operating system for performing I/O operations is divided into contiguous areas called *partitions*. A partition can be used as a file system area provided by the operating system or as a HiRDB file system area. The following figure shows the relationship between HiRDB file system areas and file system areas provided by the OS.

Figure 6–1: Relationship between HiRDB file system areas and file system areas provided by the OS



## (2) Files used for HiRDB file system areas

### (a) UNIX edition

HiRDB file system areas can be created in character special files or in regular files.

In the case of Linux, block special files can also be used. HiRDB handles block special files in the same way as character special files. If you are using block special files, note that references to character special files in this manual also apply to block special files.

Data is input to or output from regular files via the kernel buffer. By contrast, data is input to or output from character special files directly from the HiRDB buffer. Although the use of character special files is assumed for most cases, regular files provide higher performance in the following cases:

- HiRDB file of an RDAREA storing a table that normally involves a large-volume search
- Work table files

Note, however, that regular files are not robust against system failures. Therefore, the following files must be created in character special files:

- System log files

- Synchronization point dump files
- Status files
- HiRDB files that comprise a system RDAREA
- HiRDB files that comprise a user RDAREA that is updated frequently

Furthermore, when you use the system switchover facility, use character special files for the HiRDB file system areas to be created on a shared disk device. If you use regular files, the updated content might be lost when a system switchover occurs. Even with character special files, you can improve their large-volume search performance by using the prefetch facility.

#### (b) Windows edition

The HiRDB file system areas are created in Windows partitions.

Normal Windows files

Create a HiRDB file system area by creating files in a normal partition provided by Windows. To do this, use the `pdfmkfs` command.

Direct disk access (raw I/O)

You can use not only normal partitions but also Windows' direct disk access (raw I/O) to create HiRDB file system areas. The latter uses the raw I/O facility. The raw I/O facility lets you access partitions and logical drives in the same manner as files. When the raw I/O facility is used, HiRDB is no longer affected by Windows file caching operations. Therefore, you can maintain stable performance by, for example, controlling global buffers. However, some of the HiRDB file system areas cannot be created with raw I/O. To use the raw I/O facility, you need to provide unformatted partitions. In Windows, you can create partitions from the window displayed by choosing **Computer Management** and then **Disk Management**.

For details about how to create HiRDB files systems with the raw I/O facility, see the *HiRDB Version 9 Installation and Design Guide*.

### (3) HiRDB file system area creation units

We recommend that you create the types of HiRDB files system areas listed and described in the table below. For details about how to design the various HiRDB file system areas, see the *HiRDB Version 9 Installation and Design Guide*.

Table 6–2: Type of HiRDB file system area

| Type of HiRDB file system area | Option# | Description |
|---|---|---|
| RDAREA | `DB` | HiRDB file system area in which RDAREAs (other than list RDAREAs) are stored. This area is always needed. |
| Shared RDAREA | `SDB` | HiRDB file system area in which shared RDAREAs are created. This area is needed if you use shared RDAREAs. |
| System file | `SYS` | HiRDB file system area in which system log files, synchronization point dump files, and status files are stored. This area is always needed. |
| Audit trail file | | HiRDB file system area in which audit trail files are created. This area is needed to use the security audit facility. |
| Work table file | `WORK` | HiRDB file system area in which work table files are stored. This area is always needed. |
| Utility | `UTL` | HiRDB file system area in which files used by utilities (backup files, unload data files, unload log files, index information files, or differential backup management files) are created.<br>Windows edition only:<br>The Windows cache is not used.<br>**About HiRDB file system areas created in HiRDB versions earlier than 09-50**<br>• If a HiRDB file system area was created using `NUTL`, no change exists in behavior in version 09-50 and later. |

| Type of HiRDB file system area | Option# | Description |
|---|---|---|
| | | - If a HiRDB file system area was created using UTL, the Windows cache is not used in version 09-50 and later. |
| | NUTL | Windows edition only:<br>This specification is for compatibility with versions of HiRDB earlier than 09-50. Specifying NUTL is handled as equivalent to specifying UTL. |
| List RDAREA | WORK | HiRDB file system area in which list RDAREAs are stored. This area is needed to perform narrowed searches. |

#: The value of the `-k` option specification that is specified when you create a HiRDB file system area with the `pdfmkfs` command.

## (4) Creating a HiRDB file system area

The `pdfmkfs` command is used to create a HiRDB file system area.

Reference note

If you use either of the following environment setup support tools the first time you install HiRDB, a HiRDB file system area will be created based on the information you enter:

- Simple setup tool
- Batch file (`SPsetup.bat`) (Windows edition only)

For details about designing and creating HiRDB file system areas, see the *HiRDB Version 9 Installation and Design Guide*.

## (5) Maximum size of HiRDB file system areas

The following table shows the maximum size of HiRDB file system areas.

Table 6–3: Maximum size of a HiRDB file system area

| HiRDB type | File type | HiRDB file system area maximum size (MB) |
|---|---|---|
| HP-UX edition | Regular file | 1,048,575 |
| | Character special file | |
| Solaris edition | Regular file | 1,048,575 |
| | Character special file | |
| AIX edition | Regular file (JFS) | 65,411 |
| | Regular file (JFS2) | 1,048,575 |
| | Character special file | |
| Linux edition | Regular file | 1,048,575 |
| | Character special file | |
| Windows edition | Normal Windows file | 1,048,575 |
| | Direct disk access (raw I/O) | |

# 6.3 System files

A file that stores information that is needed in order to recover the system status after an error is called a system file. System files are referred to in general as follows:

- System log files
- Synchronization point dump files
- Status files

## 6.3.1 System log files

A file in which system log information is stored is called a *system log file*. *System log* refers to information on the history of database updating, which is commonly called a log (or *journal* in mainframe terminology). HiRDB collects system log information in system log files for use:

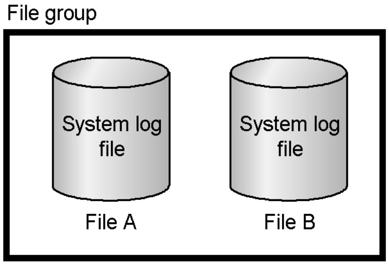
- By HiRDB for recovering the database after HiRDB or a UAP has terminated abnormally.
- By HiRDB administrators as information that is input when using the `pdrstr` command to recover a database.
- By the HiRDB administrator as the input information for acquiring statistical information.

The HiRDB administrator must create system log files as a safeguard against failures, as well as for acquiring statistical information.

### (1) Organization of system log files

HiRDB operates with logical units of system log files called *file groups*. A file group consists of either one or two system log files. A configuration of two system log files is called a *duplexed system log file*. The respective system log files are distinguished by referring to them as *file A* and *file B*. When duplexed system log files are used, HiRDB acquires the same contents into both log files. Thus, if one of the files fails, the other file can be used, thus maintaining system reliability. The following figure shows the organization of system log files.

Figure 6–2: Organization of system log files



### (2) System log file creation

The `pdloginit` command is used to create system log files. Additionally, you specify the following HiRDB system definition operands to create an environment in which the system log files can be used:

- `pdlogadfg` operand (specifies the file group name of the system log files)
- `pdlogadpf` operand (specifies the system log file names that comprise the file group)

For details about designing and creating system log files, see the *HiRDB Version 9 Installation and Design Guide*. For details about using the system log files, see the *HiRDB Version 9 System Operation Guide*.

Reference note

If you use either of the following environment setup support tools when you install HiRDB for the first time, system log files will be created based on the information you enter (the `pdlogadfg` and `pdlogadpf` operands will also be configured):

- Simple setup tool
- Batch file (`SPsetup.bat`) (Windows edition only)

### (3) System log file automatic extension facility

If a capacity shortage occurs in the system log file, the HiRDB system (or unit) terminates abnormally. To avoid this, HiRDB provides a facility for automatically extending the size of the system log file (system log file automatic extension facility). By using this facility, you can reduce the frequency with which the HiRDB system (or unit) terminates abnormally due to a capacity shortage in the system log file. For more information about the system log file automatic extension facility, see the *HiRDB Version 9 System Operation Guide*.

## 6.3.2 Synchronization point dump files

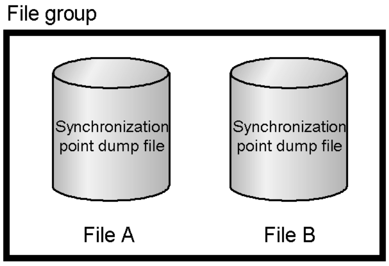
If system logs are the only system files used to recover HiRDB when it has terminated abnormally, recovery processing must employ every system log from the point in time that HiRDB was last started. For this reason, recovering the system from system logs alone might take a significant amount of time. To mitigate this issue, points (called *synchronization points*) are established at fixed time intervals while HiRDB is running, and management information (*synchronization point dumps*) needed for recovery operations is saved at these points. This eliminates the need for system logs prior to the last synchronization point, which shortens the system recovery time.

At each synchronization point, HiRDB incorporates into the database the details of all the database updates that have been made since the last synchronization point or since HiRDB was started. The HiRDB administrator must create synchronization point dump files as a safeguard against failures.

### (1) Organization of synchronization point dump files

HiRDB operates with logical units of synchronization point dump files called *file groups*. A file group consists of either one or two synchronization point dump files. A configuration of two synchronization point dump files is called a *duplexed synchronization point dump file*. The respective synchronization point dump files are distinguished by referring to them as *file A* and *file B*. When synchronization point dump files are duplexed, HiRDB collects and saves the same synchronization point dump information into both files. This enhances system reliability because, even if an error occurs in one of the files, the other file is still available. The following figure shows the organization of synchronization point dump files.

Figure 6–3: Organization of synchronization point dump files


File group
Synchronization point dump file
Synchronization point dump file
File A
File B

### (2) Number of guaranteed valid generations

If the most recent synchronization point dump file that has been generated cannot be read because an error has occurred in the file, HiRDB attempts to read the file that was created one generation earlier. If HiRDB cannot read the file that was created one generation earlier, it attempts to read the file that was created one generation previous to that one. In this way, HiRDB continues attempting to read previous generations of files if the read attempt fails.

The *number of guaranteed valid generations* refers to the number of previous generations of synchronization point dump files for which overwriting is prohibited. For example, if the number of guaranteed valid generations is 2, two generations of synchronization point dump files (the most recent generation and one generation previous to that one) cannot be overwritten. Thus, increasing the number of guaranteed valid generations enhances system reliability because, even if an error occurs in a synchronization point dump file, synchronization point dump files of the number set in the number of guaranteed valid generations will always be available.

Note that the number of synchronization point dump files you need is the number of guaranteed valid generations + 1.

### (3) Synchronization point dump file creation

The `pdloginit` command is used to create synchronization point dump files. Additionally, you specify the following HiRDB system definition operands to create an environment in which the synchronization point dump files can be used:

- `pdlogadfg` operand (specifies the file group name of the synchronization point dump files)
- `pdlogadpf` operand (specifies the synchronization point dump file names that comprises the file group)

For details about designing and creating synchronization point dump files, see the *HiRDB Version 9 Installation and Design Guide*. For details about using synchronization point dump files, see the *HiRDB Version 9 System Operation Guide*.

Reference note

> If you use either of the following environment setup support tools when you install HiRDB for the first time, synchronization point dump files will be created based on the information you enter (the `pdlogadfg` and `pdlogadpf` operands will also be configured):
>
> - Simple setup tool
> - Batch file (`SPsetup.bat`) (Windows edition only)

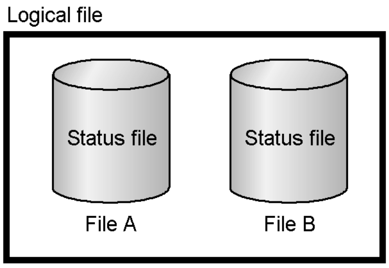
## 6.3.3 Status files

A file that stores system status information required by HiRDB for restarting the system is called a *status file*. The two types of status files are *unit status files* that store restart information for a unit and *server status files* that store restart information for a server. The HiRDB administrator must create status files for use in restarting HiRDB.

### (1) Organization of status files

HiRDB operates with logical units of status files called *logical files*. A single logical file consists of two status files. Status files are duplexed in this manner, and the respective status files are distinguished by referring to them as *file A* and *file B*. HiRDB acquires the same system status information into both files, so that if one of the files fails, the other file can be used, thus maintaining system reliability. The following figure shows the organization of status files.

Figure 6–4: Organization of status files



### (2) Unit status file creation

The `pdstsinit` command is used to create unit status files. Additionally, you specify the `pd_syssts_file_name` operand of the unit control information definition to create an environment in which unit status files can be used.

The logical file name of the status files and the names of the status files that belong to the logical file are specified in the `pd_syssts_file_name` operand.

For details about designing and creating unit status files, see the *HiRDB Version 9 Installation and Design Guide*. For details about using unit status files, see the *HiRDB Version 9 System Operation Guide*.

Reference note

If you use either of the following environment setup support tools when you install HiRDB for the first time, unit status files will be created based on the information you enter (the pd_syssts_file_name operand will also be configured):

- Simple setup tool
- Batch file (SPsetup.bat) (Windows edition only)

### (3) Server status file creation

The pdstsinit command is used to create server status files. Additionally, you specify the pd_sts_file_name operand of the server definition to create an environment in which unit status files can be used. The logical file name of the status files and the names of the status files that belongs to the logical file are specified in the pd_sts_file_name operand.

For details about designing and creating server status files, see the *HiRDB Version 9 Installation and Design Guide*. For details about using server status files, see the *HiRDB Version 9 System Operation Guide*.

Reference note

If you use either of the following environment setup support tools when you install HiRDB for the first time, server status files will be created based on the information you enter (the pd_sts_file_name operand will also be configured):

- Simple setup tool
- Batch file (SPsetup.bat) (Windows edition only)

## 6.3.4 System file components

The following table shows the components that comprise the system files.

Table 6–4: System file components

| Type of system file | | Creation unit | Number | Duplexed? |
|---|---|---|---|---|
| HiRDB single server configuration | System log file | Needed for all units. | 2 to 200 groups | O |
| | Synchronization point dump file | | 2 to 60 groups | O |
| | Server status file | | 1 to 7 files × *number duplexed* | Y |
| | Unit status file | | 1 to 7 files × *number duplexed* | Y |
| HiRDB parallel server configuration | System log file | Needed on all servers other than the system manager. | 2 to 200 groups per server | O |
| | Synchronization point dump file | Needed on all servers other than the system manager. | 2 to 60 groups per server | O |
| | Server status file | Needed on all servers other than the system manager. | 1 to 7 files × *number duplexed per server* | Y |
| | Unit status file | Needed for all units. | 1 to 7 files × *number duplexed per server* | Y |

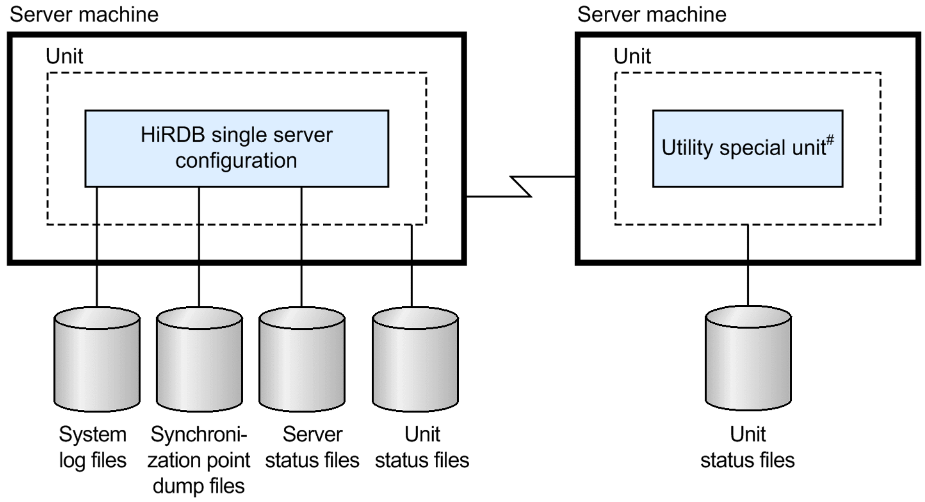
Y: Always duplexed

O: Duplexing optional

### (1) Configuration of system files in a HiRDB single server configuration

The following figure shows the configuration of the system files in a HiRDB single server configuration.

Figure 6–5: Configuration of system files in a HiRDB single server configuration



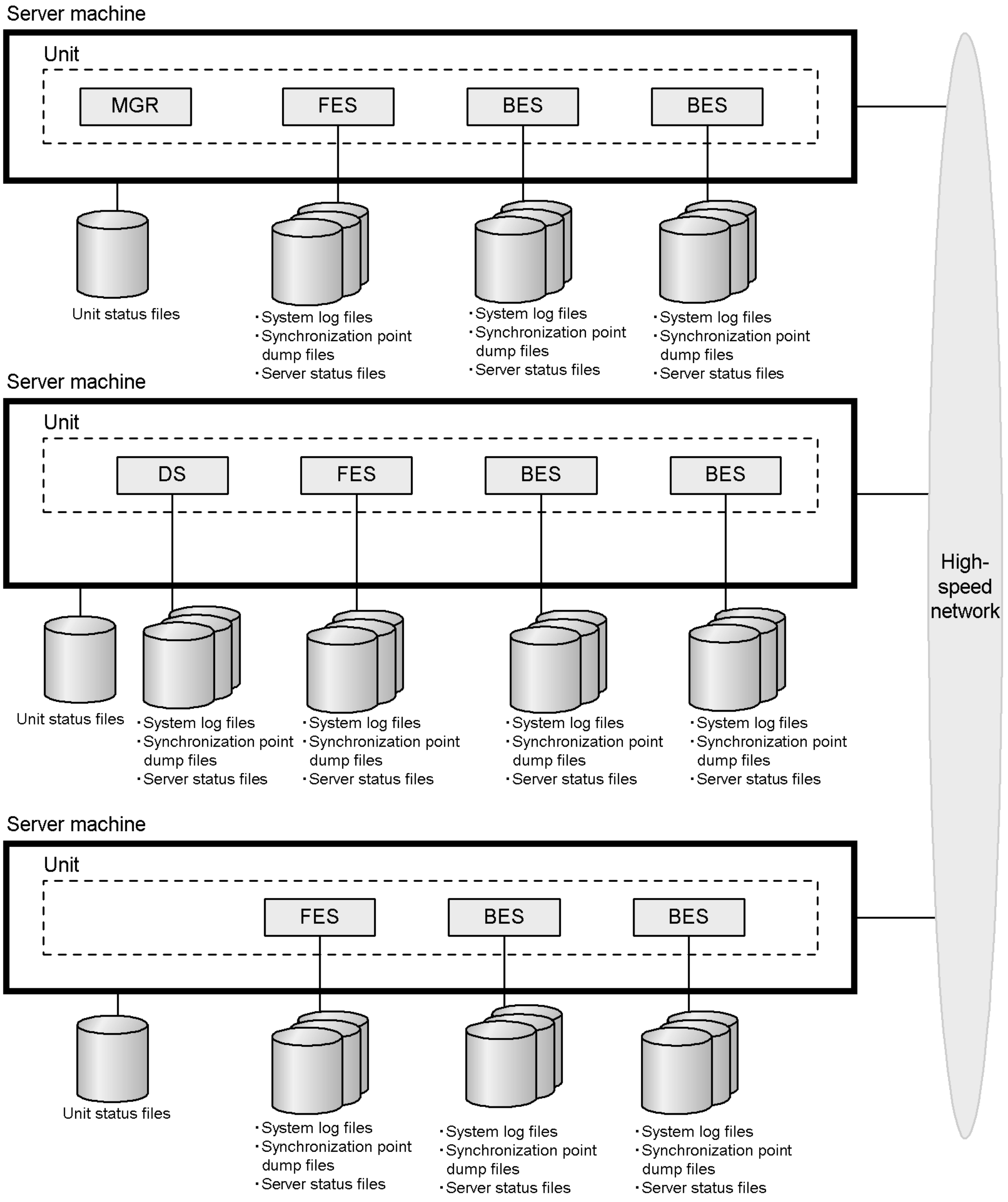
#

The utility special unit is a feature that is limited to the UNIX edition.

## (2) Configuration of system files in a HiRDB parallel server configuration

The following figure shows the configuration of the system files in a HiRDB parallel server configuration.

Figure 6–6: Configuration of system files in a HiRDB parallel server configuration



# 6.4 Work table files

A file that stores temporary information needed for executing an SQL is called a *work table file*. Work table files are created automatically by HiRDB. The HiRDB administrator must create HiRDB file system areas for the work table files.

## (1) Operations that require a work table file

The following operations require a work table file:

- Execution of an SQL statement
- Batch creation of an index
- Re-creation of an index
- Reorganization of an index
- Execution of the rebalancing utility

## (2) Organization of work table files

Work table files are required at the following servers:

- Single servers
- Dictionary servers
- Back-end servers

## (3) Creating a HiRDB file system area for work table files

The `pdfmkfs` command is used to create a HiRDB file system area for work table files. Additionally, you specify the `pdwork` operand of the server definition to create an environment in which a HiRDB file system area for work table files can be created. The name of the HiRDB file system area for work tables is specified in the `pdwork` operand.

For details (size, etc.) about designing and creating work table files, see the *HiRDB Version 9 Installation and Design Guide.*

Reference note

If you use either of the following environment setup support tools when you install HiRDB for the first time, a HiRDB file system area for work table files will be created based on the information you enter (the `pdwork` operand will also be configured):

- Simple setup tool
- Batch file (`SPsetup.bat`) (Windows edition only)

# 6.5 HiRDB system definitions

The operating environment for HiRDB is set by specifications in the HiRDB system definitions. The files in which the HiRDB system definitions are stored are called *HiRDB system definition files*.

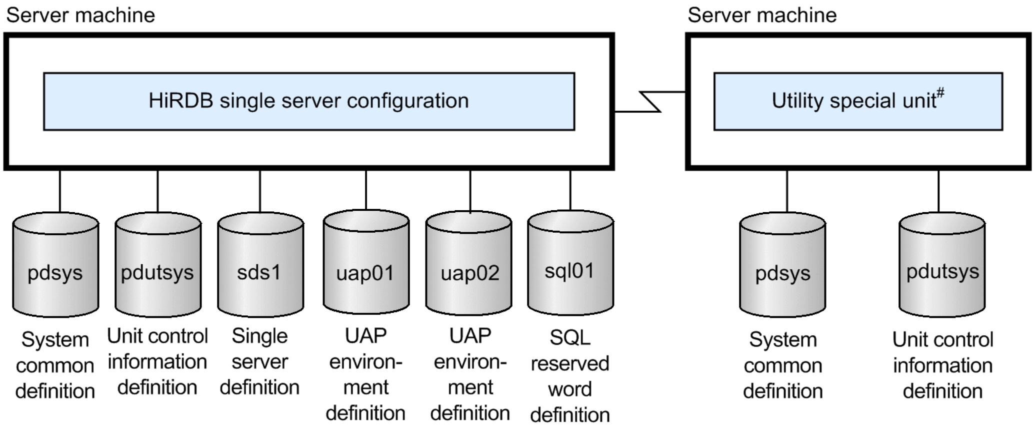
## 6.5.1 HiRDB system definitions for a HiRDB single server configuration

The table below lists the types of HiRDB system definition. The figure following that shows an example of a configuration of HiRDB system definition files.

Table 6–5: HiRDB system definitions (HiRDB single server configuration)

| HiRDB system definition type | Storage file name | Description |
|---|---|---|
| System common definition | `%PDDIR%\conf\pdsys` | Defines the configuration of HiRDB and common information. This file is required for each HiRDB single server configuration. |
| Unit control information definition | `%PDDIR%\conf\pdutsys` | Defines unit control information. This file is required for each unit. |
| Server common definition | `%PDDIR%\conf\pdsvrc` | Defines default values for single server definition operands. This file is optional. |
| Single server definition | `%PDDIR%\conf\`*server-name* | Defines the execution environment for a single server. This file is required for each single server.<br>In the UNIX edition:<br>This file does not need to be defined for a utility special unit. |
| UAP environment definition | `%PDDIR%\conf\pduapenv\`*any-name* | Defines the UAP execution environment. This file is optional. You can create a maximum of 4,096 UAP environment definitions. |
| SQL reserved word definition | `%PDDIR%\conf\pdrsvwd\`*any-name* | Defines SQL reserved words. This file is required to use the SQL reserved word deletion facility. For details about the SQL reserved word deletion facility, see the *HiRDB Version 9 UAP Development Guide*. |

Figure 6–7: Example configuration of HiRDB system definition files (HiRDB single server configuration)


Server machine
Server machine
HiRDB single server configuration
Utility special unit#
pdsys
pdutsys
sds1
uap01
uap02
sql01
pdsys
pdutsys
System common definition
Unit control information definition
Single server definition
UAP environ-ment definition
UAP environ-ment definition
SQL reserved word definition
System common definition
Unit control information definition

Note: Single server definition and UAP environment definition are not required for the utility special unit.

#

The utility special unit is a feature that is limited to the UNIX edition.

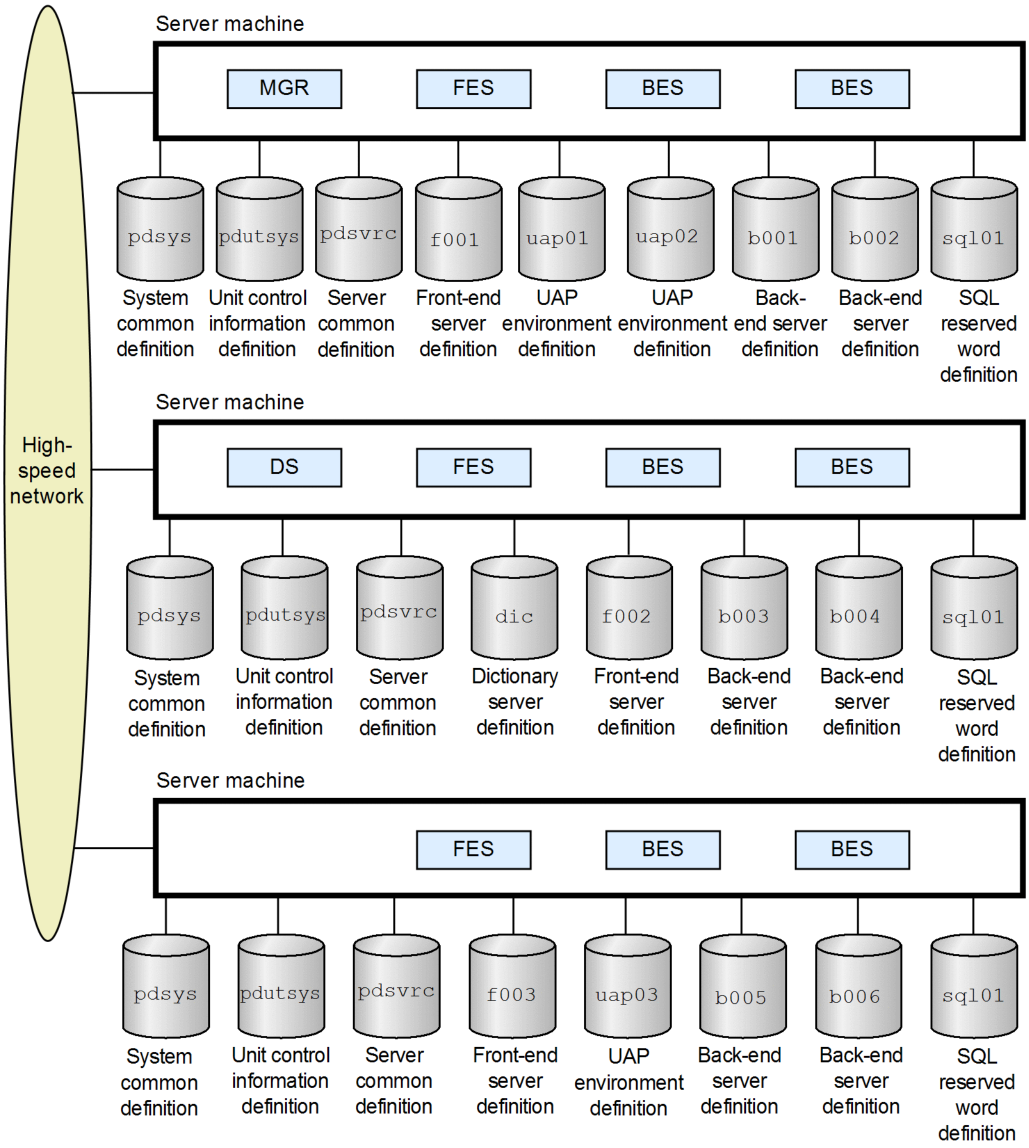
## 6.5.2 HiRDB system definitions for a HiRDB parallel server configuration

The table below lists the HiRDB system definition files. The figure following that shows a configuration example of HiRDB system definition files.

Table 6–6: HiRDB system definitions (HiRDB parallel server configuration)

| HiRDB system definition type | Storage file name | Description |
|---|---|---|
| System common definition | `%PDDIR%\conf\pdsys` | Defines the configuration of HiRDB and common information. This file is required for each unit. All units must have the same system common definition. |
| Unit control information definition | `%PDDIR%\conf\pdutsys` | Defines unit control information. This file is required for each unit. |
| Server common definition | `%PDDIR%\conf\pdsvrc` | Defines default values for individual server definition operands. This file is optional. |
| Front-end server definition | `%PDDIR%\conf\`*server-name* | Defines the execution environment for a front-end server. This file is required for each front-end server. |
| Dictionary server definition | `%PDDIR%\conf\`*server-name* | Defines the execution environment for a dictionary server. This file is required for each dictionary server. |
| Back-end server definition | `%PDDIR%\conf\`*server-name* | Defines the execution environment for a back-end server. This file is required for each backend server. |
| UAP environment definition | `%PDDIR%\conf\pduapenv\`*any-name* | Defines the execution environment for UAPs. This file is optional. You must create UAP environment definitions in a unit that contains a front-end server. If you have multiple front-end servers, define these definitions on the front-end server to which you wish to apply the UAP environment definitions. You can create a maximum of 4,096 UAP environment definitions. |
| SQL reserved word definition | `%PDDIR%\conf\pdrsvwd\`*any-name* | Defines SQL reserved words. This file is required to use the SQL reserved word deletion facility. For details about the SQL reserved word deletion facility, see the *HiRDB Version 9 UAP Development Guide*. |

Figure 6–8: Example configuration of HiRDB system definition files (HiRDB parallel server configuration)


High-speed network
Server machine
MGR
FES
BES
BES
pdsys
System common definition
pdutsys
Unit control information definition
pdsvrc
Server common definition
f001
Front-end server definition
uap01
UAP environment definition
uap02
UAP environment definition
b001
Back-end server definition
b002
Back-end server definition
sql01
SQL reserved word definition
Server machine
DS
FES
BES
BES
pdsys
System common definition
pdutsys
Unit control information definition
pdsvrc
Server common definition
dic
Dictionary server definition
f002
Front-end server definition
b003
Back-end server definition
b004
Back-end server definition
sql01
SQL reserved word definition
Server machine
FES
BES
BES
pdsys
System common definition
pdutsys
Unit control information definition
pdsvrc
Server common definition
f003
Front-end server definition
uap03
UAP environment definition
b005
Back-end server definition
b006
Back-end server definition
sql01
SQL reserved word definition

## 6.5.3 HiRDB system definition file creation

During system building, the HiRDB administrator uses one of the following methods to create HiRDB system definition files:

**Simple setup tool**

For details about how to create HiRDB system definition files with the simple setup tool, see the *HiRDB Version 9 Installation and Design Guide*.

**OS editor**

HiRDB system definition files can be created under `%PDDIR%\conf` with an OS editor and specification of required operands in the HiRDB system definitions.

**Batch file (SPsetup.bat) (Windows edition only)**

Executing the batch file creates HiRDB system definition files automatically under `%PDDIR%\conf`.

For details about creating HiRDB system definition files, see the *HiRDB Version 9 Installation and Design Guide*. For details about the operands of the HiRDB system definitions, see the manual *HiRDB Version 9 System Definition*.

**After creation of a HiRDB system definition**

You can use the `pdconfchk` command to check the operands of a HiRDB system definition. This command checks the integrity of the specifications of the definition operands that are required to start HiRDB. We recommend that you execute the `pdconfchk` command after a HiRDB system definition is created (especially when an OS editor was used to create the HiRDB system definition).

## 6.5.4 System reconfiguration command (pdchgconf command)

With the exception of UAP environment definitions, normally you must stop HiRDB before you can change any HiRDB system definitions. By using the system reconfiguration command, you can change HiRDB system definitions while HiRDB is running. This allows you to perform the following operations while HiRDB is running:

- Adding, removing, and moving units
- Adding, removing, and moving servers
- Adding system files
- Adding, removing, and modifying system buffers

The system reconfiguration command is extremely useful for systems that are running continuously 24 hours a day. For details about how to use the system reconfiguration command, see the *HiRDB Version 9 System Operation Guide*.

Note that you must have HiRDB Advanced High Availability to use the system reconfiguration command.

# 6.6 HiRDB startup and termination

This section explains HiRDB startup and termination. To start HiRDB, you execute the `pdstart` command; to terminate HiRDB, you execute the `pdstop` command. In addition, there is a method (called *automatic startup*) that enables HiRDB to start automatically when the operating system starts, as well as a method for HiRDB parallel server configurations (called *reduced activation*) that enables operable units to start, even if there are units that cannot start.

## 6.6.1 Startup and termination modes

Starting HiRDB is governed by the *startup mode*; closing HiRDB is governed by the *termination mode*. This section explains the startup and termination modes. For details about the HiRDB startup and termination procedures, see the *HiRDB Version 9 System Operation Guide*.

### (1) Startup modes

The following table shows the HiRDB startup modes.

Table 6–7: HiRDB startup modes

| Startup mode | Explanation | Input command | | |
|---|---|---|---|---|
| | | Startup on a system basis | Startup on a unit basis#1 | Startup on a server basis#1 |
| Normal startup | This startup mode is used when the previous termination mode was normal termination. Information from the previous operation is not inherited, except for the following:<br>• Replica status of replica RDAREA (UNIX edition only)<br>• Statuses of RDAREAs that are shut down due to errors | `pdstart` | `pdstart -u`<br>`pdstart -x` | `pdstart -s` |
| Restart | This startup mode is used when the previous termination mode was one of those listed as follows; information from the previous operation is inherited:<br>• Planned termination<br>• Abnormal termination<br>• Forced termination<br>For details about the information that is inherited during a restart, see *Information inherited during a HiRDB restart* in the *HiRDB Version 9 System Operation Guide*. | | | N |
| Forced startup#2 | This startup mode is used to start HiRDB forcibly when it cannot be restarted. Information from the previous operation is not inherited during this process, so the database cannot be recovered; the HiRDB administrator must recover the database. | `pdstart`<br>`dbdestroy` | `pdstart -u`<br>`dbdestroy`<br>`pdstart -x`<br>`dbdestroy` | N |

N: Not usable (this startup method cannot be executed).

#1: In the case of a HiRDB parallel server configuration, startup and termination can be executed on a unit basis or on a server basis.

#2: When HiRDB is started forcibly, all RDAREAs (including system RDAREAs) that were updated since the last time HiRDB was started will have been destroyed. Thus, when forced startup is used, the destroyed RDAREAs must be recovered with the database recovery utility (`pdrstr`). If the RDAREAs are not recovered, correct operation of

HiRDB cannot be guaranteed. These RDAREAs can be recovered using the system log only. See the `KFPS01262-I` message that was output the previous time the `pdstart` command was used, and use as the input information to the database recover utility (`pdrstr`) the log file group name that was used at that time, which is shown in the message, as well as the system log that has been generated since then.

### (2) Termination modes

The following table shows the HiRDB termination modes.

Table 6–8: HiRDB termination modes

| Termination mode | Explanation | Input command | | |
|---|---|---|---|---|
| | | Termination on a system basis | Termination on a unit basis | Termination on a server basis |
| Normal termination | Prohibits CONNECT requests, and terminates HiRDB after all user processes are finished.<br>If you cannot stop HiRDB by executing the `pdstop` command because a utility is active, the `KFPS05074-E` message is output. If a unit cannot be stopped, the `KFPS05070-E` message is output. In these cases, the `pdstop` command ends with return code `8`. | `pdstop` | `pdstop -u`<br>`pdstop -x` | `pdstop -s` |
| Planned termination | Prohibits acceptance of transactions; terminates HiRDB after all users, including utilities, have disconnected. | `pdstop -P` | N | N |
| Forced termination | Terminates HiRDB immediately without waiting for completion of transactions being processed (these transactions become rollback targets# during a restart). | `pdstop -f` | `pdstop -f -u`<br>`pdstop -f -x` | N |
| Abnormal termination | Termination mode in which HiRDB is terminated by an error. HiRDB terminates immediately without waiting for completion of transactions being processed (these transactions become rollback targets# during a restart). | NA | NA | NA |

NA: Not applicable.

N: Not usable (this termination method cannot be executed).

#: Transactions that are being processed become rollback targets during a restart, except in the following cases:

- Database load utility (`pdload`) or database reorganization utility (`pdrorg`) is being executed in the no-log mode
- UAP is being executed in the no-log mode

After HiRDB has been restarted, the HiRDB administrator must either recover the RDAREAs from backup copies or re-execute the utility. For details about the no-log mode, see the *HiRDB Version 9 System Operation Guide*. For details about the procedure for recovering RDAREAs when the no-log mode is being used, see the *HiRDB Version 9 System Operation Guide*.

## 6.6.2 HiRDB automatic startup

HiRDB *automatic startup* means that HiRDB starts automatically when OS starts. *Manual startup* means that the `pdstart` command is entered after OS has started to start HiRDB. Which startup method is to be used is specified in the `pd_mode_conf` operand of the system common definition.

When automatic startup is specified, HiRDB (the unit) restarts automatically, even after HiRDB (the unit) has terminated abnormally. However, if three restart attempts in a row result in abnormal termination, there are no more attempts to restart the system automatically.

Reference note

- If HiRDB terminates abnormally during its startup or termination processing, the next startup will have to be by manual startup.
- A startup mode (forced startup, unit-basis startup, server-basis startup) that involves specification of `pdstart` command arguments (options) cannot be specified when startup is by automatic startup.

## 6.6.3 Reduced activation (applicable to HiRDB parallel server configurations only)

If an event such as an error prevents a unit from starting, you can use the reduced activation facility to start HiRDB using only the remaining normal units; this is called *reduced activation*. To enable reduced activation, specify the following operands:

- `pd_start_level`
- `pd_start_skip_unit`

For details about using reduced activation, see the *HiRDB Version 9 System Operation Guide.*

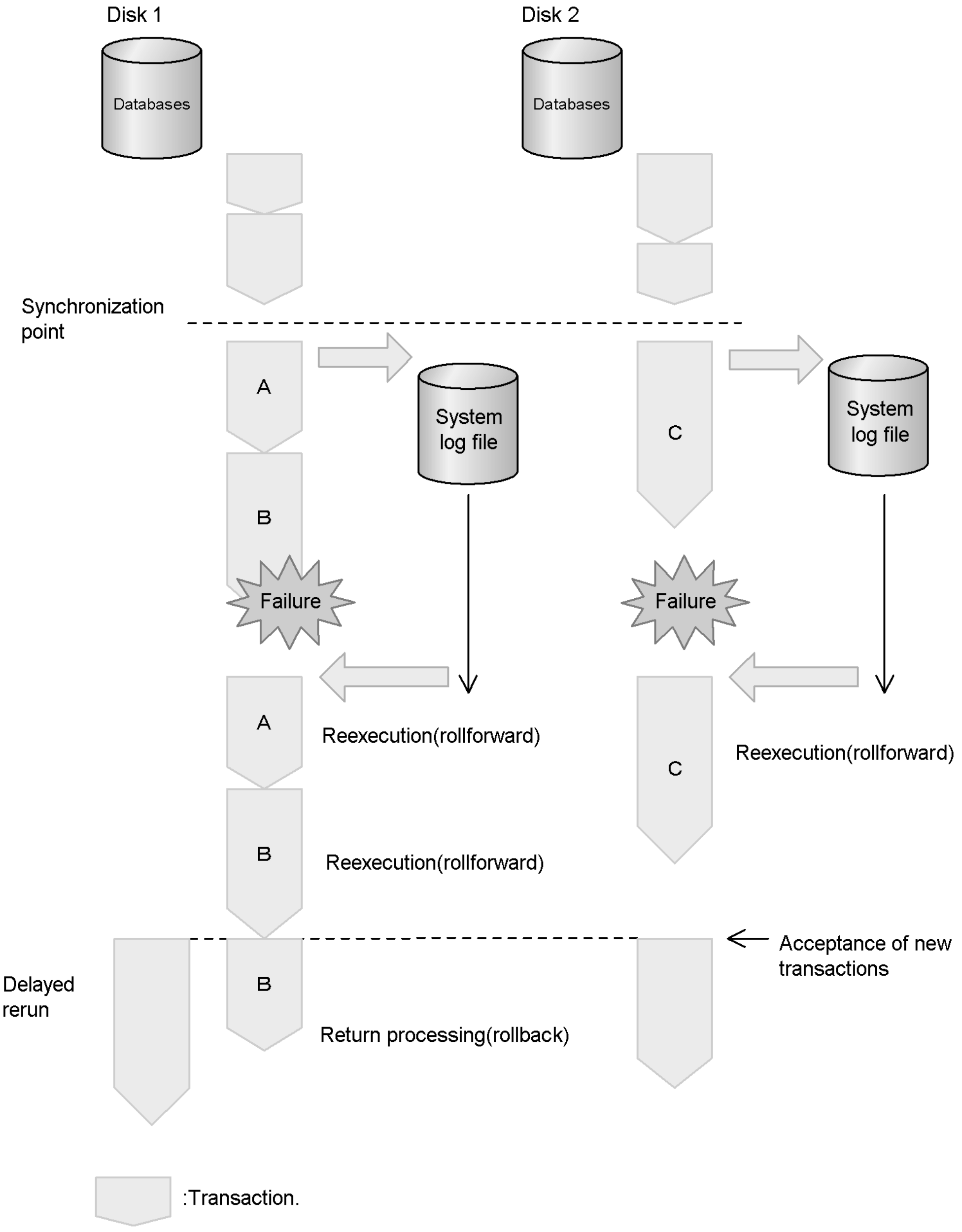
# 6.7 Delayed rerun

In HiRDB, a recovery process called *delayed rerun* reduces the downtime for transaction processing when a system failure occurs. When a failure occurs in the database, HiRDB uses the system log file to re-execute all update processing that occurred since the most recent synchronization point up to the time the failure occurred. During this processing, the transaction that was updating the database when the failure occurred is re-executed first (*rollforward*) to the point at which the failure occurred, and then the database is returned (*rollback*) to its status before the update processing was executed.

HiRDB executes rollback for the data on the disk that was being updated when the failure occurred and accepts new transactions for data that is not subject to the rollback. In this way, the system downtime for transaction processing is minimized.

The following figure shows the concept of delayed rerun.

Figure 6–9: Concept of delayed rerun



**Explanation**

A failure occurs during transaction B that accesses the database on disk 1; this failure occurs after completion of transaction C for disk 2. Consequently, transactions A, B, and C that executed since the previous synchronization point are rolled forward. Subsequently, transaction B, which was being executed when the failure occurred, is rolled back. Because the transactions for the database on disk 2 are not the targets of rollback, new transactions are being accepted.

# 6.8 Database access processing method

HiRDB uses global buffers to manage database input/output processes. This section explains the HiRDB database access processing method that uses global buffers. HiRDB provides the following facilities to improve the performance of the database access processing method:

- Global buffers
- In-memory data processing
- Prefetch facility
- Asynchronous READ facility
- Deferred write processing
- Facility for parallel writes in deferred write processing
- Incorporation during commit
- LRU management of global buffers
- Page access using the snapshot method
- Local buffers
- Global buffer pre-writing
- BLOB data file I/O facility
- Partial update or retrieval of BLOB or BINARY data
- Locator facility

## 6.8.1 Global buffers

A global buffer is a buffer that is used for input and output of data stored in the RDAREAs of a disk. Global buffers are allocated in the common memory. Buffers used for storing data for updating before the data in the database has been updated are called *update buffers*. Buffers that are used for referencing data or for storing data that has been updated in the database are called *reference buffers*.

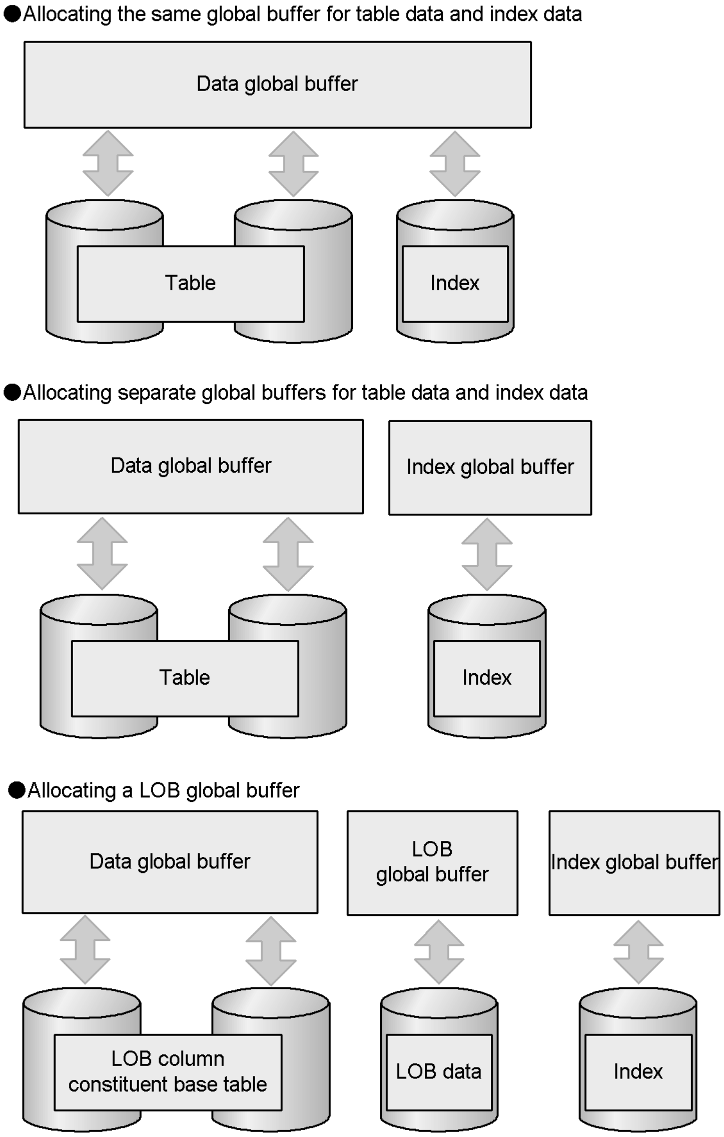
Global buffers are always allocated for RDAREAs and indexes that store data. The following table lists the types of global buffers.

Table 6–9: Types of global buffers

| Global buffer type | Explanation |
|---|---|
| Data global buffers | Global buffers used for input/output operations on table data. Data global buffers are allocated in units of RDAREAs. |
| Index global buffers | Global buffers used for input/output operations on index data. Index global buffers are allocated in units of indexes. |
| LOB global buffers | Global buffers used for input/output operations on LOB-attribute data. LOB global buffers are allocated in units of LOB RDAREAs. |

You can improve performance even more by using these global buffers in combination. For example, when you separately define data global buffers and index global buffers, data and index searches each operate independently, even if they are running at the same time. Consequently, the number of index inputs/outputs can be reduced even for a full-text search of a large amount of data, resulting in reduced processing time. The following figure shows the concept of global buffers.

Figure 6–10: Concept of global buffers


●Allocating the same global buffer for table data and index data
Data global buffer
Table
Index
●Allocating separate global buffers for table data and index data
Data global buffer
Index global buffer
Table
Index
●Allocating a LOB global buffer
Data global buffer
LOB global buffer
Index global buffer
LOB column constituent base table
LOB data
Index

## (1) Units for allocating global buffers

- Each RDAREA must have a data global buffer assigned to it. Multiple RDAREAs can be assigned to the same global buffer.
- Index global buffers can be allocated to indexes as necessary.
- LOB global buffers can be allocated to LOB RDAREAs as necessary.
- Global buffers can be allocated to system RDAREAs as necessary.
- For details about designing global buffers (and allocating global buffers to RDAREAs), see the *HiRDB Version 9 Installation and Design Guide*.

## (2) Global buffer allocation procedures

You use the `pdbuffer` operand to allocate global buffers. You can also allocate global buffers using the simple setup tool during environment setup for HiRDB. The following examples show the use of the `pdbuffer` operand to allocate global buffers:

**Examples**

```
pdbuffer -a gbuf01 -r RDAREA01,RDAREA02 -n 1000      1
pdbuffer -a gbuf01 -r LOBAREA01 -n 1000              2
pdbuffer -a gbuf01 -i USER01.INDX01 -n 1000          3
pdbuffer -a gbuf01 -b LOBAREA02 -n 1000              4
```

**Explanation**

1. Allocates a data global buffer to two RDAREAs (`RDAREA01` and `RDAREA02`).
2. Allocates a data global buffer to a LOB RDAREA (`LOBAREA01`).
3. Allocates an index global buffer to an index (`USER01.INDX01`).
4. Allocates a LOB global buffer to a LOB RDAREA (`LOBAREA01`).

For details about the `pdbuffer` operand, see the manual *HiRDB Version 9 System Definition*. For details about the simple setup tool, see the *HiRDB Version 9 Installation and Design Guide*.

### (3) Dynamic updating of global buffers

Because you must modify the `pdbuffer` operand to add, change, or delete global buffers, you normally need to stop HiRDB to perform this operation. However, if you install HiRDB Advanced High Availability, you can use the `pdbufmod` command to add, change, and delete global buffers while HiRDB is running. This is called *dynamic updating of global buffers*. As use examples, we recommend that you perform dynamic updating of global buffers in the following cases:

- Allocating global buffers for an RDAREA that was added
- Changing a global buffer allocated for an RDAREA
- Changing a global buffer definition as a result of global buffer tuning

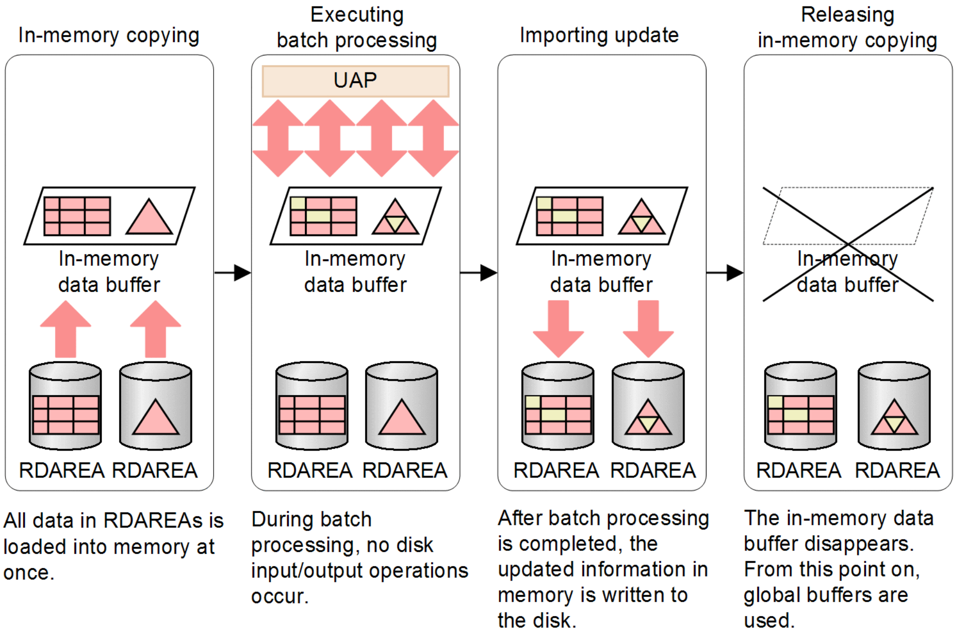
For details about dynamic updating of global buffers, see the *HiRDB Version 9 System Operation Guide*.

## 6.8.2 In-memory data processing

Applying in-memory data processing can accelerate batch-processing speed. In-memory data processing loads all data in RDAREAs into memory at one time, and updates only the data in memory while batch processing is being executed, without updating the data on the disk. When batch processing is completed, all of the updated data in memory is written onto the disk at once. During batch processing, no disk input/output operations occur. Disk input/output operations occur only during writing of the data in RDAREAs into memory, and during writing of the updated data in memory onto the disk.

The following figure provides an overview of in-memory data processing.

Figure 6–11: Overview of in-memory data processing


In-memory copying
Executing batch processing
Importing update
Releasing in-memory copying
UAP
In-memory data buffer
In-memory data buffer
In-memory data buffer
In-memory data buffer
RDAREA RDAREA
RDAREA RDAREA
RDAREA RDAREA
RDAREA RDAREA
All data in RDAREAs is loaded into memory at once.
During batch processing, no disk input/output operations occur.
After batch processing is completed, the updated information in memory is written to the disk.
The in-memory data buffer disappears. From this point on, global buffers are used.

**Explanation**

The operation of writing all data in RDAREAs into memory at a time is called *in-memory copying*. In-memory copying is performed for each RDAREA. An in-memory copied RDAREA is called an *in-memory RDAREA*. Furthermore, a data buffer used during in-memory data processing is called an *in-memory data buffer*.

Tip

To execute in-memory data processing, use an in-memory data buffer instead of a global buffer. Because an in-memory data buffer is dynamically allocated in shared memory by HiRDB, there is no need for the HiRDB administrator to define one.

During batch processing, only the data in the in-memory data buffer is updated; the data on the disk is not updated. After batch processing is completed, you can execute a command (the `pdhold` command) to write all updated information in the in-memory data buffer onto the disk at once. No writing of data onto the disk occurs at synchronization points.

To apply in-memory data processing, you need HiRDB Accelerator.

For details about how to apply in-memory data processing, see the manual *HiRDB Version 9 Batch Job Accelerator*.

## (1) Advantages over global buffers

When global buffers are used, disk input/output operations occur regularly. By contrast, when in-memory data buffers are used, disk input/output operations occur only during in-memory copying and during writing of data onto the disk. No writing of data onto the disk occurs at synchronization points either. Therefore, by using in-memory data buffers, you can reduce the number of disk input/output operations compared to when global buffers are used.

Additionally, in-memory data buffers provide the following advantages over global buffers:

- Little buffer management processing is needed.
- No tuning is required.

For details, see the manual *HiRDB Version 9 Batch Job Accelerator*.

## (2) Application criteria for in-memory data processing

Applying in-memory data processing can provide benefits in the following cases:

- When batch processing with a long processing time is being executed

Applying in-memory data processing to batch processing that updates a large volume of data and has a long processing time can provide benefits. Applying the no-log mode can shorten the processing time further.

The longer batch processing takes, the greater the difference in the number of disk input/output operations compared to when in-memory data processing is not used; therefore, applying it can be expected to provide benefits.

- When multiple batch processes are being executed in succession

  Applying in-memory data processing during successive execution of multiple batch processes can be expected to provide benefits.

  The greater the number of batch processes executed in succession, the greater the difference in the number of disk input/output operations compared to when in-memory data processing is not used. Therefore, applying it can be expected to provide benefits.

- When performance does not improve because of locked global buffers

  When performance does not improve because global buffers are locked, applying in-memory data processing can be expected to provide benefits.

  When a global buffer is used (when in-memory data processing is not used), a process of checking whether data is cached in the global buffer (*cache search processing*) is performed whenever a referencing or updating process occurs. During this process, because all pages of the global buffer become locked, other referencing or updating processes must wait for lock-release. By contrast, when in-memory data processing is used, all data is cached in the in-memory data buffer, and therefore no cache search processing occurs. Because locking of all pages of the global buffer is eliminated, improvements in concurrent executability can be expected, along with comparable improvements in performance. For example, when locking is occurring frequently in batch processing that updates multiple RDAREAs to which the same global buffer is allocated, applying in-memory data processing can be expected to improve performance.

  You can check the locking frequency in the statistical information related to global buffers of the statistics analysis utility.

- When moving data from a mainframe system

  When moving a large amount of data from a mainframe system to HiRDB (loading the data into a HiRDB database), applying in-memory data processing can be expected to provide benefits.

## 6.8.3 Prefetch facility

Table data on disk normally is read from a global buffer (or a local buffer) one page at a time. However, this data can also be read in batches of multiple pages, rather than one page at a time. This capability is provided by the *prefetch facility*. By using the prefetch facility, you can reduce the number of I/O operations between the table data on disk and the global buffer (or local buffer).

### (1) Application criteria for the prefetch facility

The prefetch facility is available when the following conditions are satisfied:

1. The accessed data is in a HiRDB file system area for which the raw I/O facility is used (or in a character special file, if you are using the UNIX edition).
2. A large data set is being manipulated.
3. One of the following SQL statements is being executed:

- A `SELECT`, `UPDATE`, or `DELETE` statement that does not use an index
- A `SELECT`, `UPDATE`, or `DELETE` statement that performs an ascending search# using an index or cluster key (except for = and `IN` conditions)

#: For multi-column indexes, searching is performed in the order specified by the index definitions.

### (2) Specifying the prefetch facility

**For a global buffer**

The prefetch facility is enabled by specifying a value of at least `1` for the `-m` option of the `pdbuffer` operand. The specification in the `-p` option of the `pdbuffer` operand is of the number of pages to be fetched at a time. For details about the `pdbuffer` operand, see the manual *HiRDB Version 9 System Definition*.

**For a local buffer**

The prefetch facility is enabled by specifying the number of pages to be fetched at a time in the `-p` option of the `pdlbuffer` operand. For details about the `pdbuffer` operand, see the manual *HiRDB Version 9 System Definition*.

## 6.8.4 Asynchronous READ facility

When you are using the prefetch facility to input multiple pages at a time into a global buffer, pages are synchronously pre-fetched from the server process and input to the prefetch buffer. When the prefetch facility is being used, the *asynchronous READ facility* sets up two prefetch buffers, and while the DB process is using one of the buffers, a READ process pre-fetches pages from the other buffer asynchronously with the DB process. By executing the DB processing and the prefetch input at the same time, you can reduce the processing time. Moreover, with a HiRDB parallel server configuration, by alternating the threads for the I/O wait time, you can reduce the I/O wait time.

Note, however, that you cannot use the asynchronous READ facility with a local buffer. Nor can you use it on an RDAREA that has a `SCHEDULE` attribute. Use the prefetch facility in these cases.

**Enabling the asynchronous READ facility**

You specify the number of asynchronous READ processes using the `pd_max_ard_process` operand. If you specify `0` (or do not specify anything) in this operand, the asynchronous READ facility remains inactive. If it is active, you must also specify the prefetch facility (by specifying a value of `1` or greater in the `-m` option of the `pdbuffer` operand).

## 6.8.5 Deferred write processing

*Deferred write processing* is a process that writes pages that have been updated in the global buffer onto the disk whenever a particular number of pages has been updated (rather than when a `COMMIT` statement is issued). The point in time at which the number of updated pages has reached the particular value (which is determined by HiRDB) is called the *deferred write trigger*.

The number of updated pages to be written onto the disk is determined by HiRDB on the basis on the updated output page rate at the deferred write trigger that is specified in the `-w` option of the `pdbuffer` operand. When deferred write processing is in effect, data is not written onto the disk even when a `COMMIT` statement is issued, thus reducing the input/output processing load.

For details about using deferred write processing, see the *HiRDB Version 9 Installation and Design Guide*.

**Setting up deferred write processing**

Deferred write processing is used by specifying both the `pd_dbsync_point` and the `pdbuffer` operands of the system common definition. For details about the `pd_dbsync_point` and `pdbuffer` operands, see the manual *HiRDB Version 9 System Definition*.

## 6.8.6 Facility for parallel writes in deferred write processing

The facility for parallel writes in deferred write processing is a function that enables multiple processes to execute deferred write processing. Because multiple processes perform write processing, the facility reduces the amount of the time required to write to disk. For details about the facility, see the *HiRDB Version 9 Installation and Design Guide*.

**Specifying the facility for parallel writes in deferred write processing**

Specify in the `pd_dfw_awt_process` operand the number of parallel WRITE processes for deferred write processing that can perform write processing. Additionally, specify in the `pd_dbbuff_rate_updpage` operand the request rate of the deferred write trigger. If the pd_dfw_awt_process operand is not specified, the facility for parallel writes in deferred write processing does not become effective.

**Considerations about using the facility**

Use of the facility for parallel writes in deferred write processing increases CPU usage, because of the increase in the number of processes.

## 6.8.7 Incorporation during commit

Normally, pages that have been updated in the global buffer are written onto the disk whenever a COMMIT statement is issued. This is called *incorporation during commit*. When incorporation during commit is in effect, there is no need to recover the database from a synchronization point during full recovery processing of the system, which reduces the amount of time required for full recovery processing. For details about using incorporation during commit, see the *HiRDB Version 9 Installation and Design Guide*.

**Setting up incorporation during commit**

Incorporation during commit is used by specifying commit in the pd_dbsync_point operand. For details about the pd_dbsync_point operand, see the manual *HiRDB Version 9 System Definition*.

## 6.8.8 LRU management method for global buffers

The LRU management method for global buffers that is appropriate to the types of job (online or batch) can be selected. The two LRU management methods are described as follows.

### (1) Management of reference buffers and update buffers in independent LRUs

This method manages reference buffers and update buffers in independent LRUs. If a shortage of global buffers occurs, the reference buffer in the global buffer that was accessed least recently is purged from the memory. If the number of references and updates per transaction is relatively small, as in the case of online jobs, managing reference buffers and update buffers in independent LRUs improves processing performance.

To apply LRU management to global buffers, SEPARATE must be specified in the pd_dbbuff_lru_option operand. For details about the pd_dbbuff_lru_option operand, see the manual *HiRDB Version 9 System Definition*.

### (2) Management of global buffers in a single LRU

This method manages all global buffers in a single LRU. If a shortage of global buffers occurs, the buffer in the global buffer pool that was accessed least recently is purged from the memory. If a large number of retrievals and updates coexist, as in a combination of online jobs and batch jobs, managing the reference buffers in a single LRU improves processing performance.

One of the following actions must be taken to manage global buffers in a single LRU:

- Specify MIX (the default) in the pd_dbbuff_lru_option operand.
- Specify the updated output page rate at the deferred write trigger in the -w option of the pdbuffer operand.

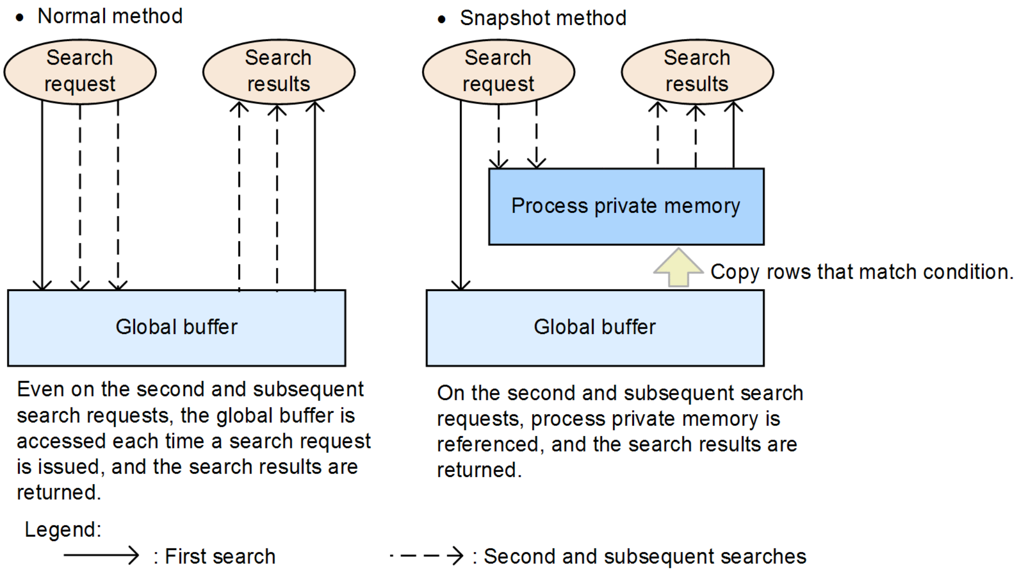
For details about the pd_dbbuff_lru_option and pdbuffer operands, see the manual *HiRDB Version 9 System Definition*.

## 6.8.9 Accessing pages using the snapshot method

When a search that cannot use a performance-enhancing facility (such as the rapid grouping facility) is being performed, global buffers are accessed roughly the same number of times as the number of rows there are that match the condition. With the *snapshot method*, the first time data is accessed, all rows in the buffer that match the search condition are copied to process private memory. The second and subsequent times that the same page is accessed, process private memory is referenced, and the search results are returned. This enables search times to be reduced after the first access. It also reduces the number of times that a global buffer is accessed, and avoids concentrating accesses to the same buffer.

The following figure provides an overview of the snapshot method.

Figure 6–12: Overview of the snapshot method


• Normal method
Search request
Search results
Global buffer
Even on the second and subsequent search requests, the global buffer is accessed each time a search request is issued, and the search results are returned.
• Snapshot method
Search request
Search results
Process private memory
Copy rows that match condition.
Global buffer
On the second and subsequent search requests, process private memory is referenced, and the search results are returned.
Legend:
: First search
: Second and subsequent searches

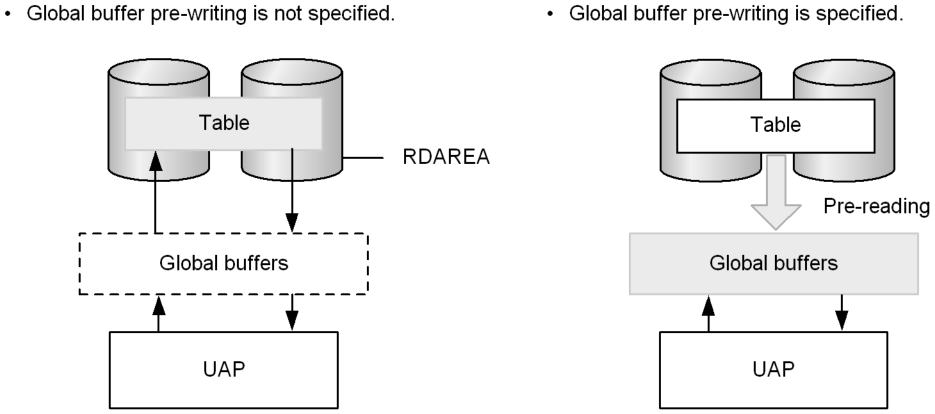
**Specification method**

Specify `SNAPSHOT` (default value) in the `pd_pageaccess_mode` operand.

## 6.8.10 Global buffer pre-writing

Global buffer pre-writing is a function that loads in advance data from a specified table or index into global buffers. The following figure provides an overview of global buffer pre-writing.

Figure 6–13: Overview of global buffer pre-writing


• Global buffer pre-writing is not specified.
Table
RDAREA
Global buffers
UAP
• Global buffer pre-writing is specified.
Table
Pre-reading
Global buffers
UAP

**Explanation**

- Global buffer pre-writing is not specified

  When a UAP accesses HiRDB immediately after HiRDB starts, data must be read from the table (a physical I/O occurs) because the global buffers do not hold any data. When data in this table is accessed later, pages from this table that have been written into global buffers are not read from the table. However, accesses to data on other pages must still be read from the table.

- Global buffer pre-writing is specified

  Table data is written into global buffers in advance, so the table can be accessed without data having to be read from the table (no physical I/O occurs). Subsequent accesses to this table do not require that data be read from the table.

### (1) Advantages of global buffer pre-writing

Because reading of table and index data from specified tables is performed in advance, the global buffer hit ratio is improved. By specifying global buffer pre-writing before online operations or other activities are started of tables or indexes that you believe will be accessed frequently, you can expect to achieve a higher buffer hit ratio.

### (2) Execution method

To enable global buffer pre-writing, specify the tables or indexes you wish to have pre-written into the global buffers, and execute the global buffer residence utility (`pdpgbfon`).

### (3) Considerations when using

Consider the following points about using global buffer pre-writing:

- You must have more global buffers than the number of pages stored in the tables or indexes that are to be pre-read.
- If you do not have enough global buffers, older page information will be forced out of the global buffers according to the LRU management method (the oldest page in the accessed global buffers is forced out as specified by the value in the `pd_dbbuff_lru_option` system definition operand). This means that executing `pdpgbfon` is pointless if you do not have enough global buffers.
- When you specify pre-writing with the global buffer residence utility (`pdpgbfon`), the pre-fetch facility becomes available. This makes it possible to reduce execution time by specifying the prefetch facility when you define global buffers.

## 6.8.11 Local buffers

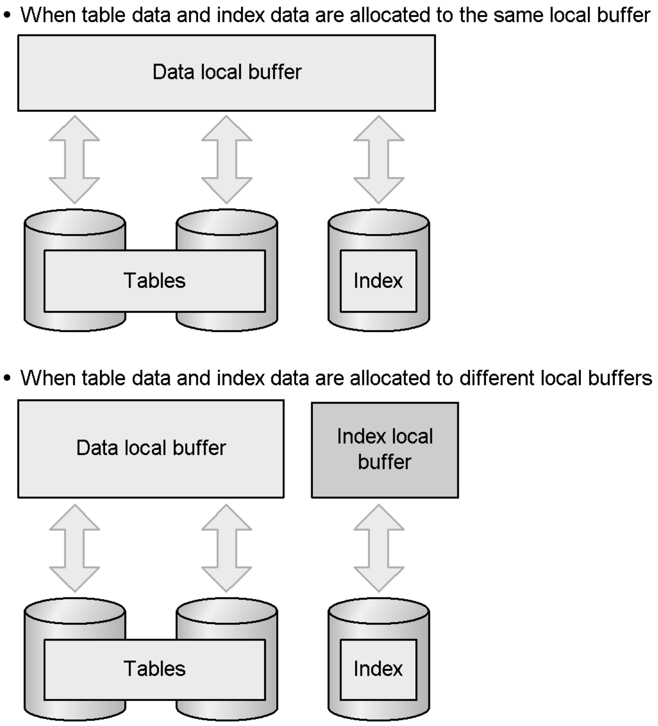
A local buffer is a buffer that is used for input and output of data stored in RDAREAs on disk, and is allocated in process private memory. The following table lists and describes the types of local buffers available.

Table 6–10: Types of local buffers

| Local buffer type | Description |
|---|---|
| Data local buffer | Local buffers used for table data I/O. Data local buffers are allocated on a per-RDAREA basis. |
| Index local buffer | Local buffers used for index data I/O. Index local buffers are allocated on a per-index basis. |

You can use these local buffers to improve performance. For example, by separately defining data local buffers and index local buffers, data and index searches each operate independently, even if they are running at the same time. This enables the number of index I/O operations to be reduced even for a full text search of a large amount of data, resulting in reduced processing time. The following figure provides an overview of local buffers.

Figure 6–14: Overview of local buffers



- When table data and index data are allocated to the same local buffer
- When table data and index data are allocated to different local buffers

### (1) Local buffer application criteria

Defining local buffers is beneficial if all of the following conditions are satisfied:

- A large amount of data is being searched or updated.
- The RDAREA to be accessed cannot be accessed from another UAP.

However, do not define local buffers for a UAP that establishes an emergency connection to HiRDB, due to the significant adverse effects it has on the system (memory resource strain, process monopolization, and so on.).

### (2) Local buffer allocation procedure

Use the `pdlbuffer` operand to allocate local buffers. The following example shows how to use the `pdlbuffer` operand to allocate local buffers:

Example:

```
pdlbuffer -a localbuf01 -r RDAREA01,RDAREA02 -n 1000  . . .1
pdlbuffer -a localbuf02 -i USER01.INDX01 -n 1000  . . .2
```

**Explanation**

1. Allocates data local buffers to two RDAREAs (`RDAREA01` and `RDAREA02`).
2. Allocates an index local buffer to an index (`USER01.INDX01`).

For details about the `pdlbuffer` operand, see the manual *HiRDB Version 9 System Definition*.

## 6.8.12 BLOB data file output facility

The following issues arise if BLOB data stored in a HiRDB server is to be retrieved by a client:

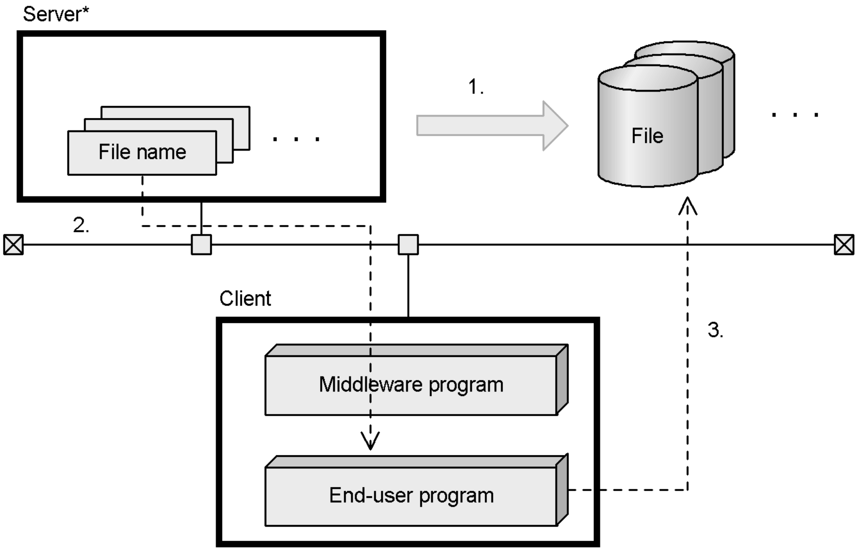
- A memory area for storage of BLOB data must be provided at the client.

- The server requires memory for a send buffer for returning BLOB data and a receive buffer to enable the client to receive BLOB data.
- Allocating large-object memory space for BLOB data reduces the available memory resources.
- If the end-user's software includes a middleware program that operates as a HiRDB client, the processes of sending and receiving BLOB data can further increase memory requirements.

To mitigate such increases in the memory requirements associated with BLOB data retrievals, you can provide settings so that, instead of retrieved BLOB data being returned to the client, it is output to a unit in which either a single server or a front-end server is running. This means that only the server IP address and the file name need to be returned to the client. This is called the *BLOB data file output facility*.

The following figure provides an overview of the BLOB data file output facility.

Figure 6–15: Overview of the BLOB data file output facility



* Machine at which either a single server or a front-end server is running

**Explanation**

1. When the BLOB data is retrieved from the client, it is output to the file row by row, and column by column.
2. The name of the file to which the BLOB data is output in (1) is returned to the client.
3. Based on the returned file name, access is made to the BLOB data file located at the server.

## (1) Application criteria

Use this technique to reduce memory requirements associated with BLOB data retrievals. This technique reduces the memory required for the client program and for inter-client communication buffers, at the expense of increased disk input/output time and capacity at the server. Therefore, if this technique is to be used, the tradeoff between memory requirements and disk input/output time must be evaluated.

## (2) Specification

Use of the BLOB data file output facility is specified in a `WRITE` specification of the SQL. A `WRITE` specification can be specified in either a cursor specification or a query specification. For details about `WRITE` specifications, see the manual *HiRDB Version 9 SQL Reference*.

The client can obtain as the SQL retrieval results only the IP address of the server, the BLOB data storage location that is set in the SQL, and the file name of the BLOB data. With this information, the client is able to identify the particular BLOB data stored at the server.

## (3) Notes on using the BLOB data file output facility

- When BLOB data files are no longer needed, the user must delete them. Note the following point with regard to deleting a file (a file can be deleted unconditionally once the cursor has been closed or the transaction resolved):
    - If the file is deleted immediately after a `FETCH` and the BLOB data is the same as the result of the `FETCH` before the same cursor is searched, it might not be possible to re-create the file under the same file name. Because of this, make note of the file's original file name and delete it after its name has been changed.
- The occurrence of an error or of rollback will not result in deletion of a BLOB data file that has been created. Files that are left undeleted occupy OS resources, such as disk space.
- When you use either of the following facilities, make sure that the disk has adequate free space:

  `FETCH` facility with arrays

  One execution of `FETCH` generates as many files as there are array elements.

  Block transfer facility

  The first execution of `FETCH` creates as many files as the number of rows involved in a block transfer. Subsequently, after the same number of `FETCH` operations as the number of block transfer rows, as many files as the number of block transfer rows are created each time `FETCH` is executed, and this process is repeated.
- If the file name is the same as other transactions or cursor searches, there is a potential for files to be overwritten or damaged. In such a case, for each transaction or cursor, change the directory name or file name in the file prefix to avoid duplicate names.

## (4) Examples of using the BLOB data file output facility

Shown as follows are examples of retrievals using the BLOB data file output facility.

### (a) Retrieval from BLOB columns

Retrieve columns `C1` and `C2` from table `T1`. The system outputs the BLOB data from `C1` to a file, and obtains the file name. The following figure shows an example of this retrieval using the BLOB data file output facility (retrieval of BLOB columns).

Figure 6–16: Example of a retrieval using the BLOB data file output facility (retrieval of BLOB columns)

Table `T1`

| C1 | C2 |
|---|---:|
| *BLOB-value-1* | 10 |
| *BLOB-value-2* | 20 |
| *BLOB-value-3* | 30 |
| *BLOB-value-4* | 40 |

SQL statement

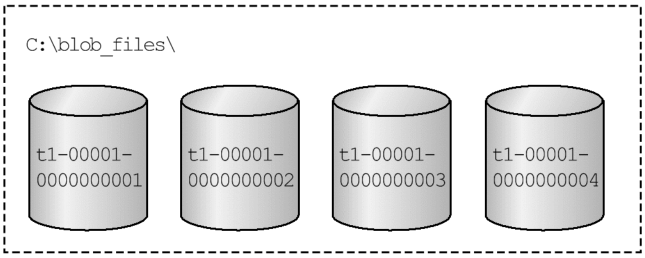
```
SELECT WRITE(C1,'C:\blob_files\t1',0),C2 FROM T1
```

Values returned as retrieval results

| C1 | C2 |
|---|---:|
| 172.16.nn.nn:C:\blob_files\t1-00001-0000000001 | 10 |
| 172.16.nn.nn:C:\blob_files\t1-00001-0000000002 | 20 |
| 172.16.nn.nn:C:\blob_files\t1-00001-0000000003 | 30 |
| 172.16.nn.nn:C:\blob_files\t1-00001-0000000004 | 40 |

BLOB data output to the server

- IP address: `172.16.`*`nn`*`.`*`nn`*



### (b) Retrieval of an abstract data type with the BLOB attribute

From table `T2`, retrieve column `ADT1` that makes the `CONTAINS` function `TRUE`. In this case, the BLOB value resulting from passing the column value to an argument of the `EXTRACTS` function is output to a file, and the file name is obtained. In this example, all entries are retrieved. The following figure shows an example of this retrieval using the BLOB data file output facility (retrieval of an abstract data type with the BLOB attribute).

Figure 6–17: Example of a retrieval using the BLOB data file output facility (retrieval of an abstract data type with the BLOB attribute)

Table T2

| ADT1 |
|---|
| *Abstract-data-type-value-1* |
| *Abstract-data-type-value-2* |
| *Abstract-data-type-value-3* |
| *Abstract-data-type-value-4* |

SQL statement

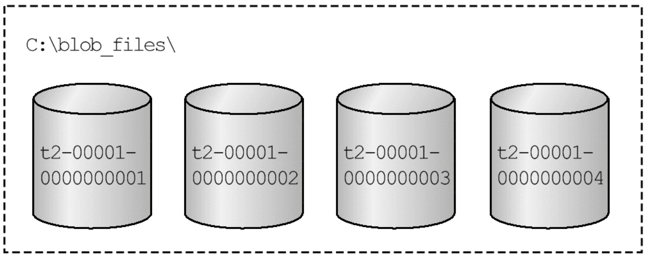
```
SELECT WRITE(EXTRACTS(ADT1,…),'C:\blob_files\t2',0) FROM T2
   WHERE CONTAINS(ADT1,…) IS TRUE
```

Values returned as retrieval results

| ADT1 |
|---|
| 172.16.nn.nn:C:\blob_files\t2-00001-0000000001 |
| 172.16.nn.nn:C:\blob_files\t2-00001-0000000002 |
| 172.16.nn.nn:C:\blob_files\t2-00001-0000000003 |
| 172.16.nn.nn:C:\blob_files\t2-00001-0000000004 |

BLOB data output to the server

- IP address: 172.16.*nn*.*nn*



## 6.8.13 Partial update or retrieval of BLOB or BINARY data

### (1) Explanation of partial update or retrieval of BLOB or BINARY data

When BLOB or BINARY data is updated after new data has been added to a registered BLOB or BINARY data item, or when an entire BLOB or BINARY data item is retrieved as the result of BLOB or BINARY data being searched, both the server and the client must allocate a large amount of memory for the BLOB or BINARY data, putting a strain on memory resources.

When you partially update or retrieve BLOB or BINARY data, you can avoid straining memory resources, because you need to allocate only the memory required for the actual amount of BLOB or BINARY data being added or extracted.

However, for BINARY data, the definition length must be 32,001 bytes or greater.

**BLOB or BINARY data addition update:**

By specifying a concatenation operation in the SET clause of the UPDATE statement, you can add new data to BLOB or BINARY data that has been registered.

**BLOB and BINARY data partial extraction:**

By specifying the SUBSTR scalar function, you can extract only a specified portion of the BLOB or BINARY data.

**BLOB or BINARY data back-end deletion update:**

By specifying the SUBSTR scalar function in which the constant 1 is specified for the processing target column and start position in the SET clause of the UPDATE statement, you can delete only the back-end portion of the BLOB or BINARY data.

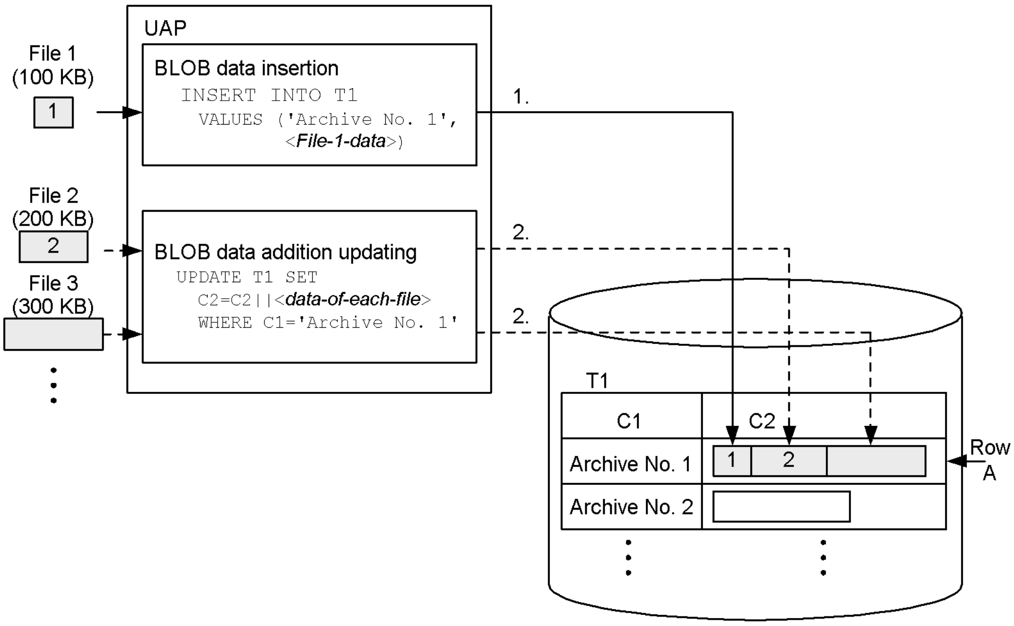
For details about partial update or retrieval of BLOB or BINARY data, see the *HiRDB Version 9 UAP Development Guide*.

## (2) Examples of partial update or retrieval of BLOB or BINARY data

### (a) BLOB data addition update

A single BLOB data item is stored across a number of files. The following figure shows an example of BLOB data addition updating.

Figure 6–18: Example of BLOB data addition updating



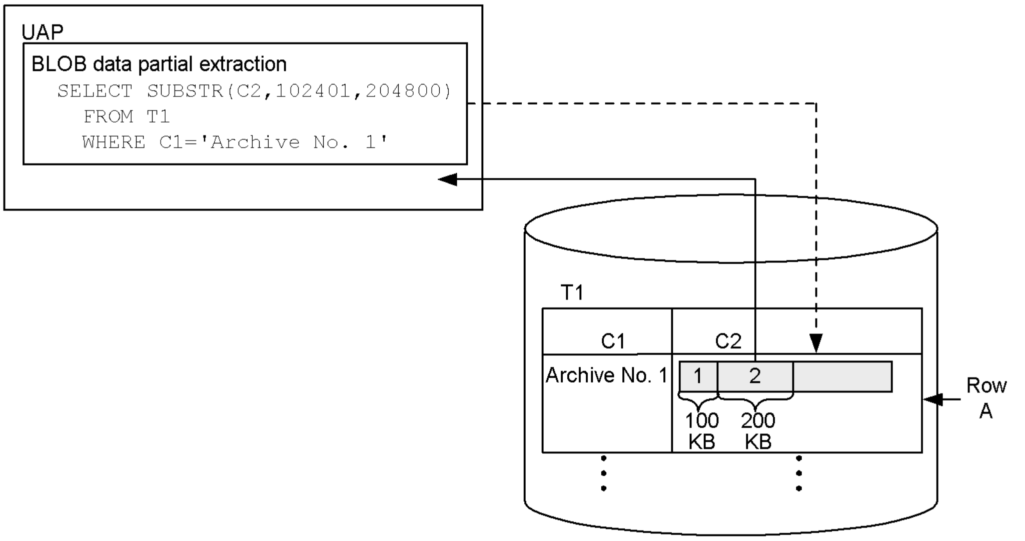
**Explanation**

1. BLOB data from file 1 is inserted into column C2 of row A in table T1.
2. Addition updating is performed by concatenating BLOB data of file 2 to column 2 of row A. Subsequent data additions are also performed in a similar manner.

### (b) BLOB data partial extraction

An area in file 2 is extracted from the BLOB data in column C2 of row A that was stored using BLOB data addition updating. The following figure shows an example of BLOB data partial extraction.

Figure 6–19: Example of BLOB data partial extraction



UAP

BLOB data partial extraction

```
SELECT SUBSTR(C2,102401,204800)
  FROM T1
  WHERE C1='Archive No. 1'
```

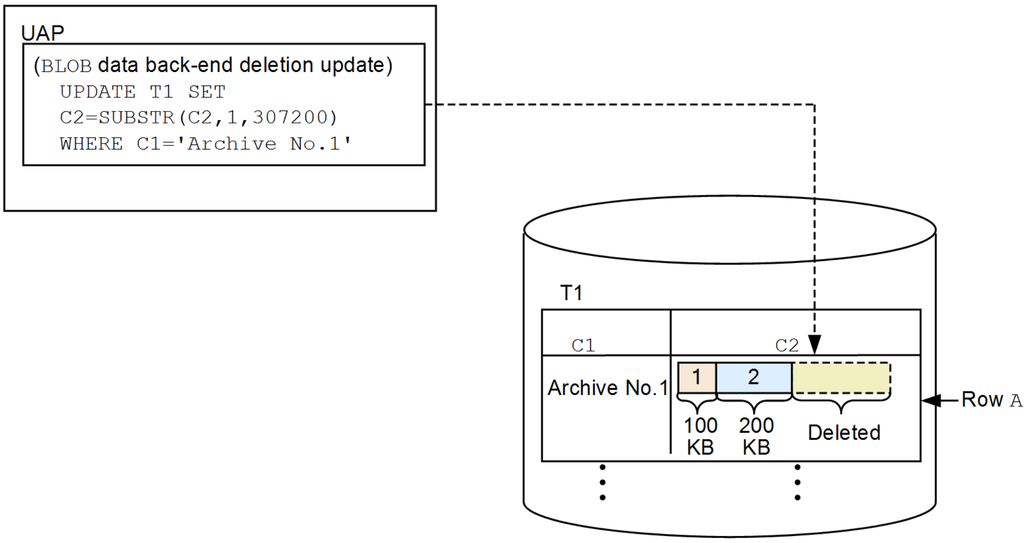
**Explanation**

Using the SUBSTR scalar function, from column 2 of row A, the length of the data from the data column of file 2 (200 × 1,024 = 204,800 bytes) is extracted beginning at the start position of the data column of file 2 (at the 100 × 1,024 + 1$^{th}$ = 102,401$^{st}$ byte position).

### (c) BLOB data back-end deletion update

The back-end portion is deleted from the BLOB data in column C2 of row A that was stored using BLOB data addition updating, leaving only files 1 and 2. The following figure shows an example of BLOB data back-end deletion update.

Figure 6–20: Example of BLOB data back-end deletion update



UAP

(BLOB data back-end deletion update)

```
UPDATE T1 SET
C2=SUBSTR(C2,1,307200)
WHERE C1='Archive No.1'
```

**Explanation**

Using the SUBSTR scalar function, only the length of data from the first byte beginning at the start position of the data column of file 1 through files 1 and 2 (100 × 1,024 + 200 × 1,024 = 307,200 bytes) is replaced with the extracted data and updated. This leaves only files 1 and 2 and deletes the data at the back end.

## 6.8.14 Locator facility

### (1) Overview of the locator facility

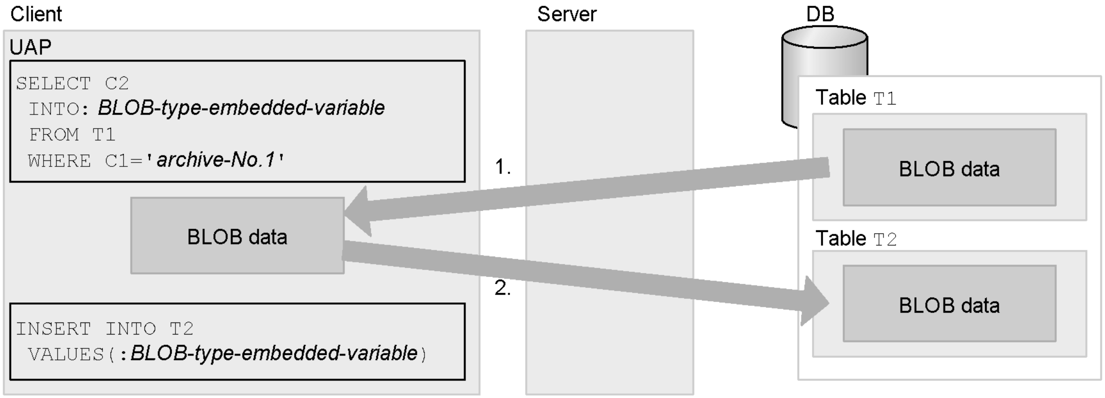
If a retrieved BLOB or BINARY data item is received by a client UAP as a BLOB-type or BINARY-type embedded variable, a memory area sufficient to store the received data must be allocated on the client side. This might strain the memory resources on the client side when large amounts of data are being retrieved. A large amount of data must also be transferred from the server to the client. However, if only a portion of the data is needed, or if the received data will be specified without change in another SQL statement and then returned to the server, transferring the data to the client is a waste of resources.

The locator facility is designed to overcome this problem. A locator is a four-byte value that identifies data on the server. By specifying a locator embedded variable in a single-row `SELECT` statement or in the `INTO` clause of the `FETCH` statement, you can retrieve in the search results only the locator value that identifies the data rather than the entire data item. You can also specify the locator embedded variable that identifies such data in another SQL statement, which enables you to process the data identified by the locator.

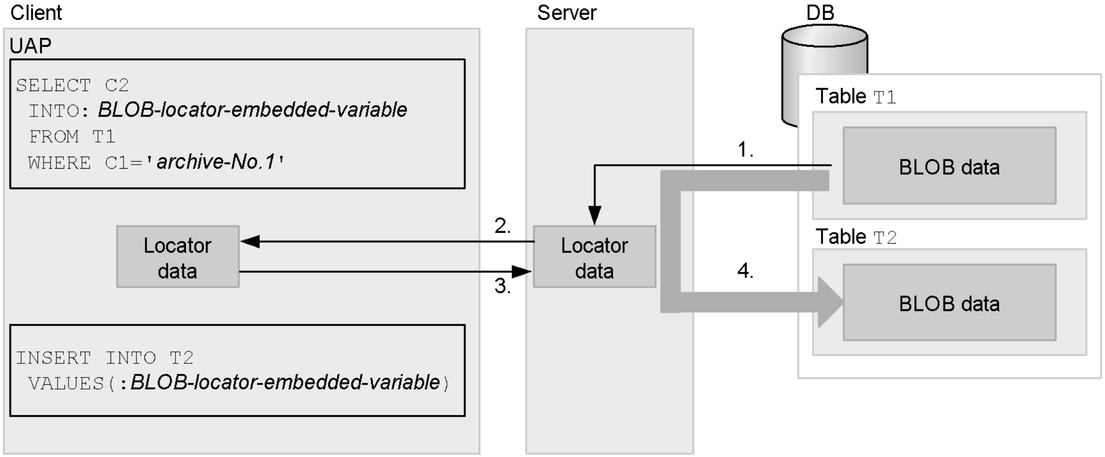
The following figure provides an overview of the locator facility.

Figure 6–21: Overview of the locator facility

- Locator facility is not used



- Locator facility is used



**Explanation**

Locator facility is not used:

1. The BLOB data retrieved from the database is transferred from the server to the client.
2. The BLOB data is transferred from the client to the server, and then stored in the database.

Locator facility is used:

1. The server creates a locator data item that identifies the data retrieved from the database.
2. The locator data item is transferred from the server to the client.
3. The locator data item is transferred from the client to the server.
4. The BLOB data on the server that was identified by the locator data item is stored in the database.

## (2) Application criteria

The locator facility is useful whenever you retrieve BLOB or BINARY data.

By using the locator facility, the client need only allocate enough memory to store the size of the actual data being used. The amount of data transferred between the server and the client is also reduced because a locator is used.

## (3) Advantages

Using the locator facility reduces the amount of memory required on the client side. It also reduces the amount of data transferred between the server and the client.

## (4) How to use

To receive the value of a locator, replace in the SQL statement the embedded variable that receives the BLOB-type or BINARY-type data with the applicable locator-type embedded variable. To process the data allocated in the locator, specify the applicable locator-type embedded variable in the SQL statement, rather than specifying a BLOB-type or BINARY-type embedded variable.

For details about the locator facility, see the *HiRDB Version 9 UAP Development Guide*.

# 6.9 Transaction control

This section explains the *transaction control* that is performed when a HiRDB database is accessed from a UAP. A *transaction* is a logical unit of work. In the case of HiRDB, transaction control has the following meaning:

**Transaction control unique to HiRDB**

This encompasses transactions that are within the range of HiRDB specifications, and refers to processing in which either a `COMMIT` or a `ROLLBACK` statement is issued by an SQL statement to determine whether an updating of table data is to be put into effect or is to be invalidated.

**Transaction control based on XA interface**

This applies to a transaction that is executed by UAP processing when linkage to OLTP is established using the XA interface; it refers to the process that determines whether the UAP processing under OLTP using an XA interface function (the `tx_commit` or `tx_rollback` function) is to be put into effect or is to be invalidated.

## 6.9.1 Connection to and disconnection from HiRDB

**Connection to the HiRDB system**

Becoming able to access a HiRDB database by means of a UAP is called *connection to HiRDB*. Connection must be made to the HiRDB system before a transaction can be started. Connection to HiRDB is defined with the `CONNECT` control SQL statement.

**Disconnection from the HiRDB system**

Terminating access to the HiRDB database by the UAP and disconnecting from the HiRDB is called *disconnection from HiRDB*. When disconnection from HiRDB occurs, transactions are terminated and a synchronization point is set. Disconnection from HiRDB is defined with the `DISCONNECT` control SQL statement.

## 6.9.2 Multi-connection facility

In HiRDB, a UAP of a single HiRDB client can connect simultaneously to multiple HiRDB servers. This is called the *multi-connection facility*. The multi-connection facility enables a single UAP at a HiRDB client to make multiple connections to a single HiRDB server. A single UAP can also connect to multiple HiRDB servers. Each of the multiple connections is treated as an independent transaction (that is, it is handled by the HiRDB server as though each connection were made from a separate UAP). Because a single UAP can make multiple connections, the number of UAPs to be executed can be reduced, thus reducing the overall memory requirements of the UAPs.

To use the multi-connection facility, a dedicated library that is provided must be linked. For details about the multi-connection facility, see the *HiRDB Version 9 UAP Development Guide*.

**Connection modes that can use the multi-connection facility**

The multi-connection facility can be used under transaction control unique to HiRDB, as well as under transaction control based on the XA interface. However, the multi-connection facility used in linkage to OLTP by means of the XA interface requires different items to be specified when HiRDB is registered into the transaction manager. For details on the registration procedure, see the *HiRDB Version 9 Installation and Design Guide*.

**Setting up the multi-connection facility**

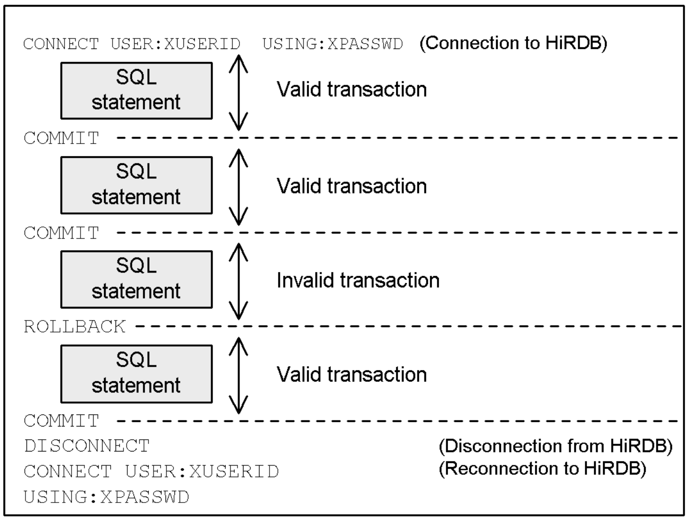
The multi-connection facility is used by defining the following SQLs in the UAP definition:

- Allocation of a connection handle (`ALLOCATE CONNECTION HANDLE` statement)
- Declaration of connection handle setup (`DECLARE CONNECTION HANDLE SET` statement)
- Cancellation of connection handle setup (`DECLARE CONNECTION HANDLE UNSET` statement)
- Release of connection handle (`FREE CONNECTION HANDLE` statement)

## 6.9.3 Transaction startup and termination

A transaction is started when the first SQL is executed, and a transaction terminates when a synchronization point is set (a commit or rollback). Any number of transactions can be started and terminated while connection to HiRDB is maintained. The following figure shows examples of transaction startup and termination.

Figure 6–22: Examples of transaction startup and termination



## 6.9.4 Commit and rollback

The process of placing into effect in the database the updates made by a transaction is called *commit*. The process of invalidating the updates made by a transaction is called *rollback*. When commit and rollback occur is determined as follows:

- At the times defined by an SQL
- At the times that are set automatically by HiRDB

Each of these types of commit and rollback timing is explained as follows.

### (1) Commit timing

This section explains commit timing.

**Timing defined by an SQL**

The `COMMIT` control SQL statement can be specified to commit a transaction whenever a `COMMIT` statement is executed.

**Commit set automatically by HiRDB**

- Commit is performed automatically by HiRDB when a definition SQL or `PURGE TABLE` statement is executed.
- Commit is performed automatically by HiRDB when a UAP terminates.

### (2) Rollback timing

This section explains rollback timing.

**Timing defined by an SQL**

The `ROLLBACK` control SQL statement can be specified to roll back the process to the previous commit point whenever a `ROLLBACK` statement is executed.

**Rollback set automatically by HiRDB**

- If a process cannot continue during SQL execution, HiRDB will roll back the process implicitly to the previous commit point.
- If a UAP terminates abnormally, HiRDB will roll back the process to the previous commit point.

## (3) Commitment control on a HiRDB parallel server configuration

A HiRDB parallel server configuration provides the following two methods of commitment control:

- One-phase commit
- Two-phase commit

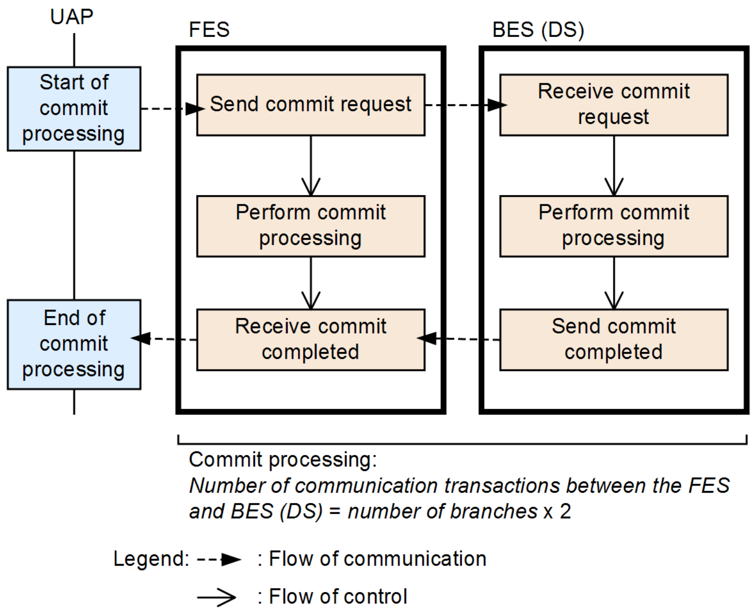
### (a) One-phase commit

With one-phase commit, only commit processing is performed, rather than both prepare processing and commit processing (which is the case with two-phase commitment control). This means that the number of communication transactions for synchronization point processing between the front-end server and the back-end server (dictionary server) is *the number of branches* × 2 (whereas, with two-phase commit, it is *the number of branches* × 4), which improves transaction processing performance. To use one-phase commit, specify `ONEPHASE` (default) in the `pd_trn_commit_optimize` operand.

One-phase commit is used only when one branch within a single transaction is being updated. Otherwise, two-phase commit must be used.

The following figure shows the processing for one-phase commit.

Figure 6–23: Processing for one-phase commit

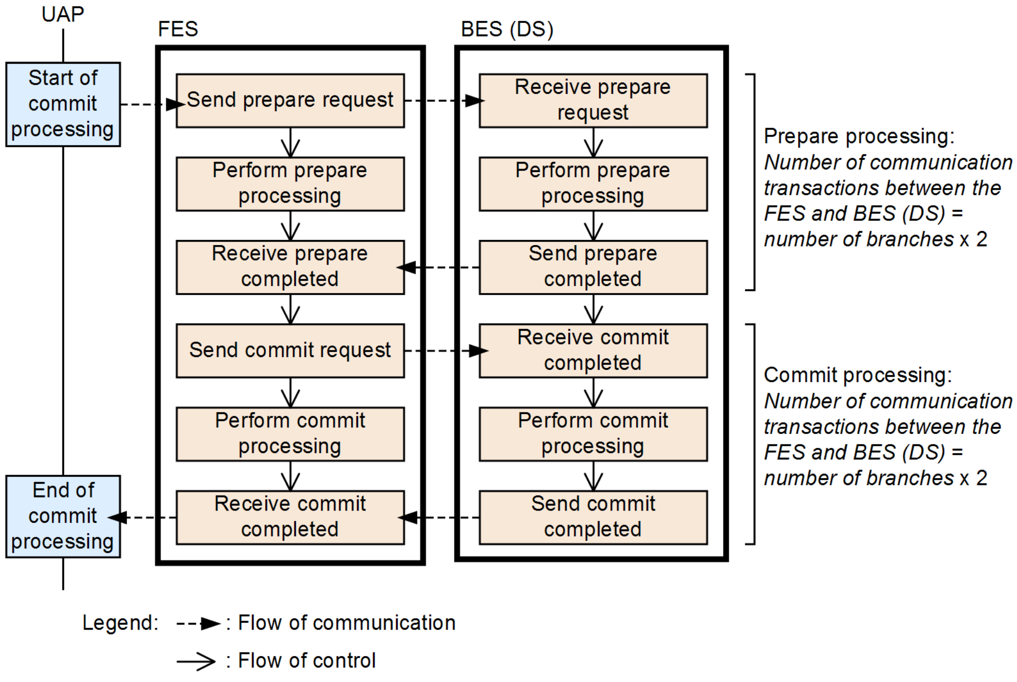

UAP
FES
BES (DS)
Start of commit processing
Send commit request
Receive commit request
Perform commit processing
Perform commit processing
End of commit processing
Receive commit completed
Send commit completed
Commit processing:
*Number of communication transactions between the FES and BES (DS) = number of branches* x 2
Legend: --► : Flow of communication
→ : Flow of control

The process of executing one-phase commit for commitment control is called *one-phase optimization*.

### (b) Two-phase commit

Two-phase commit performs synchronization point processing of the transaction by separating commitment control into two phases, prepare processing and commit processing. The number of communication transactions for synchronization point processing between the front-end server and the back-end server (dictionary server) is *the number of branches* × 4. The following figure shows the processing for two-phase commit.

Figure 6–24: Processing for two-phase commit


UAP
FES
BES (DS)
Start of commit processing
Send prepare request
Receive prepare request
Perform prepare processing
Perform prepare processing
Receive prepare completed
Send prepare completed
Send commit request
Receive commit completed
Perform commit processing
Perform commit processing
End of commit processing
Receive commit completed
Send commit completed
Prepare processing:
Number of communication transactions between the FES and BES (DS) = number of branches x 2
Commit processing:
Number of communication transactions between the FES and BES (DS) = number of branches x 2
Legend:
: Flow of communication
: Flow of control

The type of commitment control used on a HiRDB parallel server configuration is determined by the issuer of the commit and the execution environment of the transaction. The following table shows how commitment control is determined on a HiRDB parallel server configuration.

Table 6–11: Commitment control on a HiRDB parallel server configuration

| Condition | | | HiRDB commitment control |
|---|---|---|---|
| Issuer of the commit | Commitment control indicated by the commit issuer | Execution environment of the transaction | |
| UAP | -- | For a reference transaction | One-phase commit |
| | | If one server is updated by the transaction and ONEPHASE is specified (or nothing is specified) in the pd_trn_commit_optimize operand | |
| | | Other than the above | Two-phase commit |
| OLTP system | One-phase commit | For a reference transaction | One-phase commit |
| | | If one server is updated by the transaction and ONEPHASE is specified (or nothing is specified) in the pd_trn_commit_optimize operand | |
| | | Other than the above | Two-phase commit |
| | Two-phase commit | For a reference transactions | One-phase commit |
| | | Other than the above | Two-phase commit |

--: Not applicable

## 6.9.5 UAP transaction management under OLTP environment

When transaction processing that conforms to the XA interface is performed using a UAP under the OLTP environment, commit or rollback of the transaction is executed from the UAP using an *API that is compatible with X/Open*.

**Transaction transfer**

Execution of transaction commitment using a different process from the one the UAP used to access HiRDB is called *transaction transfer*. UAP in this context refers to the UAP that connects to HiRDB by means of the HiRDB XA library.

## 6.9.6 Automatic reconnect facility

When a connection to a HiRDB server is lost due to a server process going down, system switchover, network failure, or other cause, the *automatic reconnect facility* automatically re-establishes the connection. By using the automatic reconnect facility, you can continue execution of a UAP without having to be aware that the connection to the HiRDB server was lost. To use the automatic reconnect facility, specify `YES` in the `PDAUTORECONNECT` client environment definition.

**Application criteria**

Whenever you are executing any of the following processes on a HiRDB server, all HiRDB clients are placed in wait status until that process is completed:

- Executing the system configuration command (`pdchgconf`)
- Updating to the HiRDB update version
- Using the transaction queuing facility to execute planned system switchover

While in wait status, the elapsed time is monitored against the time specified in `PDCWAITTIME`. If the time specified in `PDCWAITTIME` is exceeded, wait status is released, and a `PDCWAITTIME` timeout error is returned to the UAPs.

Depending on the timing of this error, communications processing errors might occur because the UAPs might not be able to detect that the aforementioned process is being executed. If you know in advance that the aforementioned process will be executed, consider using the automatic reconnect facility. By using this facility, processing continues without an error being returned to the UAPs, even when the aforementioned process is being executed.

# 6.10 Locking

When multiple users attempt to manipulate data in the same table at the same time, the HiRDB system automatically implements locking in order to maintain data integrity. Especially inasmuch as the individual servers in a HiRDB parallel server configuration do not in principle share resources, an independent means of locking is used for each server. Depending on the nature of the operation to be performed, you can modify a lock that is implemented automatically by HiRDB.

## 6.10.1 Units of locking

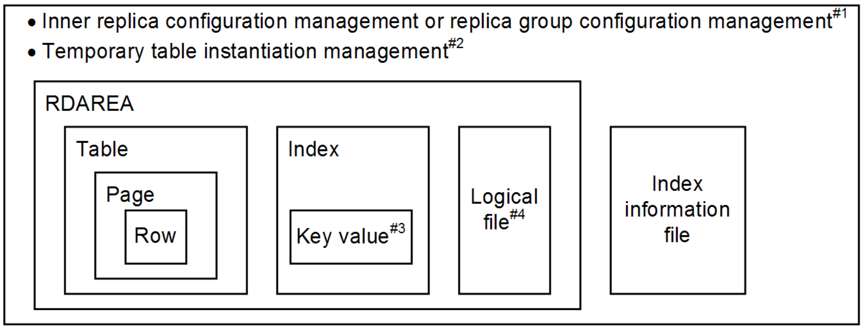
This section explains the units of locking and their inclusion relationships.

### (1) Locked resources and their inclusion relationships

HiRDB prevents illegal referencing and updating by applying locks based on a unit called the *locked resource*. Locking is applied sequentially from top to bottom of the locked resources. If when locking is applied there is a transaction that cannot execute because it is in contention with other transactions for a resource, that transaction goes onto wait status.

Locked resources have inclusion relationships. Therefore, once a lock is applied to a higher-order resource, there is no need to apply locks to the resources that are below it hierarchically. The figure below shows locked resources and their inclusion relationships. For example, a table is a higher-order resource and a page is a lower-order resource.

Figure 6–25: Locked resources and their inclusion relationships


• Inner replica configuration management or replica group configuration management#1
• Temporary table instantiation management#2
RDAREA
Table
Page
Row
Index
Key value#3
Logical file#4
Index information file

#1

In the UNIX edition only:

When the inner replica facility is used, the highest-order locked resource is the inner replica configuration management information or replica group configuration management information.

When the inner replica configuration management information cannot be locked, a lock is applied to the replica group configuration management information. When an RDAREA that does not define a replica RDAREA is accessed, locking is also applied. This prevents a replica RDAREA from being defined in a regular RDAREA or the configuration inside the inner replica group from being modified while a job is being executed.

#2

Temporarily acquired when a temporary table is instantiated. For details about the locks acquired in operations performed on temporary tables, see the *HiRDB Version 9 Installation and Design Guide*.

#3

If index key value no-lock is applied, the key value is not locked. For details about index key value no-lock, see the *HiRDB Version 9 UAP Development Guide*.

#4

The logical file is used by a plug-in.

### (2) Setting the lowest-order locked resource unit

For each table, the user can set the lowest-order locked resource unit for the locking to be applied automatically by HiRDB. The lowest-order locked resource unit, the setting procedures, and their advantages and disadvantages are explained as follows.

#### (a) Locking by row

To set the row as the lowest-order unit of locking, `LOCK ROW` is specified in the `CREATE TABLE` definition SQL. Locking that uses the row as the lowest-order unit of locking is called *row locking*. Compared to locking by page, locking by row results in better concurrent execution because it is a lower unit of locked resources. On the other hand, locking by row increases the processing time required for locking and increases memory usage.

#### (b) Locking by page

To set the page as the lowest-order unit of locking, `LOCK PAGE` is specified in the `CREATE TABLE` definition SQL. Locking that uses the page as the lowest-order unit of locking is called *page locking*. Compared to locking by row, locking by page reduces the processing time required for locking and reduces memory usage. On the other hand, locking by page results in poor concurrent execution.

## 6.10.2 Lock modes

HiRDB provides the following five *lock modes* for each type of locked resource (such as table or page):

1. Shared mode (PR: Protected Retrieve)
   Only referencing by other transactions is allowed.
2. Exclusive mode (EX: EXclusive)
   One transaction monopolizes the locked resource and does not allow referencing, adding, updating, or deletion by other transactions.
3. Shared retrieval mode (SR: Shared Retrieve)
   Only referencing is allowed. Other transactions are allowed to perform referencing, adding, updating, and deletion of the locked resource.
4. Shared update mode (SU: Shared Update)
   If the lock is applied to a certain resource in the exclusive mode, the shared update mode applies to the resources above that locked resource; referencing, adding, updating, and deletion can be performed by other transactions.
5. Protected update mode (PU: Protected Update)
   Referencing, adding, updating, and deletion can be performed; referencing only can be performed by other transactions. Unlike modes 1-4, this mode occurs as a result of lock mode transition.

## 6.10.3 Automatic locking by HiRDB

This section explains the locking of tables and rows that is performed automatically by HiRDB during data updating, addition, and deletion and during data retrieval.

### (1) Locking by HiRDB during data updating, addition, and deletion

**Locking of a table**

Referencing, adding, updating, and deletion of the table are allowed by other transactions as well. In other words, the *shared update (SU) mode* goes into effect.

**Locking of a row**

The transaction monopolizes the resource and does not allow referencing, adding, updating, or deletion of the row by other transactions. In other words, the *exclusive (EX) mode* goes into effect.

### (2) Locking by HiRDB during data retrieval

**Locking of a table**

Referencing, adding, updating, and deletion of the table are allowed by other transactions as well. In other words, the *shared retrieval (SR) mode* goes into effect.

**Locking of a row**

The transaction monopolizes the resource and allows only referencing of the row by other transactions. In other words, the *shared (PR) mode* goes into effect.

## 6.10.4 Changing the lock based on a user setting

The user can change the lock that is implemented automatically by HiRDB. Locking of tables and rows during data updating, addition, and deletion and during data retrieval is explained as follows.

### (1) Locking during data updating, addition, and deletion based on a user setting

When `IN EXCLUSIVE MODE` is specified in the `LOCK TABLE` statement, the user can change the lock to be used during data updating, addition, and deletion. Referencing, adding, updating, or deletion of data in the table by other transactions are not allowed. In other words, the *exclusive (EX) mode* goes into effect. On the other hand, because row-based locking is not applied, the overhead for applying the lock is reduced, thus preventing a shortage of buffers for locking. However, because table-based locking is applied, other transactions might have to wait for a long time.

### (2) Locking during data retrieval based on a user setting

When either of the following SQLs is specified, the user can change the lock to be used during data retrieval:

**WITHOUT LOCK of SELECT statement**

The data that is retrieved is not locked until the transaction terminates. Because locking is released beginning with rows that have been retrieved, better concurrent execution can be achieved.

**IN SHARE MODE of LOCK TABLE statement**

The transaction monopolizes the resource and allows only referencing of the data in the table by other transactions. In other words, the *shared (PR) mode* goes into effect. On the other hand, because row-based locking is not applied, the overhead for applying the lock is reduced, thus preventing a shortage of buffers for locking. However, because table-based locking is applied, other transactions might have to wait for a long time.

## 6.10.5 Lock period

When a transaction locks a resource, it monopolizes that resource until commit or rollback occurs. For example, when an update is made to a locked resource (row or page), the automatic locking by HiRDB implements the exclusive mode, which does not allow referencing, adding, updating, or deletion by other transactions. Consequently, other transactions must wait until commit or rollback occurs before it becomes possible to access the row that is being updated. However, if `UNTIL DISCONNECT` is specified in the LOCK statement, the lock will remain in effect until a `DISCONNECT` statement is executed or until commit occurs after the table is deleted.

## 6.10.6 Deadlock

*Deadlock* is the status when two transactions are both waiting for the other to release resources when they have allocated the same resources but in a different order. Consequently, processing by both transactions is stalemated and cannot proceed. Deadlock often occurs between a referencing transaction and an updating or deletion transaction. Changing the access order of the UAPs can reduce the incidence of deadlocks. When deadlock occurs during operation of HiRDB, HiRDB outputs deadlock information that includes information on deadlocks that have occurred among multiple transactions in the server. For details about preventing deadlock, see the *HiRDB Version 9 UAP Development Guide.*

# 6.11 Operation without collecting a database update log

HiRDB acquires a history of database updates (the database update log in the system log) by UAPs (or utilities#). This information is collected into system log files. Acquisition of database update log information can also be disabled; the advantage of doing so is that processing time, and consequently, execution time, for the UAP (or utility) is reduced.

#: The following utility programs can be used:

- Database load utility (pdload)
- Database reorganization utility (pdrorg)
- Rebalancing utility (pdrbal)

## (1) Acquiring a database update log

There are three modes for collecting the database update log that is used during execution of a UAP (or a utility), as shown in the following table.

Table 6–12: Database update log acquisition modes

| Database update log acquisition mode | Explanation |
|---|---|
| Log acquisition mode | Acquires the database log needed for rollback or roll-forward. Normally, the log acquisition mode is used. |
| Pre-update log acquisition mode | Acquires only the database update log necessary for a rollback. |
| No-log mode | Does not acquire a database update log. |

## (2) Specifying the database update log acquisition mode

The following table shows the methods for specifying the database update log acquisition mode.

Table 6–13: Methods of specifying the database update log acquisition mode

| Database update log acquisition mode specification method | Explanation |
|---|---|
| UAP | The database update log acquisition method is specified in the PDDBLOG operand of the client environment definition. You can specify in the PDDBLOG operand either the log acquisition mode or the no-log mode; the pre-update log acquisition mode cannot be specified. |
| Database load utility (pdload) | The database update log acquisition method is specified in the -l option of the database load utility (pdload). |
| Database reorganization utility (pdrorg) | The database update log acquisition method is specified in the -l option of the database reorganization utility (pdrorg). |
| Rebalancing utility (pdrbal) | The database update log acquisition method is specified in the -l option of the rebalancing utility (pdrbal). |
| Using a user LOB RDAREA | In the case of data stored in a user LOB RDAREA, the database update log acquisition method is specified in the RECOVERY operand of CREATE TABLE. If abstract data type data is stored in the LOB RDAREA, the pre-update log acquisition mode cannot be specified (it is ignored if specified). |

## (3) Notes on the RECOVERY operand

The database update log acquisition method specified in the RECOVERY operand is subject to change when either the PDDBLOG operand or the -l option is specified. The following table shows the relationship between the specification value in the RECOVERY operand and the PDDBLOG operand or the -l option, on the one hand, and the value that is assumed during execution of the UAP (or the utility), on the other hand.

Table 6–14: Relationship between the specification value in the RECOVERY operand and PDDBLOG operand or the -l option, and the value that is assumed during execution of the UAP (or the utility)

| Specification of PDDBLOG operand or -l option | Specification of RECOVERY operand | | |
|---|---|---|---|
| | ALL | PARTIAL | NO |
| ALL<br>a | ALL | PARTIAL | NO |
| PARTIAL<br>p | PARTIAL | PARTIAL | NO |
| NO<br>n | NO | NO | NO |

ALL and a: log acquisition mode

PARTIAL and p: pre-update log acquisition mode

NO and n: no-log mode

## (4) Differences in operating method depending on database update log acquisition mode

The different database update log acquisition modes can result in differences in the operating methods in the following two areas:

- Processing performed by HiRDB and action to be taken by the user in the event of abnormal termination of the UAP
- Timing for database recovery

### (a) Processing performed by HiRDB and the action to be taken by the user in the event of abnormal termination of the UAP

The following table shows the processing performed by HiRDB and the action to be taken by the user in the event of abnormal termination of the UAP.

Table 6–15: Processing performed by HiRDB and the action to be taken by the user in the event of abnormal termination of the UAP

| Database update log acquisition mode | HiRDB processing | User action |
|---|---|---|
| Log acquisition mode | Rolls back an updated RDAREA to its status before the UAP executed or to the last synchronization point set before the abnormal termination. | If the RDAREA is rolled back to its status before the UAP was executed, re-execute the UAP. If the RDAREA is rolled back to the last synchronization point that was set before the abnormal termination, execute the processing following the synchronization point. |
| Pre-update log acquisition mode | | |
| No-log mode | Does not perform roll back. Error shutdown (logless shutdown) is performed on the updated RDAREA. The contents of the RDAREA are destroyed. | Using as input information a backup that was made before execution of the UAP, execute the database recovery utility (pdrstr) to recover the RDAREA, and then re-execute the UAP. |

### (b) Timing at which database can be recovered

The following table shows the times at which the database can be recovered by the database recovery utility (pdrstr).

Table 6–16: Times for recovering the database

| Database update log acquisition more | Time for database recovery |
|---|---|
| Log acquisition mode | Either the backup acquisition point or any synchronization point subsequent to the backup acquisition point |
| Pre-update log acquisition mode | Backup acquisition point |
| No-log mode | |

## (5) Notes on backup (important)

1. During execution of a UAP (or utility) in the no-log mode or the pre-update log acquisition mode, do not make a backup using the `updatable` mode (`-M s` specified).
2. After execution of a UAP (or utility) in the no-log mode or the pre-update log acquisition mode, make a backup in one of the following modes:
   - Referencing/updating-impossible mode (`-M x` specified)
   - Referencing-permitted mode (`-M r` specified)

# 7 Database Management

This chapter explains how the HiRDB administrator manages a database.

# 7.1 Database recovery

The database recovery utility (`pdrstr`) is provided in the event adatabase needs to be recovered because of a disk error or for some other reason. This section briefly explains how to recover databases. For details about how to recover databases, see the *HiRDB Version 9 System Operation Guide*.

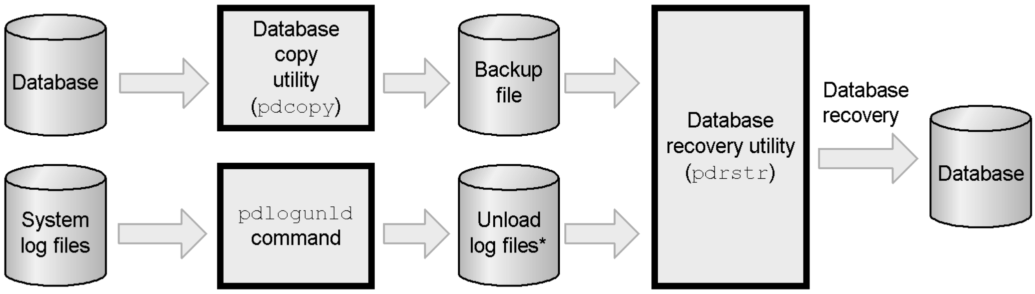
## 7.1.1 Overview of database recovery

To recover a database, you use the following files as input information for the database recovery utility (`pdrstr`).

- Backup files
- Unload log files

The following figure provides an overview of the database recovery process.

Figure 7–1: Overview of database recovery process



#: An *unload log file* is created by unloading system log files (system log information) that contain information comprising a history of the updates to the database.

**Explanation**

- Backups are made by the database copy utility (`pdcopy`).
- The `pdlogunld` command creates an unload log file.
- Using the backup and unload log files as the input, the database recovery utility (`pdrstr`) recovers the database.

For details about making backup files and of creating unload log files, see Section *7.2 Preparations for database errors*.

## 7.1.2 Times at which database can be recovered

A database can be recovered to its status at the following times:

- At the time a backup was made
- To its most recent status (the point of the most recent synchronization point)

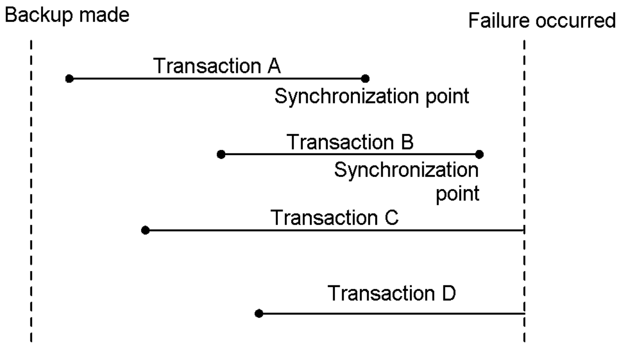
### (1) Recovering to the point a backup was made

To recover a database to its status when a backup was made, you do not need an unload file. Only the backup file is required.

### (2) Recovering to the most recent synchronization point before the failure occurred

The point in time at which a transaction is completed is called a *synchronization point*. Synchronization point processing that validates a transaction-induced update is called a *commit*. Synchronization point processing that invalidates a transaction is called a *rollback*. Recovering a database to a transaction synchronization point is called *recovering to the most recent synchronization point before the failure occurred*. Because any transaction being processed when the failure occurred (a transaction that had not reached a synchronization point) is invalid, update processing performed by such transactions is not recovered. The following figure shows the transactions that are recovered.

Figure 7–2: Transactions that are recovered (with recovery to the most recent synchronization point before the failure occurred)



**Explanation**

Because processing of transactions A and B finished and they reached their synchronization points, the database is recovered to these synchronization points.

Because processing of transactions C and D was ongoing, the processing of these transactions is invalid, and is not recovered.

To recover to the most recent synchronization point before the failure occurred, in addition to backup files, you need the unload log files from which log files are output after a backup was made.

# 7.2 Preparations for database errors

If an error occurs in the database as a result of a disk error or for some other reason, the HiRDB administrator must perform the database recovery operation. To be prepared for database errors, the HiRDB administrator is responsible for performing the following tasks:

- Making backups
- Unloading system log information (creating unload log files)

In addition, the following functions are available as options for making backups:

- Differential backup facility
- Backup hold
- Frozen update command (`pddbfrz` command)
- NetBackup linkage facility

## 7.2.1 Making backups

To be prepared for database errors, it is important to make backup copies of a database on a regular basis; the *database copy utility* (`pdcopy`) is provided for this purpose.

### (1) Backup units

The unit is specified in an option of the database copy utility (`pdcopy`). The following table shows the backup units.

Table 7–1: Backup units

| Backup unit | Explanation | `pdcopy` option specification |
|---|---|---|
| System | Makes a backup of all RDAREAs, including RDAREAs used by the system (such as the master directory RDAREA). | `-a` |
| Unit# | Makes a backup of a specified unit's RDAREAs. | `-u` *unit-name* |
| Server# | Makes a backup of a specified server's RDAREAs. | `-s` *server-name* |
| RDAREA | Makes a backup of a specified RDAREA. | `-r` *RDAREA-name* |

#: Applicable to HiRDB parallel server configurations.

### (2) Backup acquisition modes

A backup acquisition mode is specified in the `-M` option of the database copy utility (`pdcopy`). The following table lists the backup acquisition modes.

Table 7–2: Backup acquisition modes

| Backup acquisition mode (`-M` option specification) | Explanation | Difference in RDAREA recovery depending on backup acquisition point |
|---|---|---|
| Referencing/updating-impossible mode (`x`) | While the backup is being made, RDAREAs being backed up cannot be referenced or updated. Before a backup, you must use the `pdhold -c` command to shut down and close the affected RDAREAs. | A backup made with this method can be used to recover the database to its status when the backup was made. In addition, by using system log information, you can recover the database to any synchronization point since the backup was made. |
| Referencing-permitted mode (`r`) | While the backup is being made, the RDAREAs being backed up can be referenced but not updated. | |

| Backup acquisition mode (-M option specification) | Explanation | Difference in RDAREA recovery depending on backup acquisition point |
|---|---|---|
| Updatable mode (s)[#1, #2] | While the backup is being made, the RDAREAs being backed up can be both referenced and updated. | The database cannot be recovered to its status when the backup was made, but it can be recovered to any synchronization point since the backup was made. Therefore, to recover the database, you need the backup and the system log information[#3] from a synchronization point subsequent to when the backup was made. |

#1

UNIX edition only:

Specification of the updatable mode requires that the RDAREAs being backed up were created as character special files. RDAREAs created as regular files cannot be backed up in the updatable mode.

#2

Do not make a backup in the updatable mode during execution of a UAP (including a utility) that is running in the no-log mode or the pre-update log acquisition mode.

#3

The LAN IDs and generation numbers of the system log files required to recover the RDAREA are output to the process results output file of the database recovery utility.

### (3) Server machine for storage of backup file

A backup file can be created anywhere on a server machine on which HiRDB is running. A backup file need not be created on the server machine on which the RDAREAs are located. Devices such as CMT and DAT might not be available on the same server machine as the RDAREAs being backed up. The server machine on which a backup file is to be stored is specified in an option of the database copy utility (pdcopy).

## 7.2.2 Unloading system logs (creating unload log files)

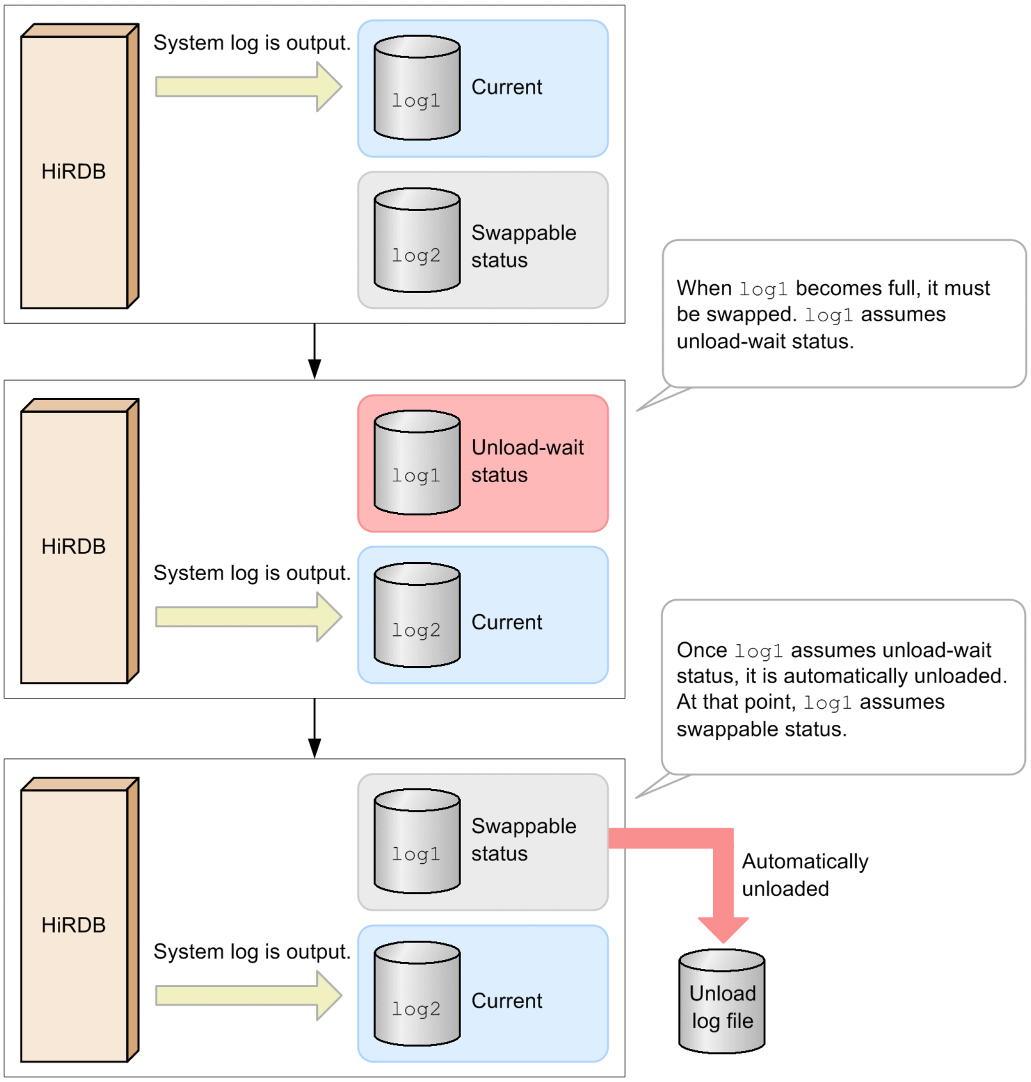
Unload log files, which are required for recovery of the database, are created by unloading the system logs. If you continue operations without unloading the system logs, no output destination exists for the system logs and HiRDB terminates abnormally. Therefore, we recommend that you automatically unload system logs by using the automatic log unloading facility. For details about the automatic log unloading facility, see *Using the automatic log unloading facility* in the *HiRDB Version 9 System Operation Guide*.

This subsection explains why unloading system logs is necessary, and how to retain unload log files.

### (1) Status of system log files

The file to which system log information is output is called the *current file*. If system log information fills the current file, the output for the system log information is changed to another system log file in a process called *system log file swapping*; what was the current file becomes an *unload-wait file*. If you use the automatic log unloading facility, the system log will automatically be unloaded when the system log file assumes the unload wait status. The following figure shows the system log file status changes.

Figure 7–3: System log file status changes



System log information cannot be output to an unload-wait file. *If all system log files become unload-wait files, system log information cannot be output any longer, and HiRDB terminates abnormally*. We therefore recommend using the automatic log unloading facility. Furthermore, if the disk holding the directory for unload log files becomes full, the automatic log unloading facility stops working. We therefore recommend that you periodically delete unnecessary unload log files to ensure that the automatic log unloading facility does not stop. For details about retaining unload log files, see *(3) Retaining unload log files*.

Note that the HiRDB administrator can also manually perform unloading by using the `pdlogunld` command. In this case, take special note of the following:

**Special considerations for unloading manually**

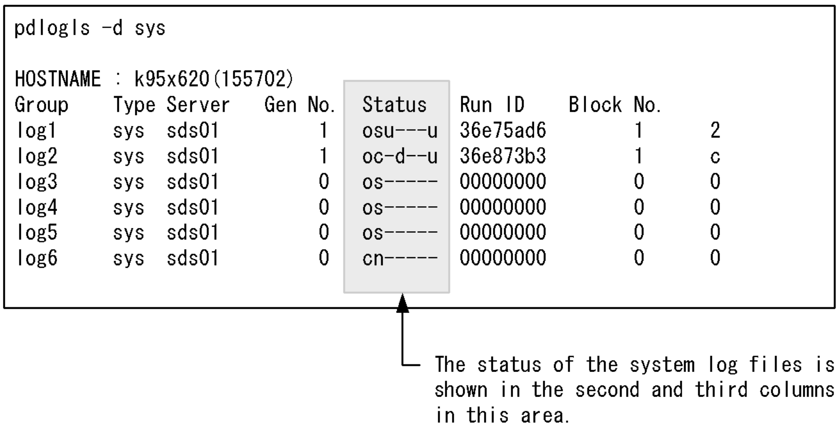
> When HiRDB is started normally following a normal termination, system log files are swapped, creating an unload-wait file. If you perform unloading manually, always unload this unload-wait file. *If termination and startup are repeated without unloading system log information, HiRDB will eventually be unable to start*.
>
> Specifying `pd_log_rerun_swap=Y` in the server definition causes system log files to be swapped when HiRDB is restarted.

## (2) Determining the status of the system log files

If you want to perform the unloading manually, you must first check the status of the system log files. You can check the status of the system log files by using the `pdlogls` command, as the following example shows.

**Example**

Use the `pdlogls` command to check the status of the system log files.



```
pdlogls -d sys

HOSTNAME : k95x620(155702)
Group    Type Server   Gen No.  Status   Run ID    Block No.
log1     sys  sds01          1  osu---u  36e75ad6         1     2
log2     sys  sds01          1  oc-d--u  36e873b3         1     c
log3     sys  sds01          0  os-----  00000000         0     0
log4     sys  sds01          0  os-----  00000000         0     0
log5     sys  sds01          0  os-----  00000000         0     0
log6     sys  sds01          0  cn-----  00000000         0     0
```

The status of the system log files is shown in the second and third columns in this area.

**Explanation**

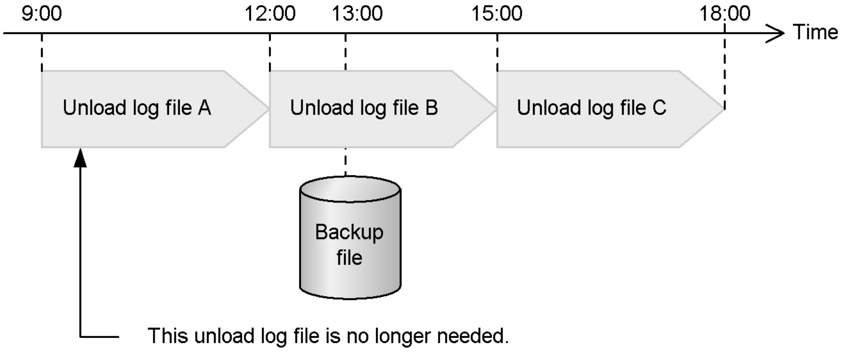
- The file for which a `c` is in column 2 is the current file. In this case, it is `log2`.
- The file for which a `u` or an `a` is in column 3 is an unload-wait file. In this case, it is `log1`. If you are performing unloading manually, this file must be unloaded.

## (3) Retaining unload log files

Repeated unloading of system log information results in a proliferation of unload log files, which occupy disk space. The HiRDB administrator must delete unload log files as soon as they are no longer needed.

When a backup is made, unload log files containing only data produced prior to the backup are no longer needed. The following figure shows the relationship between a backup and unload log files.

Figure 7–4: Relationship between backup and unload log files



**Explanation**

- The system log information that was output from 9:00 to 12:00 is unloaded to unload log file A.
- The system log information that was output from 12:00 to 15:00 is unloaded to unload log file B.
- The system log information that was output from 15:00 to 18:00 is unloaded to unload log file C.
- If a backup is made around 13:00, the backup and unload log files B and C will be needed for any recovery of the database, but unload log file A will not be needed.

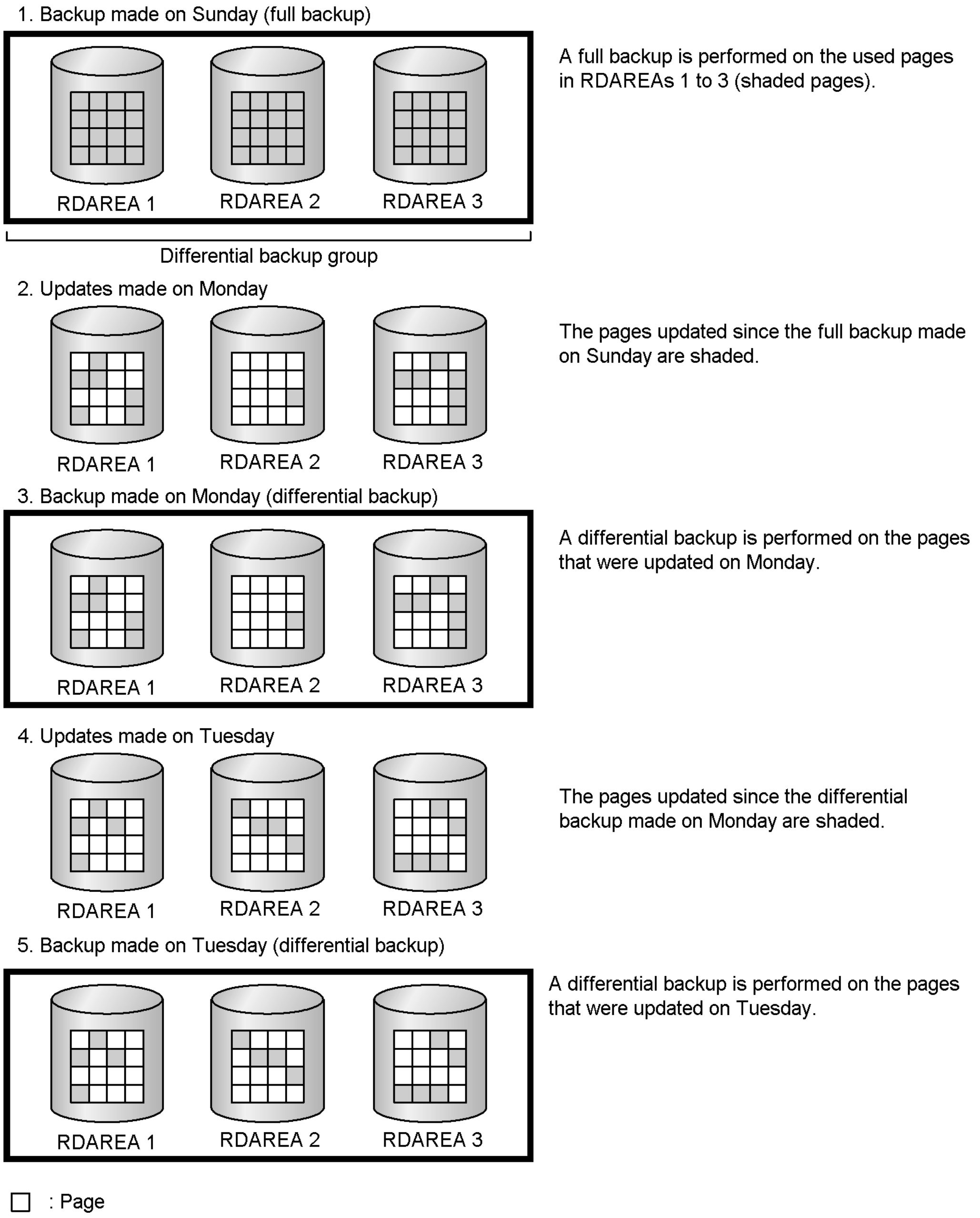
**!** Important note

Store unload log files on a different disk from the disk containing the database. If they are all stored on the same disk, an error affecting the disk will prevent recovery of the database.

## 7.2.3 Differential backup facility

Because backups are normally made on a per-RDAREA basis, the backup includes pages that have been updated as well as pages that have not been updated since the last backup. By using the differential backup facility, only the pages that have been updated since the last time a backup was made are backed up. Making a backup of only the differences since the last time a backup was made allows you to significantly reduce the time required to make a backup. Consider using the differential backup facility in situations where the database is large and the amount of data updating is relatively small. The following figure provides an overview of the differential backup facility.

Figure 7–5: Overview of the differential backup facility


1. Backup made on Sunday (full backup)
RDAREA 1
RDAREA 2
RDAREA 3
Differential backup group
2. Updates made on Monday
RDAREA 1
RDAREA 2
RDAREA 3
3. Backup made on Monday (differential backup)
RDAREA 1
RDAREA 2
RDAREA 3
4. Updates made on Tuesday
RDAREA 1
RDAREA 2
RDAREA 3
5. Backup made on Tuesday (differential backup)
RDAREA 1
RDAREA 2
RDAREA 3
□ : Page

A full backup is performed on the used pages in RDAREAs 1 to 3 (shaded pages).

The pages updated since the full backup made on Sunday are shaded.

A differential backup is performed on the pages that were updated on Monday.

The pages updated since the differential backup made on Monday are shaded.

A differential backup is performed on the pages that were updated on Tuesday.

**Explanation**

1. On Sunday, backups are made of RDAREAs 1 through 3. At this time, a backup is made of the used pages in RDAREAs 1 through 3. This kind of backup is called a *full backup*, and the grouped RDAREAs are called a *differential backup group*.
2. Updates are performed with Monday's work.

3. At the end of Monday's work day, a backup is made of RDAREAs 1 through 3. This time, a backup is made only of the pages in RDAREAs 1 through 3 that were updated. This kind of backup is called a *differential backup*.
4. Updates are performed with Tuesday's work.
5. At the end of Tuesday's work day, a backup is made of RDAREAs 1 through 3. At this time, a backup is made only of the pages in RDAREAs 1 through 3 that were updated.

**Database recovery procedure**

To recover a database up to the point the differential backup was made on Tuesday, as the input information for the database recovery utility, you use the full backup made on Sunday, the differential backup made on Monday, and the differential backup made on Tuesday. For details about how to operate the differential backup facility, see the *HiRDB Version 9 System Operation Guide*.

! Important note

A differential backup cannot be made of a LOB RDAREA.

Reference note

The differential backup facility can also be used during operation without unloading the system log.

## 7.2.4 Backup-hold

When a backup is made with a command other than the `pdcopy` command (such as when another product's backup facility is used), place the RDAREAs on *backup-hold*. An RDAREA on backup-hold can be backed up using the backup facility of another product while HiRDB is running. To place an RDAREA on backup-hold, you specify the `-b` option in the `pdhold` command.

### (1) Application example of backup-hold

UNIX edition

If you are using Logical Volume Manager (LVM) on the database, you can make backups by using the `pdcopy` command in updatable mode by placing the RDAREA to be backed up on backup-hold (`-b u` or `-b wu`).

Windows edition

You can make backups using the `pdcopy` command in updatable mode by placing the RDAREA to be backed up on backup-hold (`-b u` or `-b wu`).

### (2) Backup-hold types

The following table lists and describes the four types of backup-hold.

Table 7–3: Types of backup-hold

| Backup-hold type | Explanation |
|---|---|
| Referencing-permitted backup-hold | During backup-hold, RDAREAs that have been backed up and held can be referenced; an updating attempt will result in an SQL error (`-920`). |
| Referencing-permitted backup-hold (update WAIT mode) | During backup-hold, RDAREAs that have been backed up and held can be referenced; an updating attempt goes onto lock-release wait status until the backup-hold is released. |
| Updatable backup-hold | During backup-hold, RDAREAs that have been backed up and held can be referenced and updated. Even while an updating transaction is being executed, an RDAREA is placed immediately on updatable backup-hold status without placing the `pdhold` command on wait status. |
| Updatable backup-hold (WAIT mode) | During backup hold, RDAREAs that have been backed up and held can be referenced and updated. If an updating transaction is being executed, the `pdhold` command is kept waiting until the transaction terminates. |

For details about how to make a backup using the backup shutdown, see the *HiRDB Version 9 System Operation Guide*.

Reference note

Reference-possible backup hold and reference-possible backup hold (update WAIT mode) are also referred to as *committing a database*.

## 7.2.5 Reducing the time needed to make backups of user LOB RDAREAs (frozen update command)

Using the frozen update command (`pddbfrz` command) allows you to reduce the time needed to make backups of user LOB RDAREAs. Consider using the frozen update command for the following operations:

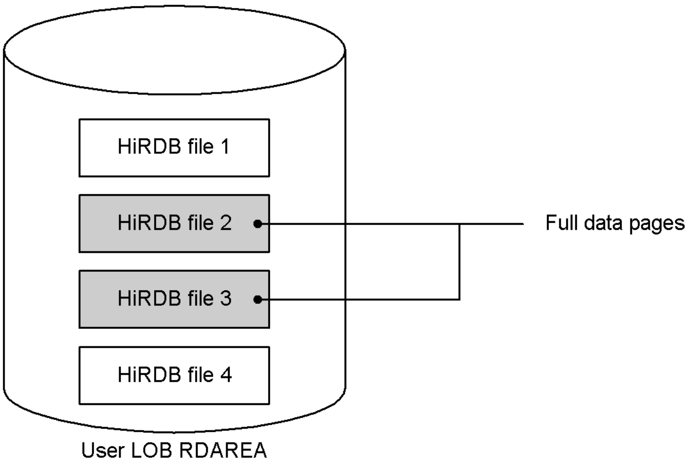
- Making a backup of user LOB RDAREAs using a method other than the database recovery utility (`pdcopy` command)
- Making a backup of HiRDB files on a file-by-file basis
- Performing operations that do not typically occur when updating or deleting registered LOB data (operations that typically occur when adding LOB data)
- Making a backup of a user LOB RDAREA consisting of multiple HiRDB files

For details about how to make backups using the frozen update command, see the *HiRDB Version 9 System Operation Guide*.

### (1) Overview of the frozen update command

When you execute the frozen update command, HiRDB files with data pages that are entirely full (all pages are allocated) in a user LOB RDAREA are placed in *frozen update status*. Data in a HiRDB file that has been placed in frozen update status cannot be updated or deleted. The following figure provides an overview of frozen update command processing.

Figure 7–6: Overview of frozen update command processing



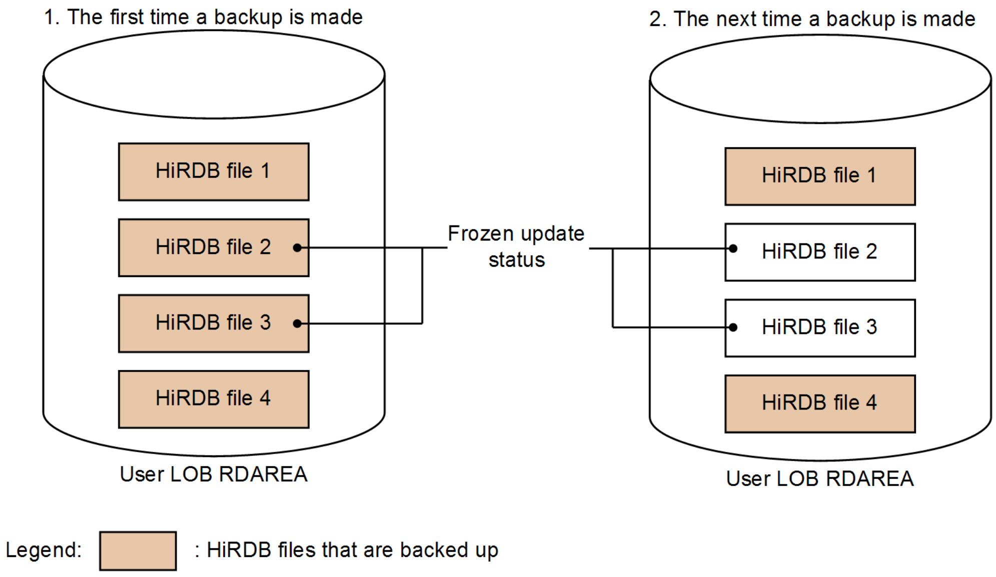
**Explanation**

- A user LOB RDAREA exists that contains HiRDB files 1 through 4. All data pages in HiRDB files 2 and 3 are full.
- Executing the frozen update command on this user LOB RDAREA places HiRDB files 2 and 3 into frozen update status. HiRDB files that are in frozen update status are indicated in the `KFPH27024-I` message.
- HiRDB files 1 and 4 are placed in permit update status.

## (2) How to use the frozen update command to make backups

The following figure shows how to use the frozen backup command to make backups.

Figure 7–7: How to use the frozen update command to make backups



**Explanation**

The frozen update command is executed before the backup is made. As a result, HiRDB files 2 and 3 are placed in frozen update status.

1. The first time the backup is made, a backup is made of all HiRDB files (HiRDB files 1 through 4).
2. Because HiRDB files 2 and 3 are in frozen update status, their contents do not change after the first time the backup is made. Therefore, the next time a backup is made, HiRDB files 2 and 3 do not need to be backed up. Only HiRDB files 1 and 4 are backed up.

**Remark**

Because the first HiRDB file (HiRDB file 1 in Figure 7-7) contains a management record, even if the data area becomes full, the file is still written to on a regular basis. Consequently, backups are always made of the first HiRDB file.

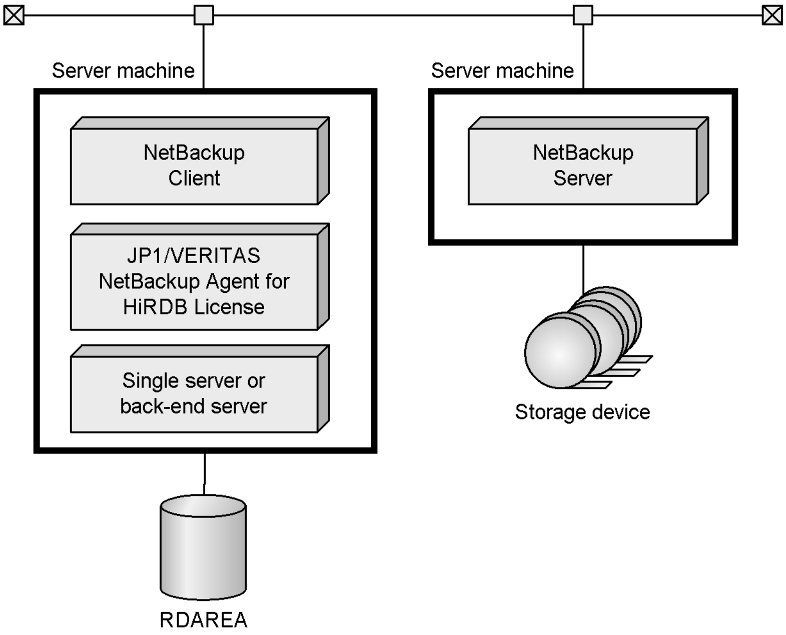
# 7.2.6 NetBackup linkage facility

Using the NetBackup linkage facility allows you to create backup files for use by the database copy utility (`pdcopy`) or database recovery utility (`pdrstr`) on a medium managed by a NetBackup server. To use the NetBackup linkage facility, you need JP1/VERITAS NetBackup Agent for HiRDB License.

## (1) System configuration example of the NetBackup linkage facility

By using the NetBackup linkage facility, you can create backup files on a medium that is managed by the NetBackup server. The following figure shows a system configuration example using the NetBackup linkage facility.

Figure 7–8: System configuration example of the NetBackup linkage facility



For details about the NetBackup linkage facility, and about the products required to use the NetBackup linkage facility, see *JP1/VERITAS NetBackup v4.5 Agent for HiRDB License Description and User's Guide*.

## (2) Advantages of using the NetBackup linkage facility

The following describes the advantages of using the NetBackup linkage facility:

- Ability to manage backup files with NetBackup

  Because NetBackup manages the output destination of backup files as well as the backup date and time, you do not need to change the name of the backup file when you make a backup. In addition, when you are recovering (restoring), and simply specifying the volume name and backup date and time, NetBackup selects the correct backup file for you. Moreover, if you are using the most recent backup file, you do not even have to specify the backup date and time.

- Ability to create backup files on storage devices connected to server machines on which HiRDB is not installed

  You can create backup files on a storage device that is connected to a server machine on which HiRDB is not installed. If you do not use the NetBackup linkage facility, you can only create backup files on storage devices connected to server machines on which HiRDB is installed.

- Ability to use a wide variety of media types for backup files

  You can create backup files on any media that are supported by NetBackup. For details about the media that are supported by NetBackup, see the NetBackup documentation.

- Ability to use different operating systems for the HiRDB server and the NetBackup server

  You can use the NetBackup linkage facility even if the operating systems on which the HiRDB server and the NetBackup server run differ. For example, even if the HiRDB server and the NetBackup client are on HP-UX, and the NetBackup server is on Windows, you can still use the NetBackup linkage facility.

# 7.3 Reorganizing tables and indexes

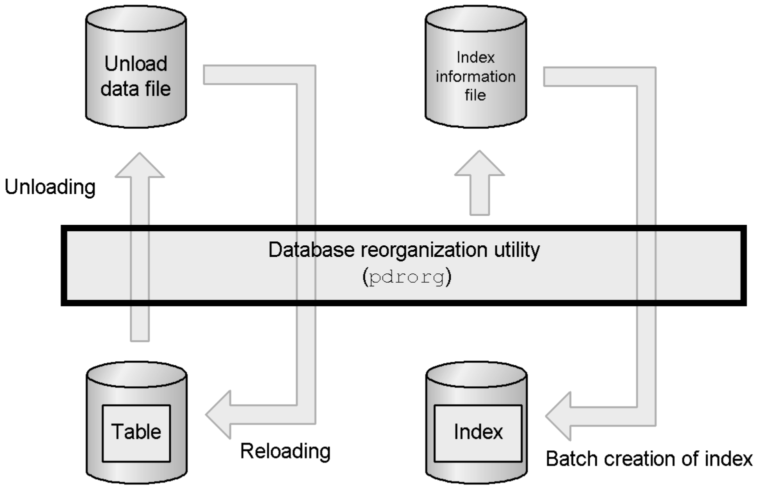
It is recommend that the table owner or the DBA privilege holder execute periodically the *database reorganization utility* (`pdrorg`) to reorganize tables and indexes.

## 7.3.1 Table reorganization

Deleting data does not release the segments and pages in which the deleted data has been stored. Therefore, when deletion of data has been performed many times, the amount of dead space in the table becomes significant, resulting in a decrease in data storage efficiency. Eventually, this can mean a shortage of space for RDAREAs, even though there has not been an increase in the amount of data.

Similarly, if data is added repeatedly, problems arise with respect to data not being stored in pages near the cluster key and to proliferation in the number of data I/O operations. The result is degradation of data retrieval performance. Executing the database reorganization utility (`pdrorg`) to reorganize a table causes the system to re-store the table's data, which can prevent these problems from arising. The following figure illustrates table reorganization.

Figure 7–9: Table reorganization



**Explanation**

- First, the table is stored temporarily in an unload file; this process is called *table unloading*. Subsequently, the data is re-stored in the table; this process is called *table data reloading*. The entire process is called *table reorganization*.
- If an index is defined for the table, the index information is output to an index information file when the data is reloaded. Based upon this information, HiRDB re-creates the index in the batch mode, which reorganizes the index as well.

### (1) Execution units for table reorganization

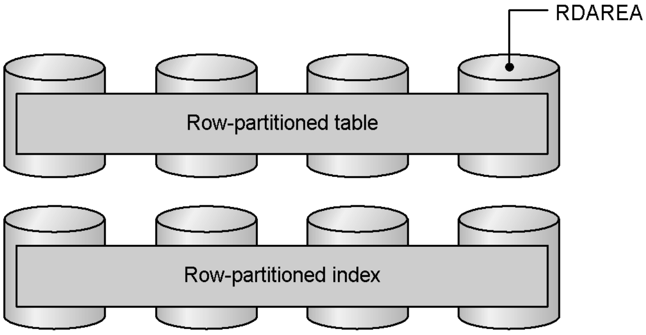
Table reorganization can be executed in the following units:

- By table
- By RDAREA
- By schema

#### (a) Reorganization by table

Reorganization processing can be performed on an entire table. Use this method as a normal rule. To do so, first execute the database condition analysis utility to determine whether the entire table needs to be reorganized. If so, you can then execute reorganization of the entire table. The following figure illustrates reorganization of an entire table.

Figure 7–10: Reorganization of an entire table



**Note**

The data indicated by shading is subject to reorganization.

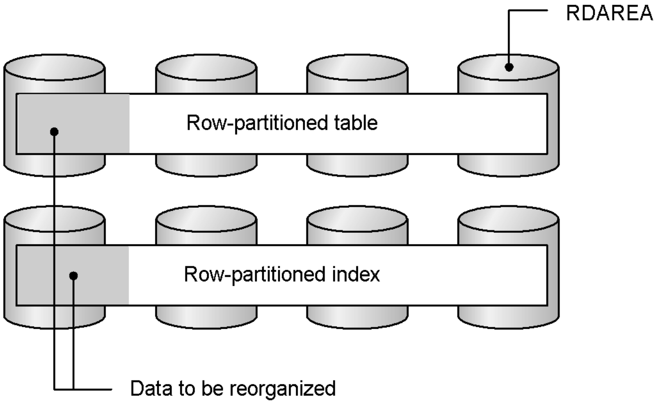
**Explanation**

The table to be reorganized is specified in the `-t` option of the database reorganization utility.

### (b) Reorganization by RDAREA

Reorganization processing is performed on a per-RDAREA basis. This method can be used only if the table is row-partitioned. Reorganization of RDAREAs is executed when the results of the database condition analysis utility indicate that it would suffice to reorganize only a portion of a row-partitioned table. This reduces the processing time compared with reorganization of the entire table. The following figure illustrates reorganization of an RDAREA.

Figure 7–11: Reorganization of an RDAREA



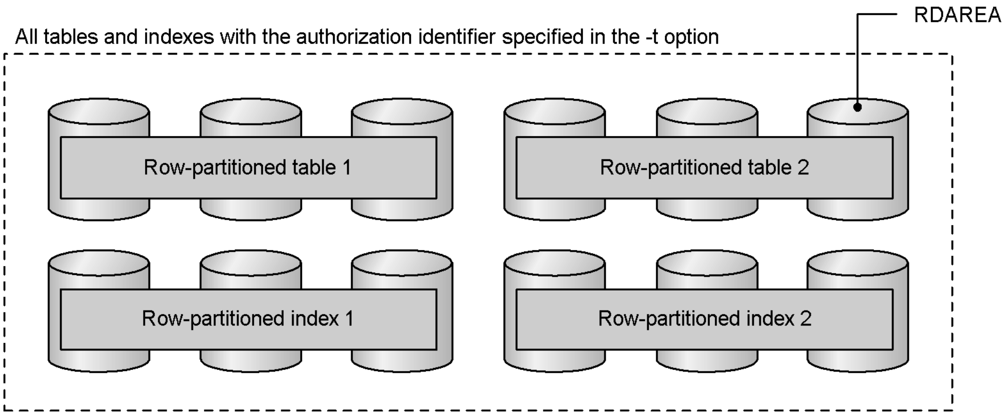
**Explanation**

The table to be reorganized is specified in the `-t` option and the RDAREAs to be reorganized are specified in the `-r` option of the database reorganization utility.

### (c) Reorganization by schema

This processing reorganizes all tables in a schema in the batch mode. Reorganization by schema can be used when you wish to reorganize all the tables you own on a batch basis. The following figure illustrates reorganization of a schema.

Figure 7–12: Reorganization of a schema



Note: All data is reorganized.

**Explanation**

The authorization identifier of the schema to be reorganized is specified in the `-t` option of the database reorganization utility. The specification format is: `-t` *authorization-identifier*`.all`.

## (2) Reorganizing a table containing a large quantity of data

When a table containing a large quantity of data is to be reorganized, you must consider whether to perform *reorganization with synchronization points set*.

Normally, while a table is being reorganized, transactions cannot be reconciled until storage processing of all the data has been completed. This means that synchronization point dumps cannot be obtained during execution of the database reorganization utility. If HiRDB terminates abnormally during reorganization of a large quantity of data, it will take a long time to restart HiRDB. To resolve this problem, you can set synchronization points at intervals of any number of data items during storage of the data (reload processing) in order to reconcile transactions. This is called reorganization with synchronization points set.

To perform reorganization with synchronization points set, you must specify a *synchronization point lines count*, which is the number data items to be stored before a synchronization point is set. This value is specified in the `option` statement of the database reorganization utility.

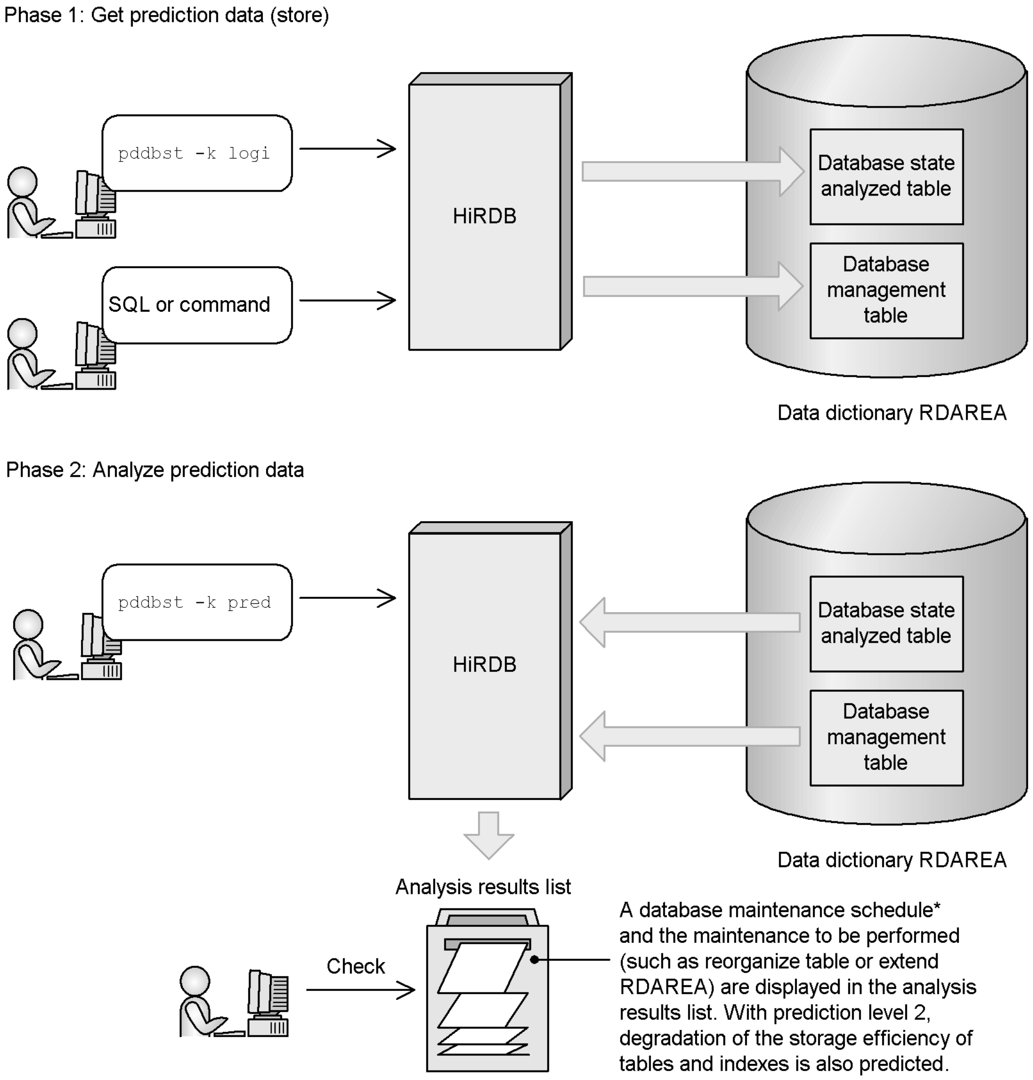
Synchronization point setting can also be specified in the database load utility; this is called *data loading with synchronization points set*.

## (3) Facility for predicting reorganization time

The decision on whether to reorganize tables or indexes or whether to extend an RDAREA must be made by the user based on using messages that are output and the execution results of the `pddbst` command to make a comprehensive evaluation of which tables to reorganize and when to reorganize them. The user might possibly reorganize a table that does not need to be reorganized or might neglect to reorganize a table that does need reorganizing because of an overlooked message that was output.

To simplify this operation, HiRDB is now able to predict when reorganization will be necessary. The function that performs this prediction is called the facility for predicting reorganization time. The following figure provides an overview of this facility.

Figure 7–13: Overview of facility for predicting reorganization time


Phase 1: Get prediction data (store)
pddbst -k logi
SQL or command
HiRDB
Database state analyzed table
Database management table
Data dictionary RDAREA
Phase 2: Analyze prediction data
pddbst -k pred
HiRDB
Database state analyzed table
Database management table
Data dictionary RDAREA
Analysis results list
Check
A database maintenance schedule* and the maintenance to be performed (such as reorganize table or extend RDAREA) are displayed in the analysis results list. With prediction level 2, degradation of the storage efficiency of tables and indexes is also predicted.

#: A day on which RDAREA maintenance needs to be performed is called a *scheduled database maintenance day*.

Predicting when reorganization will be needed is divided into two phases:

- Phase 1: Get reorganization prediction data
  - The pddbst command is executed periodically, and analysis results from the database are accumulated in the *database state analyzed table*.
  - Whenever SQL code or a command is executed, an operation log from the database is output to the *database management table*.
- Phase 2: Analyze reorganization prediction data
  Using the database state analyzed table and database management table as the input information, the pddbst command is used to analyze the reorganization prediction data. The user checks the execution results of the pddbst command and performs one of the following operations as needed:
  - Uses the pdrorg command to reorganize tables or indexes
  - Uses the pdreclaim command to release used free pages and free segments
  - Uses the pdmod command to extend an RDAREA
  - Uses the pdmod command to automatically extend an RDAREA

- Uses the `pdmod` command to reinitialize an RDAREA.

The facility for predicting reorganization time also provides the two levels described below. Prediction level 1, which predicts the time required for `pddbst` to accumulate database analysis results, can be run in a relatively short amount of time. However, Prediction level 2 might require a much longer time to run.

- Prediction level 1

  This level predicts when an RDAREA will no longer have sufficient free space. This level is designed for users who do not want to reorganize frequently while there is still sufficient free space for the RDAREA.
- Prediction level 2

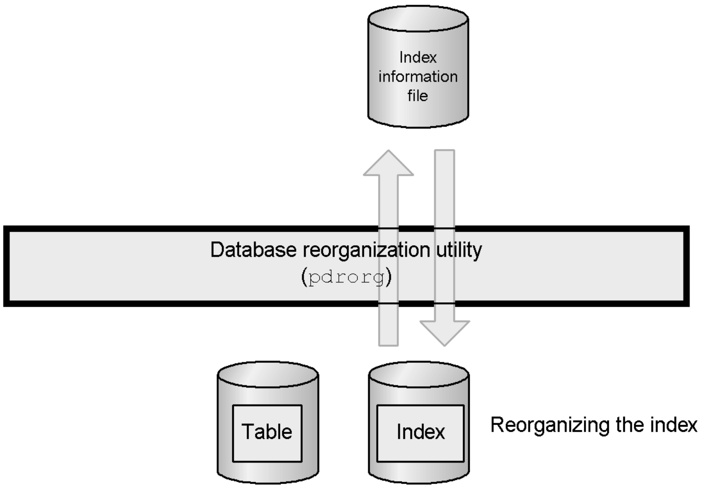
  In addition to predicting when an RDAREA will no longer have sufficient free space, this level predicts degradation of storage efficiency in tables and indexes. This level is designed for users who want to reorganize as soon as table or index access performance is reduced, regardless of the amount of free space for the RDAREA.

For details about the facility for predicting reorganization time, see the *HiRDB Version 9 System Operation Guide* and the section about the database condition analysis utility in the manual *HiRDB Version 9 Command Reference*.

## 7.3.2 Index reorganization

It is possible to reorganize an index only. The following figure illustrates index reorganization.

Figure 7–14: Index reorganization



**Explanation**

*Index reorganization* involves retrieving index key information, creating an index information file, and rearranging the index based on this information. Index reorganization can be executed by index or by RDAREAs that store an index.

### (1) Applicability and application criteria

Index reorganization can be applied to normal indexes only; it cannot be used to reorganize plug-in indexes. Index reorganization is performed to release dead space in index storage pages that has been generated by addition, deletion, or updating of large amounts of data.

### (2) Index reorganization and its use

- When data updating (`UPDATE`) has been performed frequently, reorganization of the indexes only is recommended.
- When data deletion (`DELETE`) or addition (`INSERT`) has been performed frequently, reorganization of the table is recommended.

- If there is not enough time to reorganize a table, you can reorganize the index only in order to reduce index search time.

## (3) Differences from re-creation of an index

Index re-creation involves a search of the table data; index reorganization does not involve a search of the table data. For this reason, index reorganization requires less processing time than re-creation,[#] sorting is not required, and processing performance is improved.

#: Processing time is reduced if the following condition is satisfied:

*Number of pages used in table storage RDAREA > Number of pages used in index storage RDAREA*

## (4) Notes on index reorganization

Before reorganizing simultaneously multiple indexes that are stored in the same RDAREA, use the `pdhold` command to place the RDAREA in shutdown status. When reorganization processing has been completed, you can use the `pdrels` command to release the RDAREA from shutdown status.

## (5) Technique for reducing reorganization execution time

The processing time involved in index reorganization can be reduced by not collecting database update log information (setting either the no-log mode or the pre-update log acquisition mode in the `-l` option of the database reorganization utility (`pdrorg`)).

## (6) Reorganizing an index in an RDAREA that has insufficient free space

The amount of unused area per page is specified in the `PCTFREE` operand of `CREATE TABLE` or `CREATE INDEX` when the index is reorganized is applied. However, if the index being reorganized is in an RDAREA that has insufficient free space to accommodate the specified percentage of unused area, the RDAREA can run out of space during index reorganization processing. To prevent this, specify the `idxfree` operand in the `option` statement of the database reorganization utility (`pdrorg`), and use the `PCTFREE` operand of `CREATE TABLE` or `CREATE INDEX` to change the percentage of unused area per page.

Note that this is a temporary measure that is used during reorganization. In normal operations, expand the RDAREA by executing the database structure modification utility (`pdmod`).

# 7.4 Reusing used free pages and used free segments

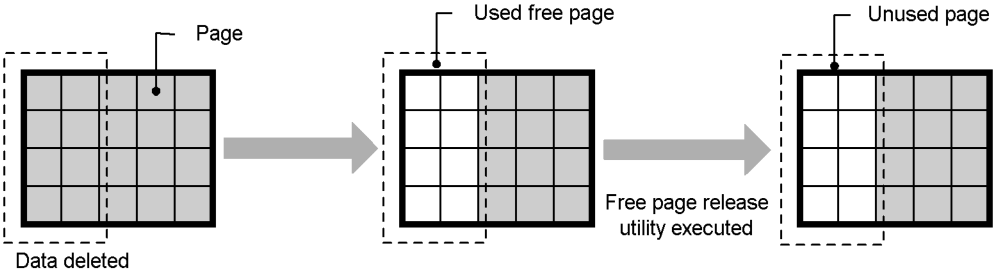
You can reuse used free pages in tables and indexes by converting them to unused pages. Similarly, you can reuse used free segments by converting them to unused segments. Before reading the descriptions in this section, however, you must be familiar with page statuses and segment statuses. For details about page statuses, see *4.3 Page design*. For details about segment statuses, see *4.2 Segment design*.

## 7.4.1 Reusing used free pages

### (1) Releasing used free pages

When a large amount of table data is deleted by a batch job or some other operation, some of the pages in which that table data is being stored (data pages) might become used free pages. Similarly, when an index is defined, some of the pages in which index key values are being stored (index pages) also become used free pages. By executing the *free page release utility* (`pdreclaim`), you can convert used free pages to unused pages, and reuse them. This is called *releasing used free pages*. The following figure shows how used free pages are released.

Figure 7–15: Releasing used free pages



Tip

- You cannot release used free pages of data stored in LOB RDAREAs.
- You cannot release used free pages of plug-in indexes.

For details about releasing used free pages, see the *HiRDB Version 9 System Operation Guide*.

### (2) Benefits gained from releasing used free pages

#### (a) Benefits gained from releasing used free pages of tables

The following table lists and describes the benefits gained from releasing used free pages of a table.

Table 7–4: Benefits gained from releasing used free pages of a table

| Benefit | Description | Degree of benefit |
|---|---|---|
| Ability to increase the table reorganization cycle | The ability to reuse used free pages improves data storage efficiency. In turn, this can increase (improve) the table reorganization time cycle. | Good |
| Improvement in performance when searching large data sets | Because they are a type of used page, used free pages are searched. However, searching is not performed on unused pages (they are skipped by the search). Thus, converting to unused pages improves the search performance in direct proportion to the ratio converted. The benefits are particularly evident when large data sets are searched. | Varies |
| Improvement in performance when INSERT and UPDATE are used. | If sufficient contiguous free space cannot be allocated when an attempt is made to save data to a used page, HiRDB performs a process called *page compaction*. Page compaction refers to a process whereby data in the affected page is repacked to | Varies |

| Benefit | Description | Degree of benefit |
|---|---|---|
| | secure sufficient contiguous free space in which to store the new data. When free used pages are released, page compaction is also performed at the same time. This eliminates the need to perform page compaction, which prolongs INSERT and DELETE processing, and improves performance by the corresponding amount. The pages on which page compaction is performed are used pages other than full pages and used free pages. | |
| Potential reduction in errors when INSERT and UPDATE are performed on branch rows. | If you execute INSERT or UPDATE on a branch row when there are no unused pages, an error occurs (KFPA11756-E message). The increase in unused pages due to the release of used free pages tends to reduce the frequency of this error. | Varies |

Legend:

Good: Always beneficial.

Varies: The degree of benefit varies depending on conditions.

### (b) Benefits gained from releasing used free pages of indexes

The following table lists and describes the benefits gained from releasing used free pages of an index.

Table 7–5: Benefits gained from releasing used free space of an index

| Benefit | Description | Degree of benefit |
|---|---|---|
| Potential reduction of insufficient capacity in RDAREAs storing indexes | If capacity runs out even though free pages (used free pages) exist, release the used free pages. Note that this tends to reduce the occurrence of insufficient capacity in RDAREAs that store indexes with key values that are frequently updated or deleted. | Great |
| Ability to increase the index reorganization cycle | The ability to reuse used free pages improves data storage efficiency. In turn, this can increase (improve) the index reorganization time cycle. | Good |
| Improvement in performance when searching large data sets that use indexes | Because they are a type of used page, used free pages are searched. However, searching is not performed on unused pages (they are skipped by the search). Thus, converting to unused pages improves the search performance in direct proportion to the ratio converted. The benefits are particularly evident when large data sets are searched. | Varies |

Legend:

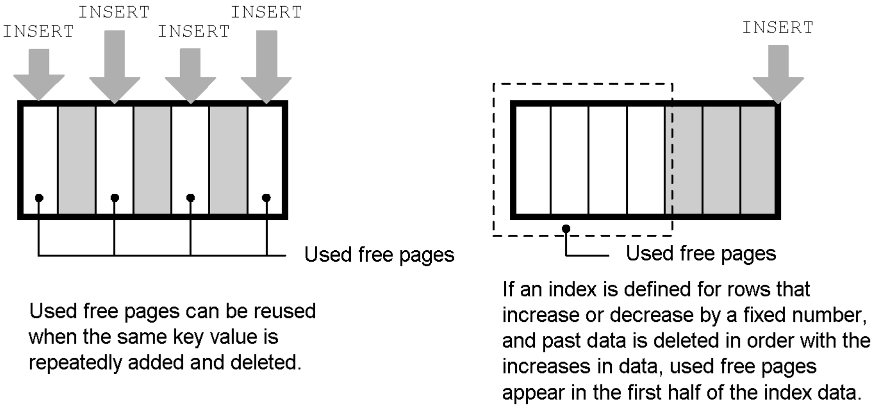
Great: Particularly beneficial.

Good: Always beneficial.

Varies: The degree of benefit varies depending on conditions.

Using this utility is particularly beneficial when deleted key values are not re-registered. Because used free pages are reused when the same key value is added or deleted repeatedly, large numbers of used free pages do not appear. However, for indexes that are defined for rows that increase or decrease by a fixed number (such as date, sequence number, and so on), and if past data is deleted in order with the increases in data, a large number of used free pages that are not reused appear in the first half of the index pages. The following figure shows the processing of a used free page being created for index pages.

Figure 7–16: Process of used free pages being created for index pages


INSERT
INSERT
INSERT
INSERT
INSERT
Used free pages
Used free pages
Used free pages can be reused when the same key value is repeatedly added and deleted.
If an index is defined for rows that increase or decrease by a fixed number, and past data is deleted in order with the increases in data, used free pages appear in the first half of the index data.

Note that, after the used free pages are released, the key values are stored in the released pages, which improves data storage efficiency.

### (3) Differences from table or index reorganization

From the standpoint of performance and data storage efficiency, reorganizing tables or indexes is superior to releasing used free pages. However, while releasing used free pages, you can still access the tables or indexes on which the utility is operating. With reorganization, you cannot access the tables or indexes on which the utility is operating. This means that you do not need to interrupt normal operations when releasing free pages.

Use the execution results of the database condition analysis utility to evaluate whether to reorganize tables or indexes, or to release used free pages. The following lists the evaluation criteria:

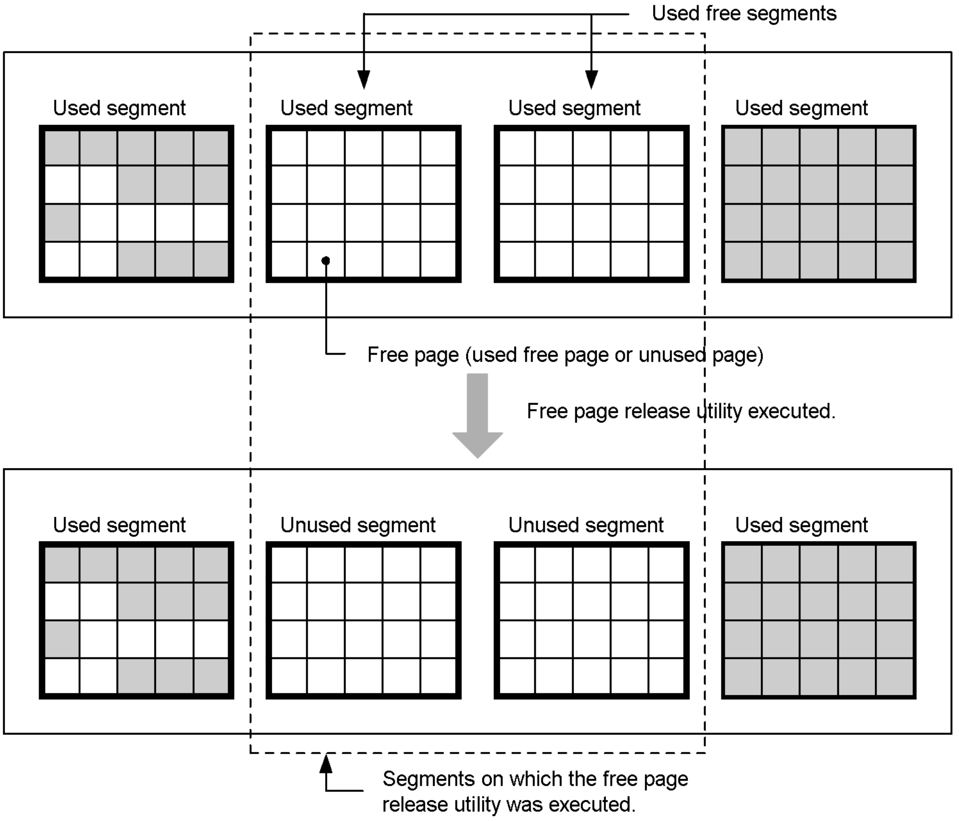
- If there is a large number of used free pages, release the used free pages.
- If the page utilization ratio of the used pages differs significantly from the segment free page ratio (value of the `PCTFREE` operand of `CREATE TABLE`), perform reorganization.

## 7.4.2 Reusing used free segments

### (1) Releasing used free segments

By executing the free page release utility, you can convert used free segments to unused segments, and reuse them. This is called *releasing used free segments*. The following figure shows how used free segments are released.

Figure 7–17: Releasing used free segments


Used free segments
Used segment
Used segment
Used segment
Used segment
Free page (used free page or unused page)
Free page release utility executed.
Used segment
Unused segment
Unused segment
Used segment
Segments on which the free page release utility was executed.

For details about releasing used free segments, see the *HiRDB Version 9 System Operation Guide*.

## (2) Benefits gained from releasing used free segments

Once segments are allocated, only the table (or index) that is assigned to use a particular segment can use it; no other table can do so. Releasing used free segments converts used free segments to unused segments, which allows other tables to use them.

# 7.5 Adding, expanding, and moving RDAREAs

As the scale of operations increases, the amount of data in your databases increases as well. However, even if a database becomes larger than your original estimate, with HiRDB, you can subsequently add, expand, or move the database. With a HiRDB parallel server configuration, you can also add a server machine and add or move a database onto the new server machine.

HiRDB provides the following functions to support increases in the amount of data in your databases:

- Adding RDAREAs
- Expanding RDAREAs
- RDAREA automatic extension
- Moving RDAREAs (HiRDB parallel server configurations only)

## 7.5.1 Adding RDAREAs

RDAREAs can be added with the `create rdarea` statement of the database structure modification utility (`pdmod`). RDAREAs must be added when a new table is created.

To use a newly added RDAREA, it is necessary to allocate a global buffer to it. For this purpose, you should use the `pdbufls` command to check for an available (previously defined) global buffer.

## 7.5.2 Expanding an RDAREA

As more and more data is added to a table, the available space in its RDAREAs becomes smaller and smaller. When the amount of remaining space becomes too small, it is possible to expand RDAREAs with the `expand rdarea` statement of the *database structure modification utility* (`pdmod`).

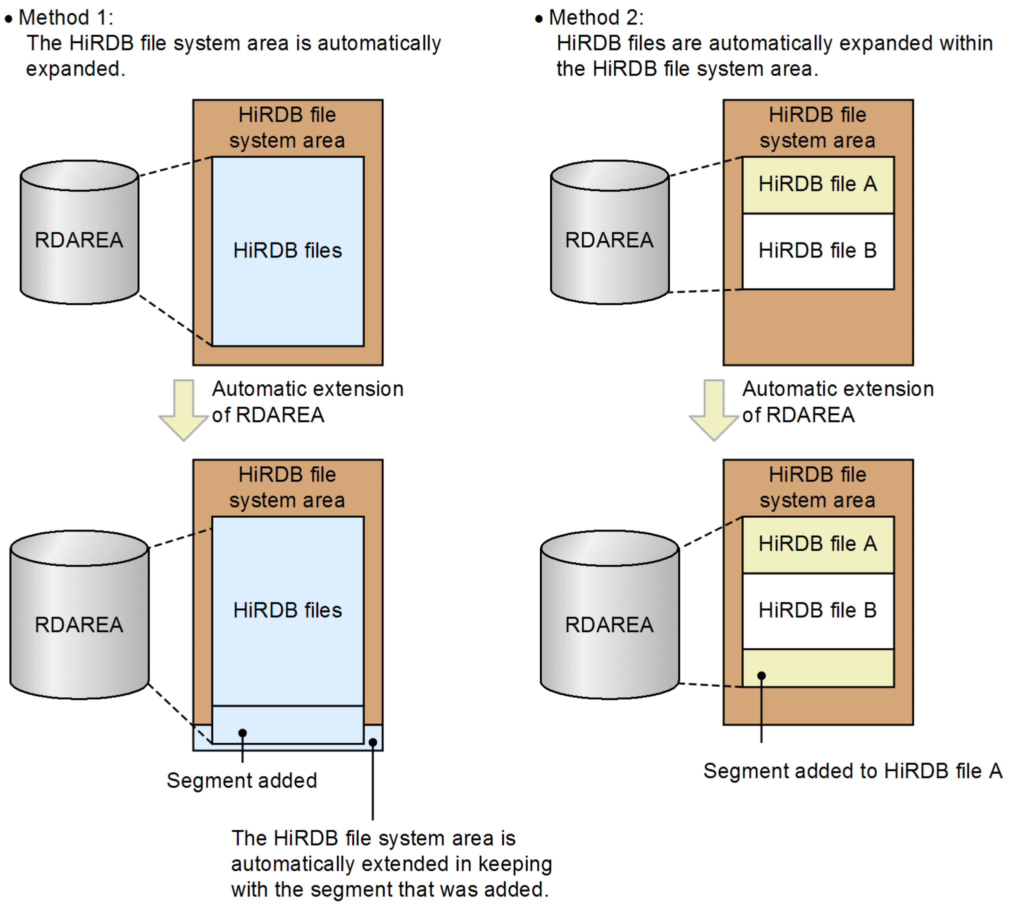
## 7.5.3 RDAREA automatic extension

When a space shortage is about to occur in an RDAREA, the size of the RDAREA can be expanded by adding segments to the HiRDB files of that RDAREA. This is called *RDAREA automatic extension*. The following two methods are available for automatically extending an RDAREA:

1. Automatically expanding the HiRDB file system area
2. Expanding HiRDB files within the HiRDB file system area

The following figure illustrates RDAREA automatic extension.

Figure 7–18: RDAREA automatic extension


• Method 1:
The HiRDB file system area is automatically expanded.
• Method 2:
HiRDB files are automatically expanded within the HiRDB file system area.
RDAREA
HiRDB file system area
HiRDB files
HiRDB file A
HiRDB file B
Automatic extension of RDAREA
Automatic extension of RDAREA
RDAREA
HiRDB file system area
HiRDB files
RDAREA
HiRDB file system area
HiRDB file A
HiRDB file B
Segment added
Segment added to HiRDB file A
The HiRDB file system area is automatically extended in keeping with the segment that was added.

**Explanation**

In *Method 1*, when adding a segment causes the HiRDB file system area to exceed its upper size limit, the HiRDB file system area is automatically expanded to the necessary size. With this method, you can create only one HiRDB file in each HiRDB file system area. The single HiRDB file that can be created is allocated to an RDAREA as the last HiRDB file area comprising the RDAREA.

In *Method 2*, HiRDB files are automatically expanded or added within the HiRDB file system area, up to its upper size limit.

For details about RDAREA automatic extension, see the *HiRDB Version 9 System Operation Guide*.

## (1) RDAREAs eligible for automatic extension

Automatic extension can be applied to the following RDAREAs:

- Data dictionary RDAREAs
- User RDAREAs
- Registry RDAREA
- Data dictionary LOB RDAREAs
- User LOB RDAREAs
- Registry LOB RDAREA

## (2) Automatic extension triggers

HiRDB automatically extends RDAREAs when any of the automatic extension triggers described below occur.

- When the number of free segments within a RDAREA has fallen to or below the number of segments that are set to be extended in a single automatic extension

  For example, when the number of segments that are extended in a single automatic extension is set to 50, if the number of free segments falls to or below 50 segments, automatic extension occurs.
- When there are no free segments within an RDAREA and a new segment cannot be allocated

You can specify an automatic extension trigger in the `pd_rdarea_extension_timing` operand. For details, see the manual *HiRDB Version 9 System Definition*.

## (3) Criteria for selecting an RDAREA automatic extension method

To apply RDAREA automatic extension, we generally recommend that you select Method 1. Because Method 1 automatically expands the HiRDB file system area to the maximum HiRDB file size (64 gigabytes), it is easy to estimate the area size and expand the database.

However, because Method 1 is subject to the following restrictions, select Method 2 for systems to which Method 1 cannot be applied.

- Because a HiRDB file system area is allocated to each RDAREA, the same number of HiRDB file system areas as the number of RDAREAs must be created.
- Because a HiRDB file system area is automatically expanded to 64 gigabytes as long free space is available on the disk, you cannot set an upper limit for disk usage.

## (4) Setting automatic extension

The following subsections describe the procedures for setting automatic extension.

### (a) Method 1

With this method, if RDAREA automatic extension causes the HiRDB file system area to exceed its upper size limit, the HiRDB file system area is automatically extended to the maximum HiRDB file size (64 gigabytes).

**Procedure**

1. If necessary, change the HiRDB system definition.

   Specify an automatic extension trigger in the `pd_rdarea_extension_timing` operand. For details about how to change the HiRDB system definition, see the *HiRDB Version 9 System Operation Guide*.
2. When you create the HiRDB file system area using the `pdfmkfs` command, specify the `-a` option.
3. When you create the RDAREA, use a utility control statement# to specify the number of segments by which the RDAREA can be extended.

#

This can be specified in the `CREATE RDAREA`, `EXPAND RDAREA`, `INITIALIZE RDAREA`, or `ALTER RDAREA` statement of the database initialization utility, the database structure modification utility, or the registry facility initialization utility.

### (b) Method 2

With this method, RDAREAs are automatically extended to the upper size limit set for the HiRDB file system area. If the HiRDB file system area has no free space available, automatic extension cannot occur. In such a case, either extend the RDAREA or reorganize the tables and indexes within it. If the number of extents exceeds the upper limit of 24, combine the extents of the HiRDB file system area into a single extent, or extend the RDAREA. For details about how to combine extents and HiRDB file system areas, see the *HiRDB Version 9 System Operation Guide*.

**Procedure**

1. If necessary, change the HiRDB system definition and specify an automatic extension trigger in the `pd_rdarea_extension_timing` operand. For details about how to change the HiRDB system definition, see the *HiRDB Version 9 System Operation Guide*.
2. When you create the HiRDB file system area using the `pdfmkfs` command, specify the maximum number of extensions (in the `-e` option).

3. When you create the RDAREA, use a utility control statement# to specify the number of segments by which the RDAREA can be extended.

#

This can be specified in the `CREATE RDAREA`, `EXPAND RDAREA`, `INITIALIZE RDAREA`, or `ALTER RDAREA` statement of the database initialization utility, the database structure modification utility, or the registry facility initialization utility.

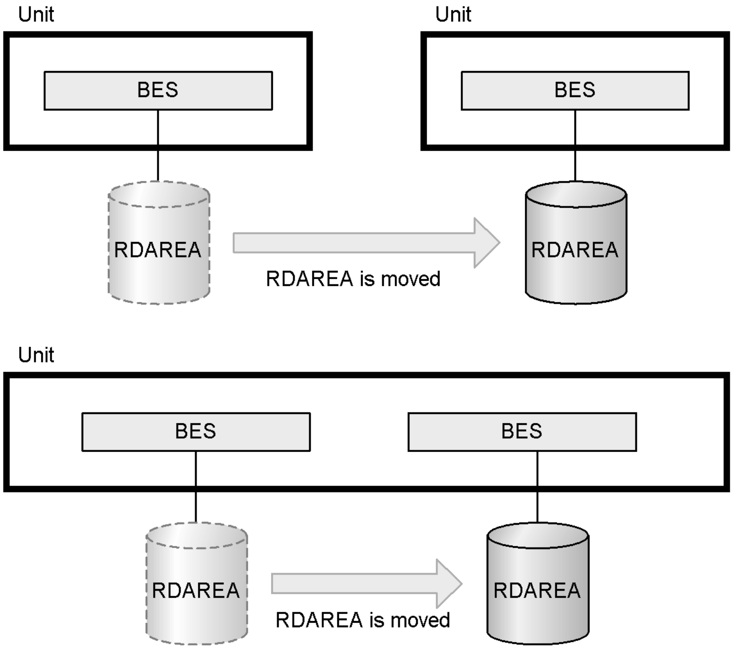
## 7.5.4 Moving RDAREAs (HiRDB parallel server configurations only)

You can use the `move rdarea` statement of the database structure modification utility (`pdmod` command) to move an RDAREA to another back-end server. The capability of moving an RDAREA is provided only by a HiRDB parallel server configuration. The following lists the RDAREAs that can be moved:

- User RDAREAs
- User LOB RDAREAs

The following figure provides an overview of moving RDAREAs.

Figure 7–19: Moving RDAREAs



For details about moving RDAREAs, see the *HiRDB Version 9 System Operation Guide*.

# 7.6 Space conversion facility

In a data comparison, one double-byte space character and two single-byte space characters will be recognized as different data. Therefore, table data containing a mixture of double-byte and single-byte space characters can produce incorrect retrieval results.

**Example**

The following data are recognized as being different:

TV ▯▯21-inch
TV □21-inch

where
▯▯ : indicates two single-byte spaces
□ : indicates one double-byte spaces

The space conversion facility enables double-byte space characters and single-byte space characters to be intermixed in table data.

The double-byte space character being discussed here is coded as shown as follows. Two single-byte space characters are coded as `X'2020'`.

- Shift-JIS Kanji Code: `X'8140'`
- EUC Japanese Kanji Code: `X'A1A1'` (UNIX edition only)
- EUC Chinese Kanji Code: `X'A1A1'`
- Chinese Kanji Code (GB18030)#: `X'A1A1'`
- Unicode (UTF-8) or Unicode (IVS-supported UTF-8)#: `X'E38080'`

#: `NCHAR` and `NVARCHAR` cannot be used if the character encoding is Unicode (UTF-8), Unicode (IVS-supported UTF-8), or Chinese Kanji Code (GB18030).

## (1) Space conversion levels

As shown in the following table, three levels of space character conversion are provided by the space conversion facility.

Table 7–6: Space conversion levels

| Level | Explanation |
|---|---|
| Level 0 | No space conversion. |
| Level 1 | Converts as follows data spaces that occur in literals, embedded variables, and `?` parameters in the data manipulation SQL or data spaces that are stored by utilities:<br>• When a character string literal is being handled as a national character string literal, two single-byte space characters in succession are converted into one double-byte space character; in the case of a single occurrence of a single-byte space, no conversion is performed.<br>• When a character string literal is being handled as a mixed character string literal, one double-byte space character is converted into two single-byte space characters.<br>• During storage of data in a national character string-type column or during comparison of data with a national character string-type value expression, two single-byte space characters in succession in an embedded variable or `?` parameter are converted into one double-byte space character; in the case of a single occurrence of a single-byte space, no conversion is performed.<br>• During storage of data in a mixed character string-type column or during comparison of data with a mixed character string-type value expression, one double-byte space character is converted into two single-byte space characters. |
| Level 3 | Adds the following processing to the processing of space conversion level 1: |

| Level | Explanation |
|---|---|
| | • During retrieval of data in a national character string-type value expression, one double-byte space character is converted into two single-byte space characters. |

The following figures illustrate Level 1 and Level 3 processing.

Figure 7–20: Level 1 processing

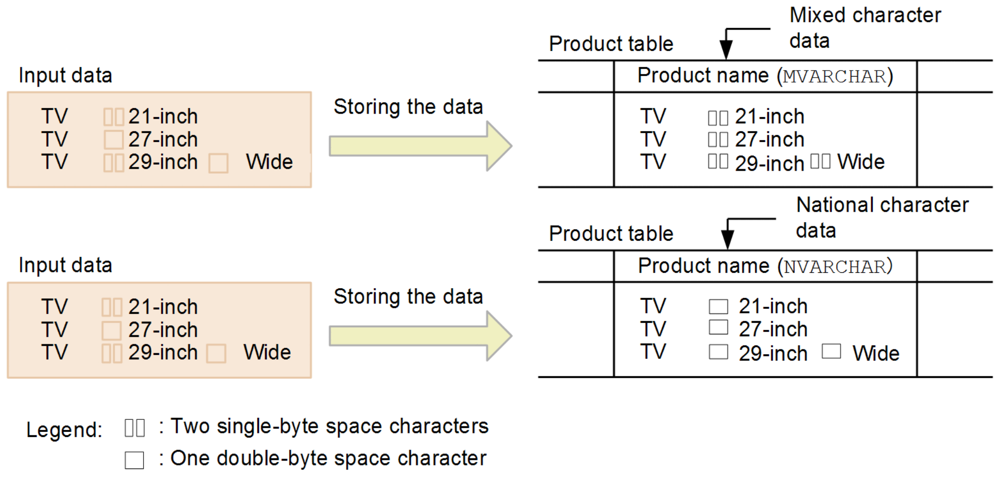
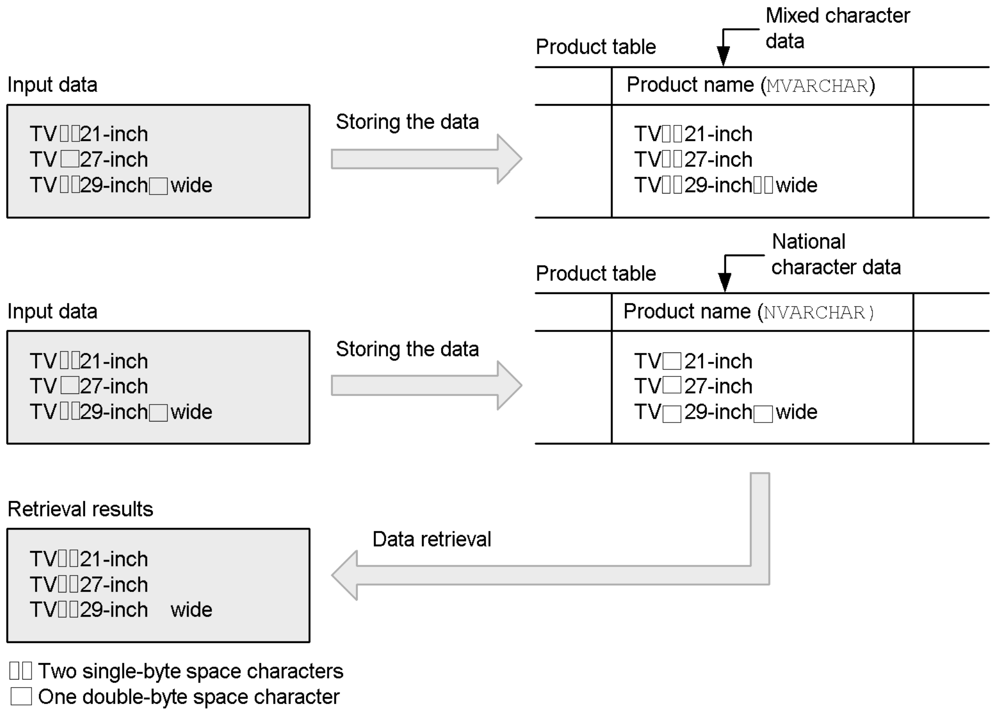


Figure 7–21: Level 3 processing



## (2) Setting the space conversion level

The desired space conversion level can be specified in the following operands:

- `pd_space_level` operand in the system common definition
- `PDSPACELVL` operand in the client environment definition

- `spacelvl` operand in the `option` statement of the database load utility (`pdload`)
- `spacelvl` operand in the `option` statement of the database reorganization utility (`pdrorg`)

For details about using the space conversion facility, see the *HiRDB Version 9 System Operation Guide*.

# 7.7 Facility for conversion to a decimal signed normalized number

The decimal, date interval, and time interval data formats are signed packed formats consisting of a value's integer and sign parts. Normally, HiRDB handles X'C' (positive), X'D' (negative), and X'F' (positive) as the sign parts of signed packed-format data as valid values, and stores directly in the database any signs that are input from a UAP or utility.[#] In addition, HiRDB treats +0 (sign part X'C' or X'F') and -0 (sign part X'D') as distinct values.

Use of the facility for conversion to a decimal signed normalized number enables conversion of the sign part of a signed packed decimal, date interval, or time interval data format.

#: Type conversions and operations during the execution of SQL statements sometimes result in conversion of signs. In addition, the use of multicolumn indexes can result in conversion of signs.

## (1) Specification of the sign part of signed packed format

The following table lists the specifications of the sign part of signed packed format in HiRDB.

Table 7–7: Specification of the sign part of signed packed format

| Sign part | Meaning |
|---|---|
| `X'C'` | Indicates a positive value. |
| `X'D'` | Indicates a negative value. |
| `X'F'` | Indicates a positive value. |

## (2) Rules on conversion of the sign part of signed packed format

When the facility for conversion to a decimal signed normalized number is used, HiRDB converts the sign part of signed packed format during data input, according to the rules described in the tables below. The process of converting a sign part is called *normalization*. When the sign part is converted, the codes `+0` and `-0` can be treated as being the same value.

Table 7–8: Rules for conversion of the sign part of signed packed format (other than "0" data)

| Sign part of embedded variable | Not normalized | Normalized |
|---|---|---|
| `X'A'` | Error | Converts to `X'C'`. |
| `X'B'` | Error | Converts to `X'D'`. |
| `X'C'` | No conversion | No conversion |
| `X'D'` | No conversion | No conversion |
| `X'E'` | Error | Converts to `X'C'`. |
| `X'F'` | No conversion | Converts to `X'C'`. |
| `X'0'` to `X'9'` | Error | Error |

Table 7–9: Rules for conversion of the sign part of signed packed format ("0" data)

| Sign part of "0" data | Not normalized | Normalized |
|---|---|---|
| `X'A'` | Error | Converts to `X'C'`. |
| `X'B'` | Error | |
| `X'C'` | No conversion | |
| `X'D'` | No conversion | |

| Sign part of "0" data | Not normalized | Normalized |
|---|---|---|
| X'E' | Error | |
| X'F' | No conversion | |

## (3) Application criteria

When UAPs with different sign part specifications are used, it might be beneficial to use the facility for conversion to a decimal signed normalized number. In such a case, use this facility after the sign conversion rules are adequately confirmed.

For example, if a UAP is transferred from XDM/RD E2 to HiRDB, it might be beneficial in some situations to use the facility for conversion to a decimal signed normalized number, because XDM/RD E2 and HiRDB have different specifications for the sign part of the decimal type.

## (4) Environment setup

To use the facility for conversion to a decimal signed normalized number, you must specify `pd_dec_sign_normalize=Y` in the system common definition.

Specify the facility for conversion to a decimal signed normalized number as soon as HiRDB is installed. If sign parts are to be normalized once operation of HiRDB is underway, it might be necessary to reload all table data in which the decimal type is defined.

For details about using the facility for conversion to a decimal signed normalized number, see the *HiRDB Version 9 System Operation Guide*.

# 8 Error-handling Facilities

This chapter describes error-handling facilities.

# 8.1 System switchover facility

Linking to a cluster software product allows you to use the *system switchover facility*, which is designed to improve system reliability and availability. This section provides an overview of the system switchover facility. For details about how to operate the system switchover facility, see the *HiRDB Version 9 System Operation Guide*.

## 8.1.1 Overview of the system switchover facility

If a failure occurs on the HiRDB that is actively processing jobs, job processing can be automatically switched over to the standby HiRDB. This ability is called the system switchover facility. Job processing is interrupted from the time the failure occurs to the time processing is switched over to the standby HiRDB. The system switchover facility is used to keep system downtime to a minimum when a failure occurs.

There are two types of system switchover facilities: the *standby system switchover facility* and the *standby-less system switchover facility*. Moreover, the system switchover facility can be operated in *monitor mode* or *server mode*, depending on the types of failures to be monitored. The following table shows the system switchover facilities and the modes in which they can be used.

Table 8–1: Types of system switchover facility and operation modes

| System switchover facility type | | Operation method | |
|---|---|---|---|
| | | Monitor mode | Server mode |
| Standby system switchover facility | Normal system switchover | Y | Y |
| | User server hot standby | N | Y |
| | Rapid system switchover | N | Y |
| Standby-less system switchover facility# | Standby-less system switchover (1:1) facility | N | Y |
| | Standby-less system switchover (effects distributed) facility | N | Y |

Legend:

Y: Can be used.

N: Cannot be used.

#

To use the standby-less system switchover facility, you must have HiRDB Advanced High Availability.

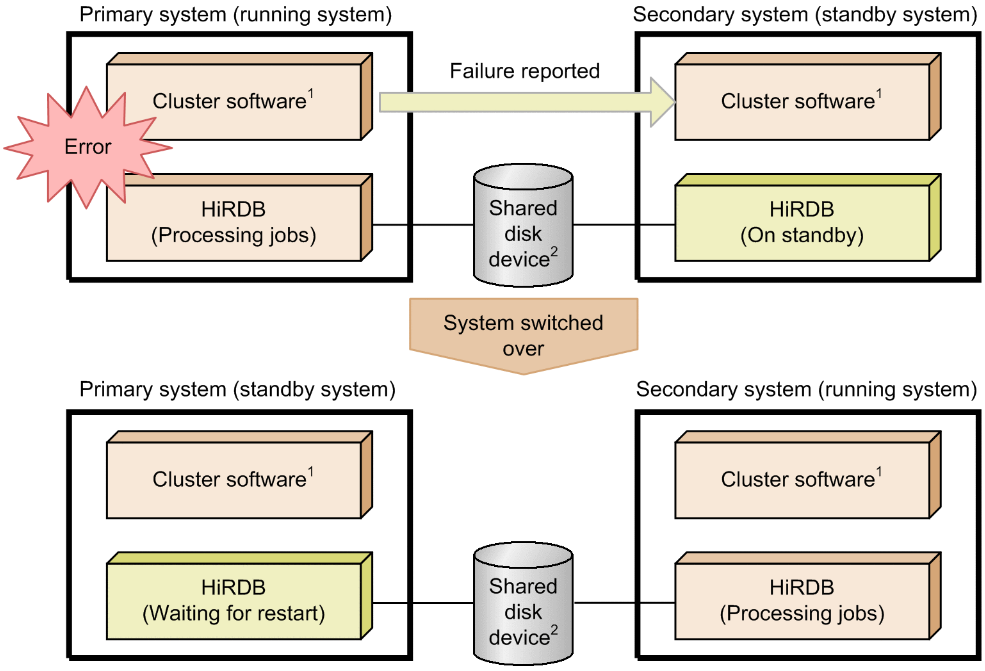
### (1) Overview of the standby system switchover facility

By deploying a standby HiRDB that is separate from the HiRDB actively processing jobs, if a failure occurs on the HiRDB that is actively processing jobs, job processing is automatically switched over to the standby HiRDB. This ability is called the *standby system switchover facility*.

To implement the standby system switchover facility, you need a cluster system configuration that uses multiple server machines. For a HiRDB single server configuration, the system is switched over on a per-system basis. However, note that system switchover cannot be performed for utility special units (UNIX edition only). For a HiRDB parallel server configuration, the system is switched over on a per-unit basis. Note that using the user server hot standby or rapid system switchover facility can shorten the time required for system switchover. For details about the user server hot standby or rapid system switchover facility, see *Reducing system switchover time (user server hot standby, rapid system switchover facility)* in the *HiRDB Version 9 System Operation Guide*.

The system on which jobs are currently being processed is called the *running system*, and the system that is currently in reserve is called the *standby system*. Whenever a system switchover occurs, the running system and the standby system are swapped. In addition, to distinguish between the two systems while you are building them and configuring the environments, the system that is initially started as the running system is called the *primary system*, and the system that is first started as the standby system is called the *secondary system*. Although the running and the standby systems change whenever a system switchover occurs, the primary system and secondary system do not. The following figure provides an overview of the system switchover facility (standby system switchover facility).

Figure 8–1: Overview of the system switchover facility (standby system switchover facility)



#1

In this manual, the product that performs the system switchover is referred to as *cluster software*. For details about the cluster software products supported by HiRDB, see *(3) Cluster software supported by HiRDB* below.

#2

For details about the shared disk unit, see *(4) Shared disk unit* below.

**Explanation**

If a failure occurs on the running system while it is processing jobs, the failure is reported to the standby system, and the system is switched over. The standby system becomes the running system and resumes job processing.

## (2) Overview of the standby-less system switchover facility

If a failure occurs on the HiRDB that is actively processing jobs, the system is switched over to another unit whose currently running back-end server takes over the processing. This is called the *standby-less system switchover facility*. In contrast with the standby system switchover facility, with the standby-less system switchover facility you do not have to allocate a standby unit.

The standby-less system switchover facility is further classified as follows:

- Standby-less system switchover (1:1) facility
- Standby-less system switchover (effects distributed) facility

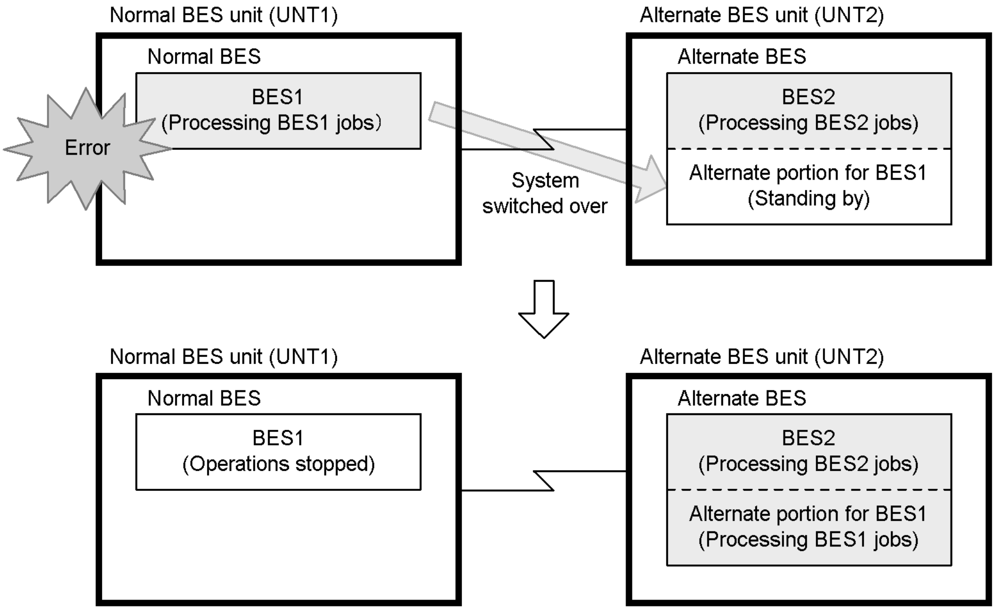
The standby-less system switchover facility can be used in a back-end server unit of a HiRDB parallel server configuration. It cannot be used in a unit in which a server other than a back-end server resides.

### (a) Standby-less system switchover (1:1) facility

With the *standby-less system switchover (1:1) facility*, there is a one-to-one relationship between the unit on which the failure occurs and the unit to whose back-end server processing is switched.

A back-end server whose processing is transferred to another unit when a failure occurs is called a *normal BES*, and a back-end server that takes over processing is called an *alternate BES*. Similarly, the unit containing the normal BES is called the *normal BES unit*, and the unit containing the alternate BES is called the *alternate BES unit*. The following figure provides an overview of the standby-less system switchover (1:1) facility.

Figure 8–2: Overview of the standby-less system switchover (1:1) facility



**Explanation**

- Normally, both BES1 and BES2 perform processing.
- If a failure occurs on the normal BES unit (UNT1), the system switches over, and processing is taken over by the alternate BES. The area in which processing is taken over is called the *alternate portion* and, when the alternate portion is performing processing, it is said to be *alternating*.
- After the failure is resolved and the normal BES unit is started, the processing taken over by the alternate BES is switched over to the normal BES, and returned to normal status. This is called *reactivating the system*.

**Remarks**

Using the concepts of the primary and other systems in the standby system switchover facility, consider the following with respect to the standby-less system switchover (1:1) facility:

- Think of the normal BES unit as the primary system, and the alternate BES unit as the secondary system.
- Under normal conditions, think of the normal BES unit as the running system, and the alternate portion as the standby system. During alternating, think of the alternate portion as the running system, and the normal BES unit as the standby system.

**Prerequisites**

To use the standby-less system switchover (1:1) facility, all of the following must be satisfied:

- HiRDB Advanced High Availability is installed.
- Hitachi HA Toolkit Extension is installed (not required if HA Monitor is the cluster software).
- The system switchover facility is running in server mode.

**Advantages of the standby-less system switchover facility**

The following describes the advantages of the standby-less system switchover facility over the standby system switchover facility:

- You do not need to set aside a standby system unit, which means you can use system resources more efficiently. However, remember that the load increases on the back-end server that takes over processing when the system is switched over, which might adversely affect processing performance.
- The server processes are already running, which allows you to reduce system switchover time so that it is about the same as when the rapid system switchover facility is used. For details about the rapid system switchover facility, see *8.1.5 Functions that reduce system switchover time (user server hot standby and the rapid system switchover facility)*.

### (b) Standby-less system switchover (effects distributed) facility

When a failure occurs, processing requests directed to back-end servers in the failed unit can be distributed to and executed in multiple active units. This is called the standby-less system switchover (effects distributed) facility. This facility enables you to use your system resources more efficiently, without having to allocate a standby server machine or a standby unit. Of course, there might be adverse effects on transaction processing performance because of the increased processing load on the units that have taken over processing for the servers in the failed unit. However, because the processing requests directed to the failed servers are distributed to and executed in a number of units, the workload increase for each unit is minimized, reducing overall degradation of system performance.

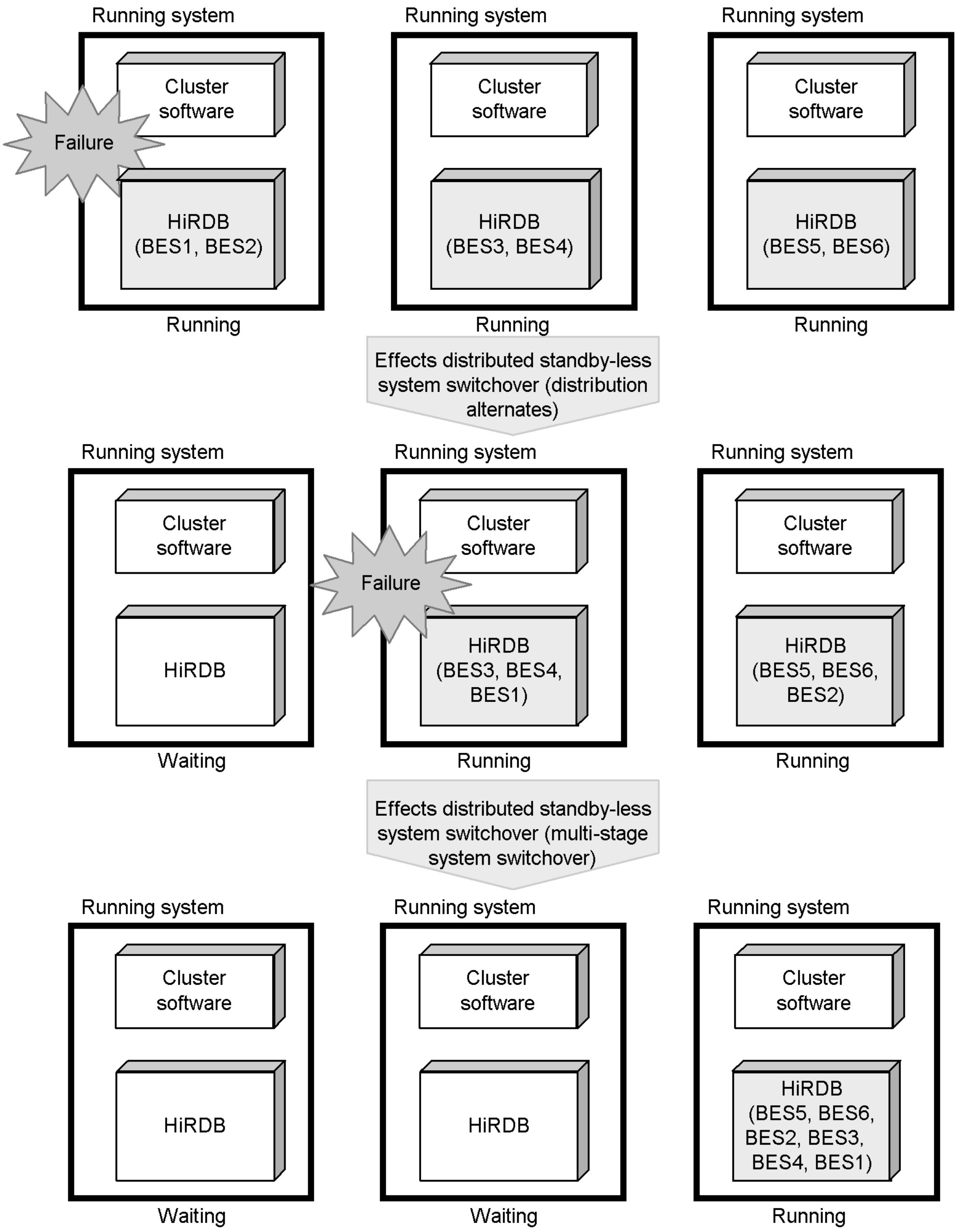
The standby-less system switchover (effects distributed) facility distributes the workload to and switches over among multiple back-end servers. The workload can also be distributed among multiple units. If another failure occurs, this time on a unit that was a switchover destination from the previous error, processing can be continued by again switching to a running unit (this is called *multi-stage system switchover*). Multi-stage system switchover cannot be performed with the standby-less system switchover (1:1) facility, so if a failure occurs at a switchover destination under that facility, processing for the failed unit cannot be continued.

It is appropriate to use the standby-less system switchover (effects distributed) facility in a system in which system resources must always be used at high efficiency and for which degradation of system performance must be minimized.

With the standby-less system switchover (effects distributed) facility, a back-end server that relinquishes processing when a failure occurs is called a *host BES*, and a back-end server that takes over processing is called a *guest BES*. The unit containing the host BESs is called the *regular unit*, and a unit containing a guest BES is called an *accepting unit*. All accepting units must be pre-defined as an *HA group*. The resources for back-end servers associated with guest BESs are called *guest areas*.

The following figure provides an overview of the standby-less system switchover (effects distributed) facility (with distribution alternates and multi-stage system switchover).

Figure 8–3: Overview of standby-less system switchover (effects distributed) facility (with distribution alternates and multi-stage system switchover)



**Prerequisites**

To use the standby-less system switchover (effects distributed) facility, the following conditions must be satisfied:

- HiRDB Advanced High Availability is installed
- The standby-less system switchover (effects distributed) facility can switch only to a unit dedicated to back-end servers (a unit that consists only of back-end servers).
- A unit that uses the standby-less system switchover (effects distributed) facility must consist of one or more back-end servers for the primary system. It cannot be used as a dedicated accepting unit.

## (3) Cluster software supported by HiRDB

The following table lists the cluster software that is supported by HiRDB.

Table 8–2: Cluster software supported by HiRDB

| Cluster software supported by HiRDB | OS | | | | |
|---|---|---|---|---|---|
| | HP-UX | Solaris | AIX | Linux | Windows |
| HA Monitor | Y | N | Y | Y | N |
| MC/ServiceGuard | Y | N | N | N | N |
| VERITAS Cluster Server | N | Y | N | N | N |
| Sun Cluster | N | Y | N | N | N |
| HACMP | N | N | Y | N | N |
| PowerHA | N | N | Y | N | N |
| ClusterPerfect | N | N | N | Y | N |
| LifeKeeper | N | N | N | Y | N |
| Microsoft Cluster Server (MSCS) or Microsoft Failover Cluster (MSFC) | N | N | N | N | Y |

Legend:

Y: Supported.

N: Not supported.

**Note**

- For details about cluster software, see the corresponding product documentation.
- Supported facilities differ depending on the type of cluster software you use. For details about the facilities supported for each type of cluster software, see *Table 8-3 Failures monitored in monitor mode and in server mode* and *Table 8-4 Cluster software that can be operated in monitor mode and in server mode*.

### (4) Shared disk unit

System switchover requires that there be an external hard disk (a character special file in the case of the UNIX edition) that is shared by the primary and secondary systems. This hard disk is called the *shared disk unit*; it is used to transfer information from the running system to the standby system when system switchover occurs. The following HiRDB files must be created on the shared disk unit:

- HiRDB files that comprise RDAREAs
- System files (system log files, synchronization point dump files, status files)
- Back-up files
- Unload log files
- HiRDB file system area for audit trail files (if the security audit facility is being used)

## 8.1.2 Operating the system switchover facility

You can operate the system switchover facility in either *monitor mode* or *server mode*. In monitor mode, only system failures are monitored. In server mode, both system failures and server failures are monitored. In addition, system switchover might take less time in server mode than in monitor mode. The following table shows the failures that are monitored in monitor mode and in server mode.

Table 8–3: Failures monitored in monitor mode and in server mode

| System switchover facility operation method | Monitored failure | |
|---|---|---|
| | System failure#1 | Server Failure#2 |
| Monitor mode | Y | N |

| System switchover facility operation method | Monitored failure | |
|---|---|---|
| | System failure#1 | Server Failure#2 |
| Server mode | Y | Y |

Legend:

Y: Monitored.

N: Not monitored.

#1

In this table, the following failures are assumed to be system failures; however, system failure conditions differ depending on the cluster software used. We recommend that you check your cluster software documentation for verification.

- Hardware failure
- OS failure
- Power outage
- Cluster software failure
- System slowdown

#2

In this table, the following failures are assumed to be server failures; however, server failure conditions differ depending on the cluster software used. We recommend that you check your cluster software documentation for verification.

- Abnormal termination of HiRDB (or unit if a HiRDB parallel server configuration)
- Slowdown of HiRDB (or unit if a HiRDB parallel server configuration)
- Database path errors

## (1) Cluster software that can be operated in monitor mode and in server mode

In this manual, the product that executes system switchover is referred to as *cluster software*. The following table lists the cluster software supported by HiRDB, and whether each product can be operated in monitor mode and server mode.

Table 8–4: Cluster software that can be operated in monitor mode and in server mode

| Cluster software | Monitor mode | Server mode |
|---|---|---|
| HA Monitor | Y | Y |
| MC/ServiceGuard | Y | Y |
| VERITAS Cluster Server | Y | Y |
| Sun Cluster | Y | N |
| HACMP | Y | N |
| PowerHA | Y | N |
| ClusterPerfect | Y | N |
| LifeKeeper | Y | N |
| Microsoft Cluster Server (MSCS)<br>Microsoft Failover Cluster (MSFC) | Y | Y |

Legend:

Y: Can be operated in this mode.

N: Cannot be operated in this mode.

### (2) Products needed to operate in server mode

The following table lists the products you need to operate the system switchover facility in server mode.

Table 8–5: Products needed to operate in server mode

| Function | HiRDB Advanced High Availability | Hitachi HA Toolkit Extension |
|---|---|---|
| Normal system switchover | -- | Y# |
| User server hot standby | -- | Y# |
| Rapid system switchover facility | -- | Y# |
| Standby-less system switchover (1:1) facility | Y | Y# |
| Standby-less system switchover (effects distributed) facility | Y | Y# |

Legend:

Y: The indicated product is needed to use the facility.

--: The product is not required.

#: You do not need Hitachi HA Toolkit Extension if HA Monitor is used as the cluster software (UNIX edition only).

## 8.1.3 System switchover facility configurations

The system switchover facility provides the following three switchover modes:

- Automatic system switchover
  The running system is switched over automatically when a failure occurs on it.
- Planned system switchover
  UNIX edition:
  A cluster software command is executed on the running system to switch over the system intentionally.
  Windows edition:
  The cluster administrator moves a HiRDB group to switch over the system intentionally.
- Grouped system switchover
  This is the system switchover mode used when HiRDB is linked to another product, such as an OLTP product. When a failure occurs on the running system, the OLTP product and HiRDB (unit) are grouped, and the grouped system is switched over. With grouped system switchover, you can use either automatic system switchover or planned system switchover. To determine whether you can use grouped system switchover, check your cluster software documentation.

## 8.1.4 System configuration examples

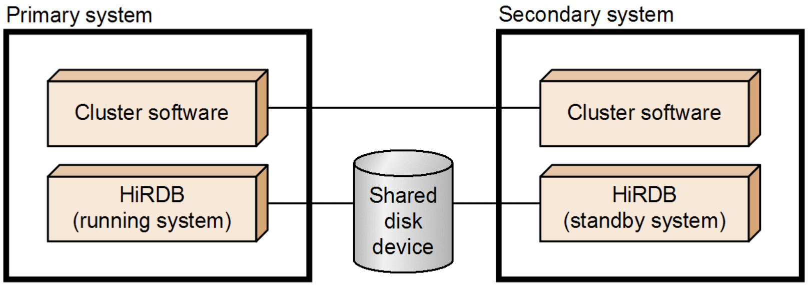
This subsection provides system configuration examples when the system switchover facility is being used.

### (1) System configuration examples of the standby system switchover facility

#### (a) One-to-one system switchover configuration

This configuration provides a one-to-one correspondence between running systems and standby systems. Use this configuration when you wish to guarantee response times, even when a system has been switched over. However, you cannot use the resources on the standby server machines (one set of server machine resources cannot be used for every two server machines). The following figure illustrates a one-to-one system switchover configuration.

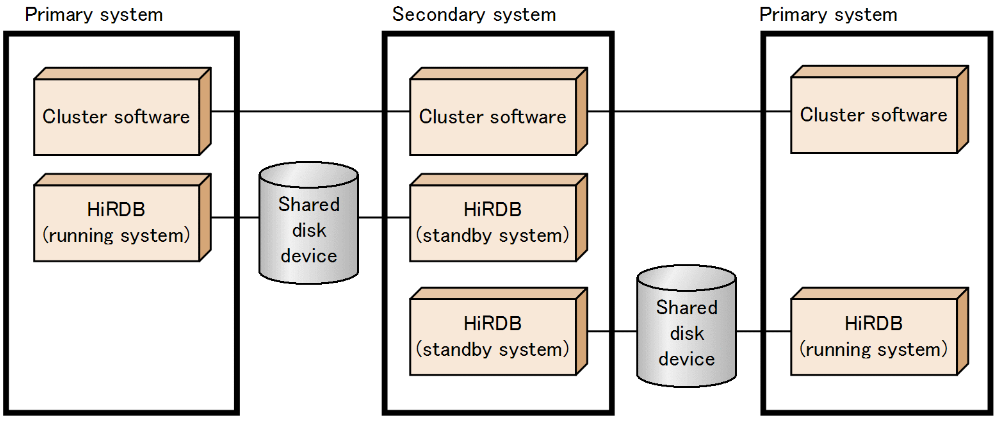
Figure 8–4: One-to-one system switchover configuration



### (b) Two-to-one system switchover configuration

This configuration provides a two-to-one correspondence between running systems and standby systems. The secondary system is configured as a multi-HiRDB system. Use this configuration with operations for which you wish to guarantee response times, even when a system has been switched over (response times deteriorate, though, if the two running systems have both been switched over). However, you cannot use the resources on the standby server machines (one set of server machine resources cannot be used for every three server machines). The following figure illustrates a two-to-one system switchover configuration.

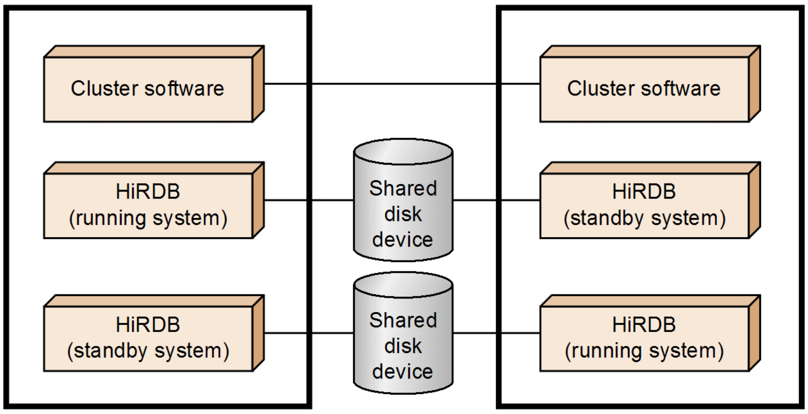
Figure 8–5: Two-to-one system switchover configuration



### (c) Mutual system switchover configuration

This configuration provides an alternating standby system on each server machine while, at the same time, the server machine is operating as a running system. Every server machine is configured as a multi-HiRDB system consisting of a HiRDB running system and a HiRDB standby system. Use this configuration when you wish to utilize server machine resources most efficiently. However, response times deteriorate when a system has been switched over. The following figure illustrates a mutual system switchover configuration.

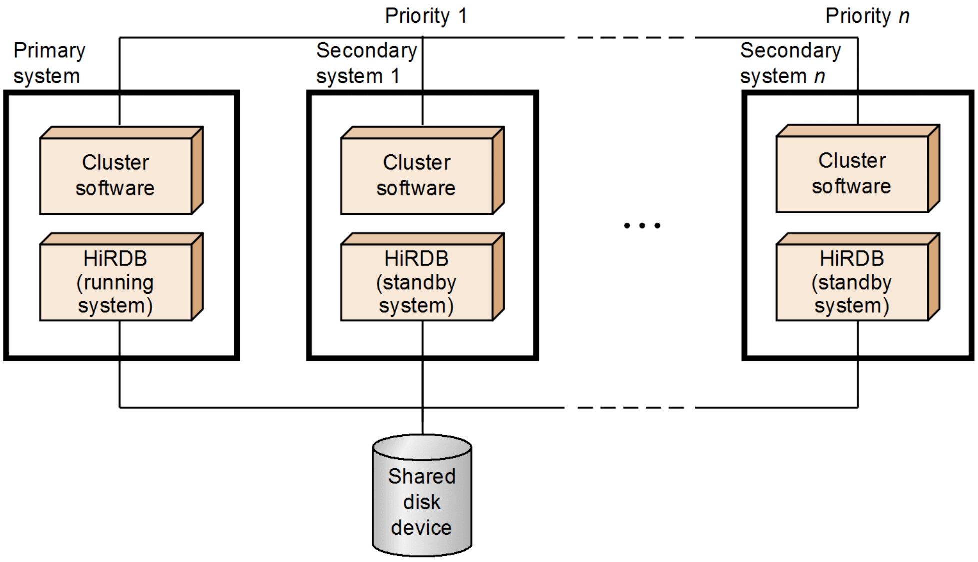
Figure 8–6: Mutual system switchover configuration



### (d) Multi-standby configuration

In this configuration, a single running system has multiple standby systems. You can apply this configuration to guard against a failure in a standby system before the running system recovers from a failure (multi-point failures). Standby systems are assigned different levels of priority, and when a failure occurs in the running system, the system is switched to the standby system having the highest priority. The following figure shows the multi-standby configuration.

Figure 8–7: Multi-standby configuration



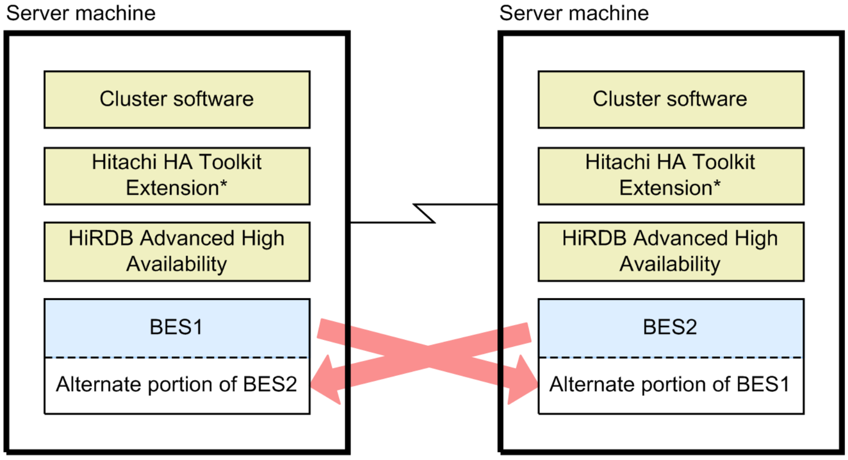
## (2) System configuration examples of standby-less system switchover (1:1) facility

This subsection provides examples of typical system configurations when the standby-less system switchover (1:1) facility is used.

### (a) Mutual alternating configuration

This configuration example employs reciprocal alternate BESs on two back-end servers using standby-less system switchover (1:1). The following figure illustrates a system configuration example of such a mutual alternating configuration.

Figure 8–8: System configuration example of a mutual alternating configuration


Server machine
Server machine
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
BES1
Alternate portion of BES2
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
BES2
Alternate portion of BES1

* You do not need Hitachi HA Toolkit Extension if HA Monitor is used as the cluster software (UNIX edition only).

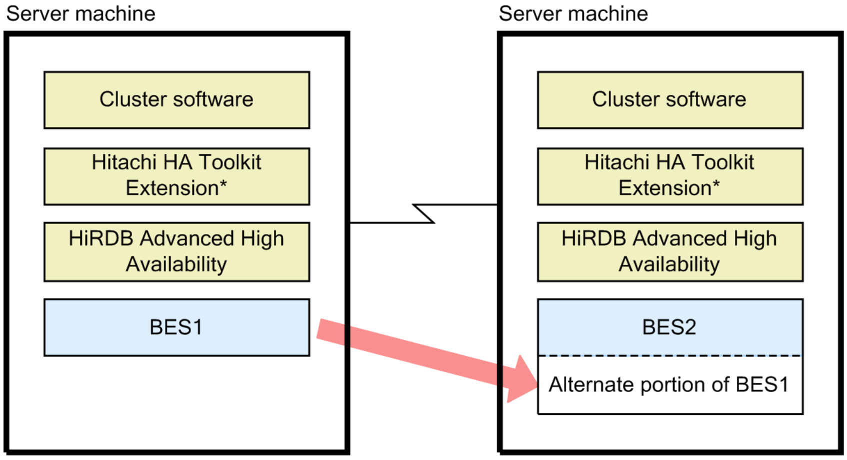
**Explanation**

- BES1 is the alternate BES for BES2. If a failure occurs on BES2, the alternate portion for BES2 takes over the BES2 processing.
- BES2 is the alternate BES for BES1. If a failure occurs on BES1, the alternate portion for BES1 takes over the BES1 processing.

### (b) One-way alternating configuration

This configuration example employs an alternate BES on one back-end server only using standby-less system switchover (1:1). The following figure illustrates a system configuration example of a one-way alternating configuration (2-node configuration).

Figure 8–9: System configuration example of a one-way alternating configuration (2-node configuration)


Server machine
Server machine
Cluster software
Cluster software
Hitachi HA Toolkit Extension*
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
HiRDB Advanced High Availability
BES1
BES2
Alternate portion of BES1

* You do not need Hitachi HA Toolkit Extension if HA Monitor is used as the cluster software (UNIX edition only).

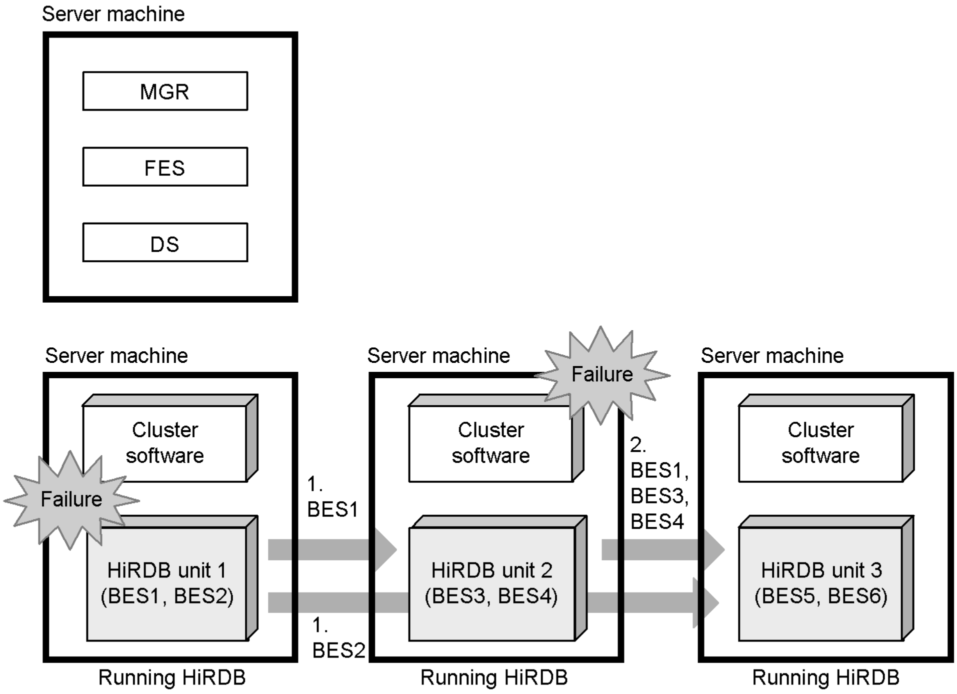
**Explanation**

BES2 is the alternate BES for BES1. If a failure occurs on BES1, the alternate portion for BES1 takes over the BES1 processing. If a failure occurs on BES2, BES1 does not take over its processing.

## (3) System configuration examples of the standby-less system switchover (effects distributed) facility

The following figure illustrates a system configuration example of the standby-less system switchover (effects distributed) facility. When a failure occurs in a regular unit, processing directed to the back-end servers of the failed primary system is distributed to and executed on multiple active server machines at their back-end servers.

Figure 8–10: System configuration example of the standby-less system switchover (effects distributed) facility



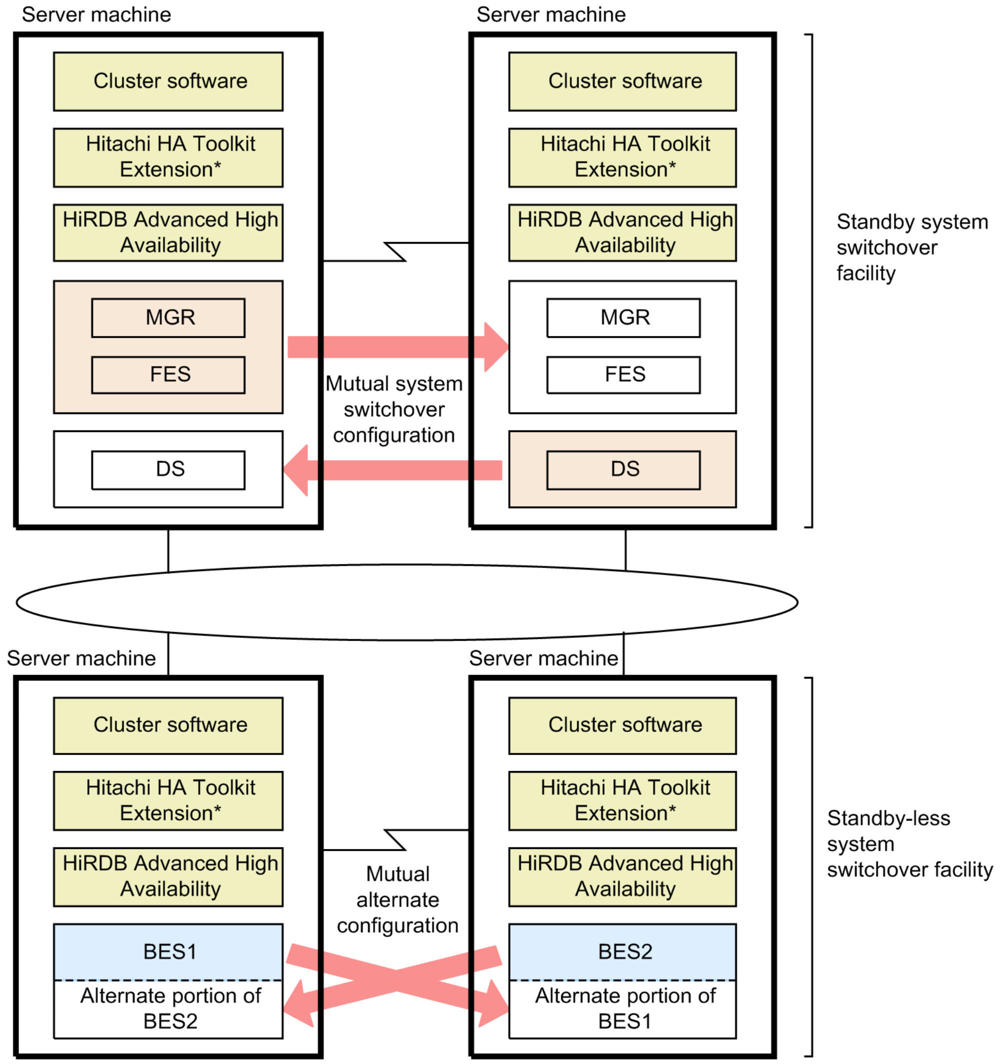
**Explanation**

1. If a failure occurs in unit 1, unit 2 executes the processing as a guest BES for BES1 and unit 3 executes the processing as a guest BES for BES2.
2. If a failure occurs in unit 2 while unit 1 is still down, unit 3 executes the processing as guest BESs for BES1, BES2, BES3, and BES4.

## (4) System configuration example of a mixed standby-less (1:1) and standby type setup

The following figure shows a configuration example of a mixed standby-less (1:1) and standby type setup.

Figure 8–11: System configuration example of a mixed standby-less (1:1) and standby type setup


Server machine
Server machine
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
MGR
FES
DS
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
MGR
FES
DS
Mutual system switchover configuration
Standby system switchover facility
Server machine
Server machine
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
BES1
Alternate portion of BES2
Cluster software
Hitachi HA Toolkit Extension*
HiRDB Advanced High Availability
BES2
Alternate portion of BES1
Mutual alternate configuration
Standby-less system switchover facility

* You do not need Hitachi HA Toolkit Extension if HA Monitor is used as the cluster software (UNIX edition only).

**Explanation**

- The units containing a MGR (system manager), FES (front-end server), and DS (dictionary server) employ the standby system switchover facility in a mutual system switchover configuration.
- The units containing a BES (back-end server) employ the standby-less system switchover (1:1) facility in a mutual alternating configuration.
- HiRDB Advanced High Availability is needed on all server machines. HiRDB Advanced High Availability is also required on server machines for which neither the standby-less system switchover facility nor the standby system switchover facility is employed.

## 8.1.5 Functions that reduce system switchover time (user server hot standby and the rapid system switchover facility)

HiRDB offers the following functions to reduce system switchover time:

- User server hot standby
- Rapid system switchover facility

Note that, to use these functions, the system switchover facility must be operating in the server mode. You cannot use these functions if the system switchover facility is operating in the monitor mode.

## (1) User server hot standby

When a system switchover occurs, the following processing is performed to start the standby HiRDB:

- System server startup processing
- System file transfer processing
- Server process startup processing
- Roll forward processing

Of the preceding, the time needed for the server process startup processing accounts for a large proportion of the entire system switchover time. The time required for server process startup processing is directly proportional to the number of resident server processes, which means that system switchover time increases as the number of resident processes increases. You can reduce this time, however, by starting the server processes of the standby HiRDB beforehand, so that server process startup processing is not performed when the system is switched over. This allows you to reduce the system switchover time by the amount of time required for system process startup processing. This concept is called *user server hot standby*. For example, on a server machine that runs at a speed of approximately 100 MIPS, one second is required to start a single server process. In this case, user server hot standby would reduce the system switchover time by about one second for each resident server process.

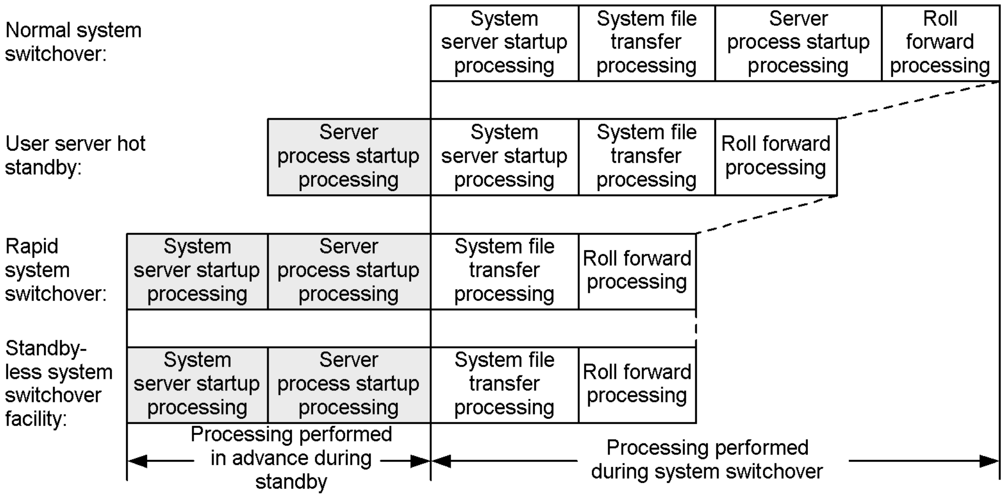
For details about user server hot standby, see the *HiRDB Version 9 System Operation Guide*.

## (2) Rapid system switchover facility

A facility is provided to start the server processes on the standby HiRDB beforehand, so that startup processing of the server processes need not be performed when the system is switched over. This facility is called the *rapid system switchover facility*. With it, you can reduce the system switchover time by the amount of time required to perform startup processing of server processes during system switchover. For details about the rapid system switchover facility, see the *HiRDB Version 9 System Operation Guide*.

Note that the rapid system switchover facility can reduce the system switchover time more than user server hot standby can (the rapid system switchover facility includes the user server hot standby function). The following figure compares the system switchover times of various operating modes.

Figure 8–12: Comparison of system switchover times


Normal system switchover:
System server startup processing
System file transfer processing
Server process startup processing
Roll forward processing
User server hot standby:
Server process startup processing
System server startup processing
System file transfer processing
Roll forward processing
Rapid system switchover:
System server startup processing
Server process startup processing
System file transfer processing
Roll forward processing
Standby-less system switchover facility:
System server startup processing
Server process startup processing
System file transfer processing
Roll forward processing
Processing performed in advance during standby
Processing performed during system switchover

**Explanation**

The processing indicated in the shaded area is performed in advance, after which the associated processes stand by; therefore, this processing does not need to be performed when a system is switched over. Thus, system switchover time is reduced by the amount of time required to perform the processing indicated in the shaded area.

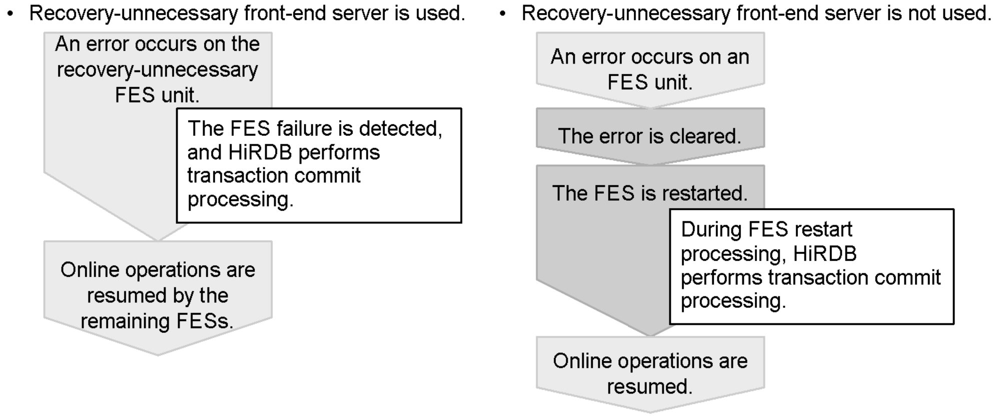
# 8.2 Recovery-unnecessary front-end servers

Installing HiRDB Non Recover FES enables you to use recovery-unnecessary front-end servers. This section provides a brief description of recovery-unnecessary front-end servers. For details about recovery-unnecessary front-end servers, see the *HiRDB Version 9 Installation and Design Guide*.

## 8.2.1 Overview of recovery-unnecessary front-end servers

When an error in a unit that contains a front-end server causes the front-end server to terminate abnormally, transactions that were being executed by that front-end server might remain uncommitted. Transactions in uncommitted status maintain an exclusive hold on the database, restricting access to and updating of that locked portion of the database. Normally, the front-end server error must be cleared and the server restarted to commit the uncommitted transactions. However, if a front-end server that is terminated abnormally is a recovery-unnecessary front-end server, HiRDB automatically commits its uncommitted transactions. You can then you use another front-end server or back-end server to resume processing on the database. A unit that contains a recovery-unnecessary front-end server is called a *recovery-unnecessary front-end server unit*. The following figure shows the operation with and without using a recovery-unnecessary front-end server.

Figure 8–13: Operation with and without using a recovery-unnecessary front-end server


• Recovery-unnecessary front-end server is used.
An error occurs on the recovery-unnecessary FES unit.
The FES failure is detected, and HiRDB performs transaction commit processing.
Online operations are resumed by the remaining FESs.
• Recovery-unnecessary front-end server is not used.
An error occurs on an FES unit.
The error is cleared.
The FES is restarted.
During FES restart processing, HiRDB performs transaction commit processing.
Online operations are resumed.

Note that HiRDB Non Recover FES must be installed in order to use recovery-unnecessary front-end servers.

**Application criteria**

This feature enables the front-end servers that remain after an error to resume online operations without the erroneous front-end server having to be restarted. It is recommended that this feature be used with systems that require continuous, 24-hour per day operation.

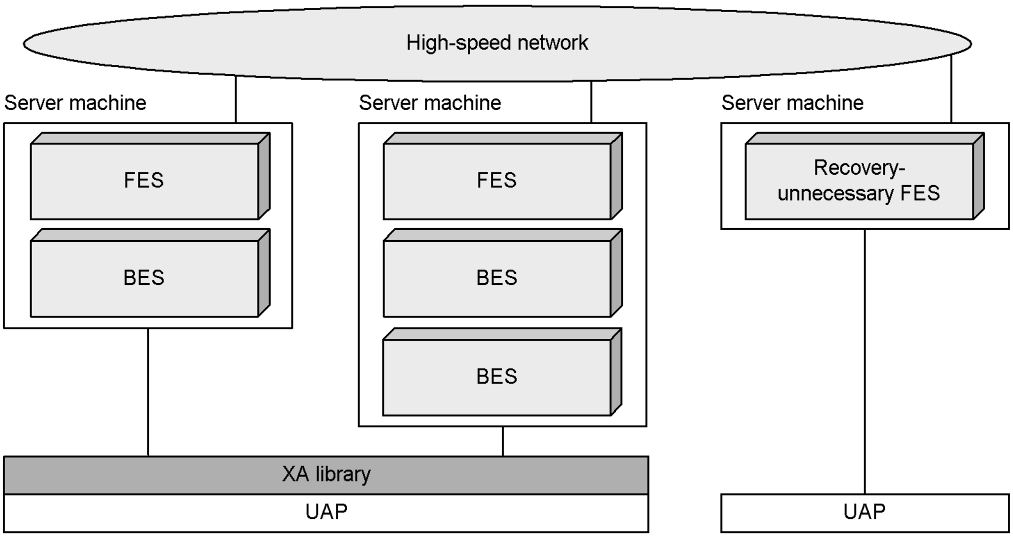
**Specification method**

To use a recovery-unnecessary front-end server, specify `stls` in the `-k` option of the `pdstart` operand.

## 8.2.2 Configuration example of a system that uses a recovery-unnecessary front-end server

The following figure shows a configuration example of a system that uses a recovery-unnecessary front-end server.

Figure 8–14: Configuration example of a system that uses a recovery-unnecessary front-end server



**Explanation**

- Set up the recovery-unnecessary front-end server in a stand-alone unit.
- UAPs that connect using the X/Open XA interface cannot be executed on a recovery-unnecessary front-end server. Specify `PDFESHOST` and `PDSERVICEGRP` in the client environment definition to direct them to connect to front-end servers other than a recovery-unnecessary front-end server.
- You can still execute the `pdrplstart` and `pdrplstop` commands, even if a recovery-unnecessary front-end server or a recovery-unnecessary front-end server unit is stopped.

# 8.3 Real Time SAN Replication (disaster recovery) (UNIX edition only)

This section provides a brief description of Real Time SAN Replication, which is a disaster recovery system designed to support quick recovery from a large-scale disaster, such as an earthquake or fire. For details about how to operate Real Time SAN Replication, see the *HiRDB Version 9 Disaster Recovery System Configuration and Operation Guide*.

! Important note

Real Time SAN Replication is supported only on the HP-UX, AIX (other than AIX V7.1), and Linux (Linux 5 (AMD/Intel 64)) editions of HiRDB.

Reference note

The discussion in this section assumes that the reader is familiar with RAID Manager, TrueCopy, and Universal Replicator.

## 8.3.1 Overview of Real Time SAN Replication

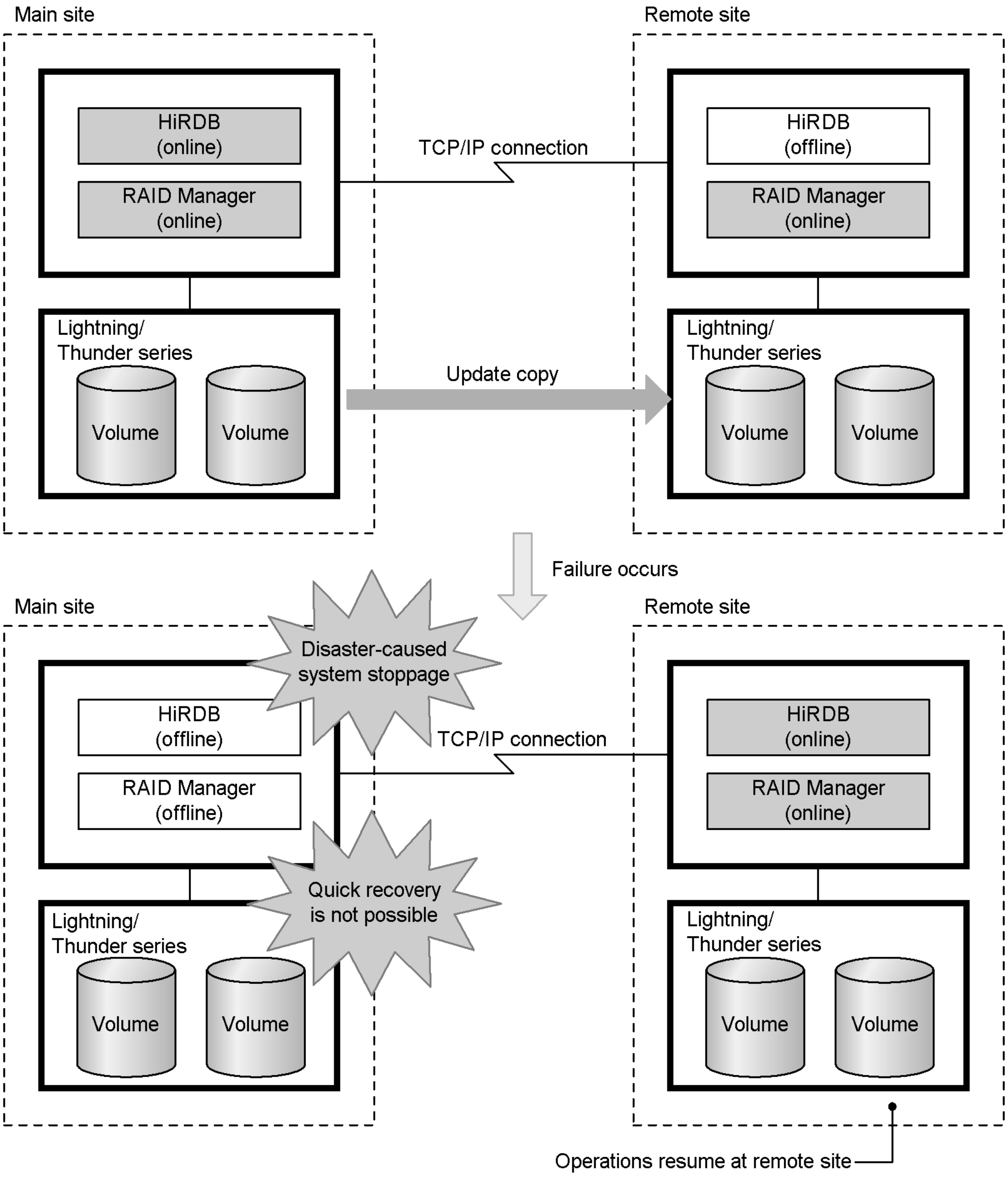
### (1) Overview

A system that is being used on a regular basis might become physically difficult or impossible to recover in the aftermath of a disaster such as an earthquake or fire. A system configuration can be created that provides for operations to be resumed under such circumstances by starting a standby system that has been pre-positioned at a remote site. This is known as *Real Time SAN Replication* (RiSe). The site at which is located the system that is being used on a regular basis is called the *main site*, and the site at which the standby system is pre-positioned is called the *remote site*.

Data at both the main site and the remote site is stored on Hitachi disk array systems. When data at the main site is updated, the Hitachi disk array system's TrueCopy or Universal Replicator is used to mirror (update copy) the data to the remote site.

The following figure provides an overview of Real Time SAN Replication.

Figure 8–15: Overview of Real Time SAN Replication



**Explanation**

- The HiRDB at the main site is used normally to perform operations. When a file at the main site is updated, the updated data is update-copied to the remote site. This update copying maintains the same data at both the main site and the remote site.
- If a disaster such as an earthquake or fire strikes the main site and the system at the main site cannot be recovered quickly, operations can be resumed by restarting HiRDB at the remote site.

Reference note

- Update copy is performed by TrueCopy or Universal Replicator, which copy data directly between Hitachi disk array systems without going through the hosts.
- RAID Manager is an optional program for a Hitachi disk array system, and provides commands for controlling and operating TrueCopy and Universal Replicator.

### (2) Files subject to update copying

The files that are subject to update copying are listed below. When any of these files is updated, the updated information is copied to the same file at the remote site.

- Database files (HiRDB files comprising RDAREAs)
- System log files
- Synchronization point dump files
- Status files

### (3) Synchronous copy and asynchronous copy

Update copying can be performed by means of synchronous copy or asynchronous copy. The following table compares the features of synchronous copy and asynchronous copy.

Table 8–6: Features of synchronous copy and asynchronous copy

| Item | Synchronous copy | Asynchronous copy |
|---|---|---|
| Processing method | Update processing is not complete at the main site until the matching update processing is completed at the remote site; update processing at the main site waits until update processing at the remote site is finished. | Update processing at the main site is completed without waiting for update processing to be performed at the remote site. |
| Data compatibility between main site and remote site | Data is always synchronized between the main site and the remote site. | Loss of data synchronization might occur. Therefore, data between the main site and the remote site might not always match. |
| Effect on performance | Transaction processing performance declines. The percentage of decline is proportional to the distance between the sites.#1 | No effect on performance occurs.#1, #2 |

#1: This feature is consistent with logical values guaranteed by TrueCopy.

#2: This feature is consistent with logical values guaranteed by Universal Replicator.

## 8.3.2 Prerequisite platforms and products

### (1) Prerequisite platforms

The prerequisite platforms are listed below. It is recommended that you use the same platform at both the main site and the remote site.

- AIX (other than AIX V7.1)
- HP-UX
- Linux (Linux 5 (AMD/Intel 64))

### (2) Prerequisite products

Lightning/Thunder series products are required at both the main and remote sites. For details about prerequisite products, see the *HiRDB Version 9 Disaster Recovery System Configuration and Operation Guide*.

## 8.3.3 Methods of mirroring data to the remote site

The following subsections describe the four processing methods provided by Real Time SAN Replication to mirror data from the main site to the remote site. The system configuration and the operating procedures differ depending on the method used to mirror data, so the HiRDB administrator must choose the method that best suits the local system environment. The following are the four methods:

- All synchronous method
- All asynchronous method
- Hybrid method
- Log-only synchronous method

## 8.3.4 All synchronous method

When you use the all synchronous method, update copying to the remote site is performed using synchronous copying. With synchronous copy, update processing is not complete at the main site until update processing at the remote site is completed (update processing at the main site waits for update processing at the remote site to be finished). Therefore, when the all synchronous copy method is used, updates at the main site are always mirrored at the remote site. This means that if the HiRDB at the main site fails due to a disaster or other calamity, HiRDB can be restarted and service can be resumed at the remote site in the same status as existed immediately before the failure.

However, because HiRDB at the main site pauses each time a file at the remote site is being updated, transaction processing at the main site is affected adversely.

The following figure provides an overview of the all synchronous method, and the table below it shows the processing method used with the all synchronous method when update copying to the remote site is performed.

Figure 8–16: Overview of the all synchronous method

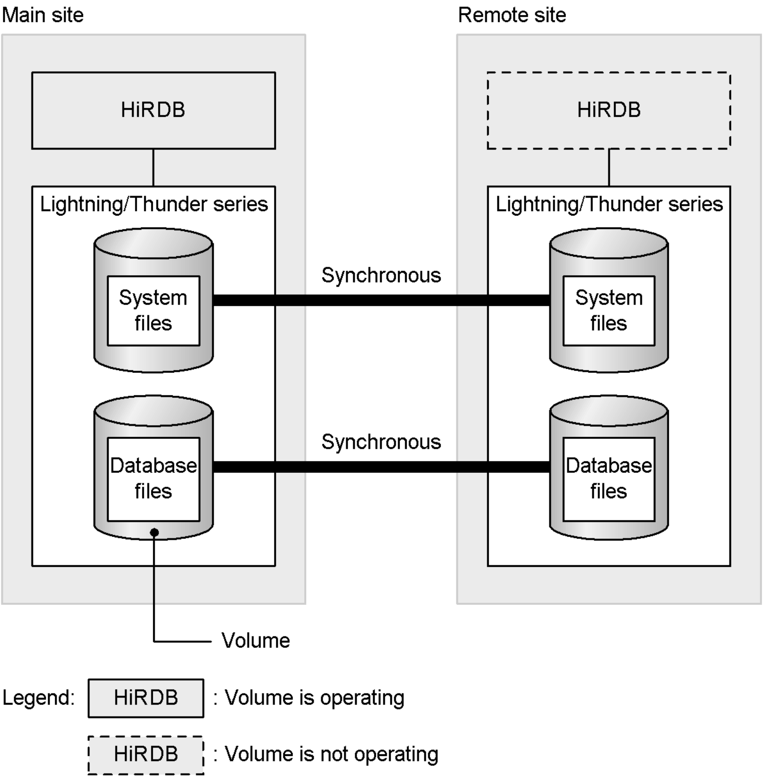


Table 8–7: Processing method used when update copying to the remote site is performed (all synchronous method)

| Files that are copied to the remote site | | Processing method used for update copying |
|---|---|---|
| Database files | | Synchronous copy |
| System files | System log files | |
| | Synchronization point dump files | |
| | Status files | |

## 8.3.5 All asynchronous method

When you use the all asynchronous method, update copying to the remote site is performed using asynchronous copying. With asynchronous copy, there are no adverse effects on transaction performance at the main site because update processing at the main site is completed without waiting for update processing to be completed at the remote site.

However, updates at the main site in a file subject to update copying might not always be reflected at the remote site. This means that if HiRDB at the remote site is restarted after the HiRDB at the main site has experienced a failure due to a disaster or other calamity, the status in which HiRDB is restarted might differ from the status of HiRDB immediately preceding the failure. Therefore, resumption of uninterrupted service from the main site cannot be guaranteed when you use the all asynchronous method.

The following figure provides an overview of the all asynchronous method, and the table below it shows the processing method used with the all asynchronous method when update copying to the remote site is performed.

Figure 8–17: Overview of the all asynchronous method

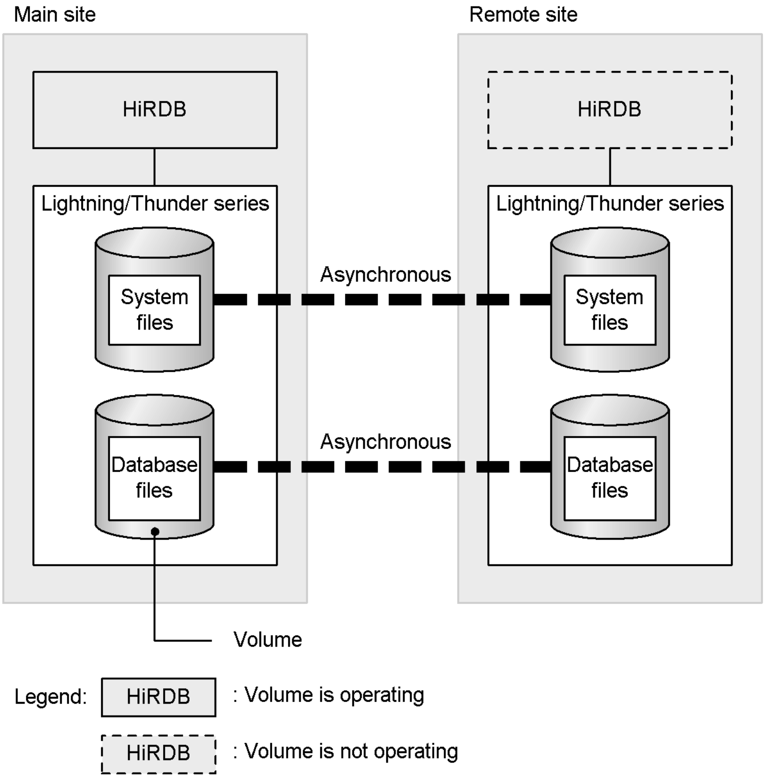


Table 8–8: Processing method used when update copying to the remote site is performed (all asynchronous method)

| Files that are copied to the remote site | | Processing method used for update copying |
|---|---|---|
| Database files | | Asynchronous copy |
| System files | System log files | |
| | Synchronization point dump files | |
| | Status files | |

## 8.3.6 Hybrid method

When you use the hybrid method, update copying to the remote site is performed as follows:

- Update copying of database files is performed using asynchronous copying
- Update copying of system files is performed using synchronous copying

System logs and other files containing information needed to recover the database are copied to the remote site with synchronous copying, so their synchronization is guaranteed. This means that if the HiRDB at the main site fails due to a disaster, the HiRDB at the remote site can be restarted in the same status that the HiRDB at the main site was in immediately preceding the failure. The hybrid method is designed for use mainly in large-scale systems.

Asynchronous copying is then used for recoverable database files, thus reducing the severity of the adverse effects on transaction performance compared with the all synchronous method.

Reference note

Although the hybrid method provides the advantages of both the all synchronous method and all asynchronous method, operation is more difficult than under the other two methods.

The following figure provides an overview of the hybrid method, and the table below it shows the processing methods used with the hybrid method when update copying to the remote site is performed.

Figure 8–18: Overview of the hybrid method

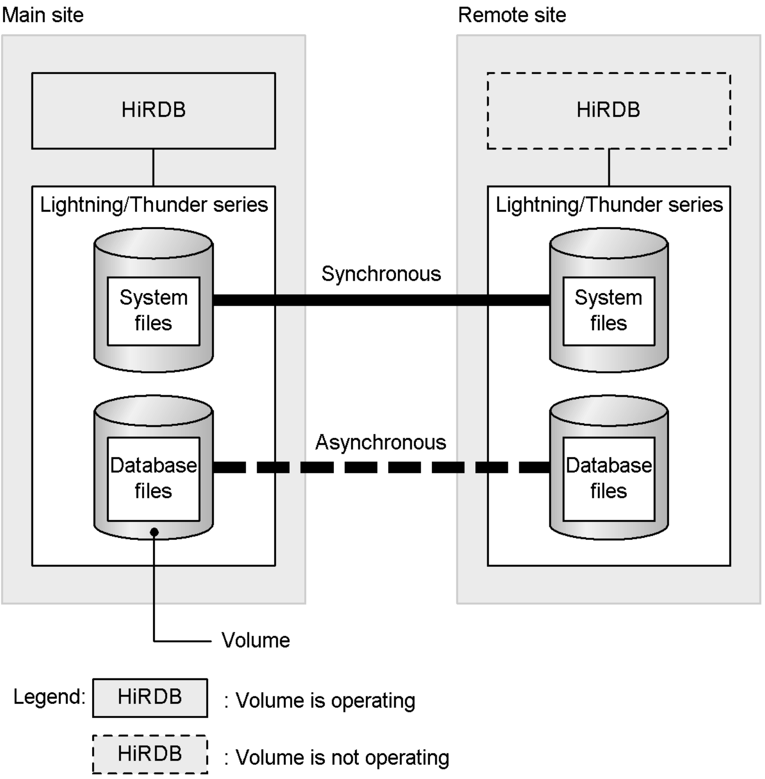


Table 8–9: Processing methods used when update copying to the remote site is performed (hybrid method)

| Files that are copied to the remote site | | Processing method used for update copying |
|---|---|---|
| Database files | | Asynchronous copy |
| System files | System log files | Synchronous copy |
| | Synchronization point dump files | |
| | Status files | |

## 8.3.7 Log-only synchronous method

When you use the log-only synchronous method, update copying to the remote site is performed as follows:

- Update copying of database files is performed once during initial setup or when preparation for log application is performed.
- Update copying of system files is performed synchronously.

The log-only synchronous method updates a database by accepting transactions at the *transaction execution site* and by then performing update processing of the database at the *log application site* based on system logs copied from the transaction execution site. Normally, the main site is the transaction execution site, and the remote site is the log application site.

System log files and other information needed to recover a database are copied synchronously to guarantee that they are mirrored at the remote site. This means that if the HiRDB system at the main site terminates abnormally, the HiRDB system at the remote site can be restarted in the same status that the HiRDB system at the main site was in immediately preceding its abnormal termination. The log-only synchronous method is designed for use mainly in small to medium-sized systems.

Because copying of database files is performed once during initial setup or when preparation for log application is performed, the amount of communication traffic is generally less than with the other methods. This reduces adverse effects on transaction performance. Because the database at the remote site is updated based on system logs, the HiRDB at the remote site must always be operational.

! Important note

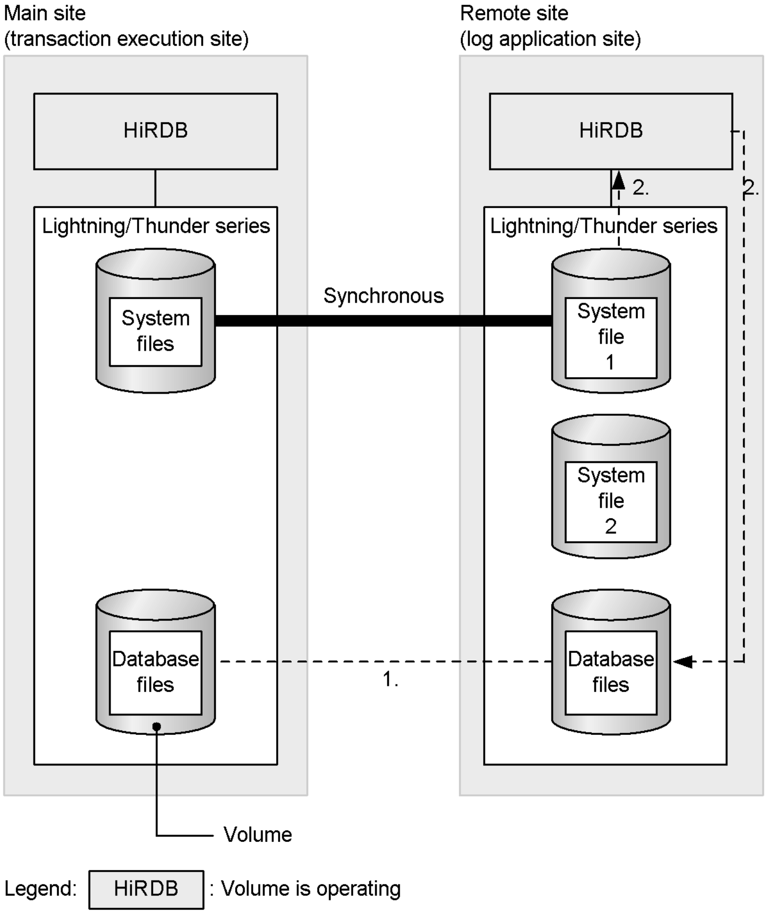
To use the log-only synchronous method, you must have installed the HiRDB Disaster Recovery Light Edition optional program product and have used the `pdopsetup` command to set it up.

Reference note

Although the log-only synchronous method generally requires less communication traffic than the other methods, this method is more difficult to use than the hybrid method.

The following figure provides an overview of the log-only synchronous method, and the table below it shows the processing methods used with the log-only synchronous method when update copying to the log application site is performed.

Figure 8–19: Overview of log-only synchronous method



**Explanation**

1. The database files are copied only during initial setup or when preparation for log application is performed; copying is not performed at other times.
2. Database update processing is performed by reading the logs that have been copied synchronously from the transaction execution site. This is called *log application*.

Table 8–10: Processing methods when update copying is performed at the log application site (for the log-only synchronous method)

| Files copied to the log application site | | Processing method used when update copying is performed |
|---|---|---|
| Database files | | Copied only during initial setup or when preparation for log application is performed. |
| System file 1 | System log files | Synchronous copy |
| | Synchronization point dump file for transaction processing | |
| | Status file for transaction processing | |
| System file 2 | Synchronization point dump file for log application processing | Not copied |
| | Status file for log application processing | |

Reference note

The synchronization point dump file for transaction processing and status file for transaction processing are used in log application processing (they are the same synchronization point dump file and status file normally used in

HiRDB operations). The synchronization point dump file for log application processing and status file for log application processing are used to obtain synchronization point dump and system status information while log application processing is being performed.

! Important note

Unlike the all synchronous, all asynchronous, and hybrid methods, the log-only synchronous method requires system file 2 (synchronization point dump file for log application processing and status file for log application processing). Therefore, take into account the need for these additional files when you configure the hardware.

# 9 Facilities Related to Security Measures

This chapter describes facilities that provide support for database security measures.

# 9.1 Security facility

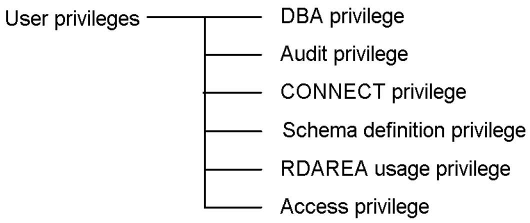
To prevent database access by outsiders, HiRDB provides a *security facility*. Based on a concept called *user privileges*, the security facility allows only users who have an appropriate privilege to access the database.

This section provides an overview of the security facility. For details about how to operate the security facility, see the *HiRDB Version 9 System Operation Guide*.

## 9.1.1 User privileges

This section explains the user privileges that are set up by HiRDB. The following figure shows the HiRDB user privileges.

Figure 9–1: HiRDB user privileges



These HiRDB user privileges are granted to various users, such as HiRDB administrators, DBA privilege holders, and schema owners.

**Privileges granted to HiRDB administrators**

The administrator's own DBA privilege, audit privilege, and RDAREA usage privilege

**Privileges granted to DBA privilege holders**

DBA privilege, schema definition privilege, RDAREA usage privilege, and CONNECT privilege

**Privilege granted to schema owners**

Access privilege

### (1) DBA privilege

The DBA privilege is required in order to grant and revoke DBA privileges, CONNECT privileges, and schema definition privileges. It permits the following actions:

- Granting DBA privileges, CONNECT privileges, and schema definition privileges to other people
- Revoking DBA privileges, CONNECT privileges, and schema definition privileges granted to other people
- Defining other users' schemas
  Defining a schema allows the schema owner to define base tables, view tables, indexes, abstract data types, stored procedures, stored functions, and triggers.
- Deleting other users' schemas, base tables, view tables, indexes, abstract data types, stored procedures, stored functions, and triggers
- Defining items related to the connection security facility
- Connecting to HiRDB (CONNECT privilege holders)

### (2) Audit privilege

This privilege is required for auditors. This privilege allows a user to perform the following actions:

- Accessing audit trail tables
- Loading data to audit trail tables
- Granting and revoking the `SELECT` privilege to audit trail tables

- Deleting audit trail tables
- Changing auditor passwords
- Defining and deleting audit events
- Swapping audit trail files

To use the security audit facility, you need to set the audit privilege. For details about the security audit facility, see *9.2 Security audit facility*.

### (3) CONNECT privilege

The CONNECT privilege is required in order to use HiRDB. This privilege permits a user to connect to the database. An attempt to connect to the database by a user who does not have the CONNECT privilege results in an error.

### (4) Schema definition privilege

The schema definition privilege is required in order to define a schema. This privilege permits a user to take the following actions:

- Defining the user's own schema

  Defining a schema allows the schema owner to define base tables, view tables, indexes, abstract data types, stored procedures, stored functions, and triggers.
- Deleting the user's own schemas, base tables, view tables, indexes, abstract data types, stored procedures, stored functions, and triggers

### (5) RDAREA usage privilege

The RDAREA usage privilege is required in order to use an RDAREA. This privilege permits a user to define tables and indexes in the RDAREA to which the privilege applies. An RDAREA for which the RDAREA usage privilege is granted by specifying an authorization identifier is called a *private user RDAREA*, and an RDAREA for which the RDAREA usage privilege is granted by specifying `PUBLIC` is called a *public user RDAREA*.

### (6) Access privileges

An access privilege is required in order to access a table. Only those users who have an access privilege are allowed to access a table. Access privileges are set for each table. The types of access privilege are listed in the following table.

Table 9–1: Access privilege types

| Access privilege type | Explanation |
|---|---|
| `select` privilege | Allows retrieval (`select`) of row data from the table. |
| `INSERT` privilege | Allows addition (`INSERT`) of row data to the table. |
| `DELETE` privilege | Allows deletion (`DELETE`) of row data from the table. |
| `UPDATE` privilege | Allows updating (`UPDATE`) of row data in the table. |

## 9.1.2 Operating the security facility

HiRDB administrators, DBA privilege holders, and schema administrators operate the security facility by assigning various privileges to HiRDB users. For details about assigning privileges, see the *HiRDB Version 9 System Operation Guide*.

# 9.2 Security audit facility

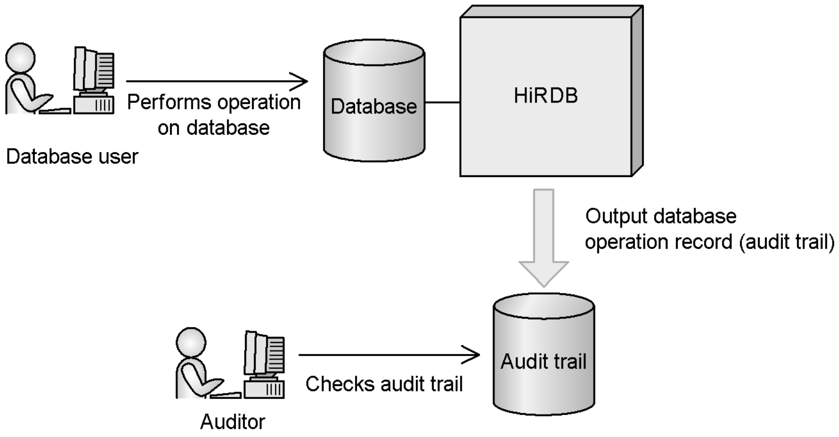
This section describes the security audit facility. For details about how to operate the security audit facility, see the *HiRDB Version 9 System Operation Guide*.

## 9.2.1 Overview of the security audit facility

### (1) Functional overview

HiRDB security is protected by means of privileges. The information that can be accessed or updated, and the objects that can be manipulated (tables, indexes, and so on), are controlled with privileges. To check whether these privileges are being applied properly, HiRDB can record a variety of actions that are performed on the databases. This functionality is called the *security audit facility*, and the operations record that it outputs is called an *audit trail*. By examining the output audit trail, you can check whether there has been fraudulent access. This check is performed by users, called *auditors*, who have been assigned the audit privilege. The following figure provides an overview of the security audit facility.

Figure 9–2: Overview of the security audit facility


Performs operation on database
Database user
Database
HiRDB
Output database operation record (audit trail)
Checks audit trail
Audit trail
Auditor

The security audit facility collects information about who is using privileges, which privileges they are using, and the objects on which they are using the privileges to perform operations. The auditor uses the `CREATE AUDIT` statement to specify the operations on which the security audit facility is to collect information. Once specified, an audit trail is collected whenever an operation for which an audit trail is specified to be collected is performed.

Reference note

- The purpose of the security audit facility is not to enhance security. It is designed simply to output an operation log that enables checking of whether privileges are being used correctly.
- You can link HiRDB to JP1/NETM/Audit and have JP1/NETM/Audit collect and centrally manage HiRDB's audit trails. For details, see *2.4.8 JP1/NETM/Audit*.

### (2) Audit trail collection times

HiRDB collects an audit trail when any of the following events occurs:

- When a privilege check is performed because a command or SQL statement is executed
- When an event ends

The security audit facility does not collect an audit trail when an SQL syntax error occurs or when an incorrectly keyed command is entered.

For details about audit trail collection triggers, see the *HiRDB Version 9 System Operation Guide*.

### (3) Audit trail collection examples

Examples of audit trail collection are provided in this subsection.

**Example 1: Collecting an audit trail when a table is searched**

The table access privilege (SELECT privilege) is used when a table is searched, so an audit trail is collected.

| Item searched (SQL specification) | | Contents of audit trail | | | | |
|---|---|---|---|---|---|---|
| | | User | Privilege used | Type of manipulated object | Name of manipulated object | Operation type |
| A user (USR1) issues the following SELECT statement:<br>SELECT C1 FROM USR1.T1 | Privilege | USR1 | Table access privilege (SELECT privilege) | Table | USR1.T1 | Table access (SELECT) |
| | End | USR1 | -- | Table | USR1.T1 | Table access (SELECT) |
| A user (USR2) issues the following SELECT statement:<br>SELECT T1.C1,T2.C1<br>FROM USR1.T1 T1,USR2.T2 T2<br>WHERE T1.C1=T2.C1 | Privilege | USR2 | Table access privilege (SELECT privilege) | Table | USR1.T1 | Table access (SELECT) |
| | | USR2 | Table access privilege (SELECT privilege) | Table | USR2.T2 | Table access (SELECT) |
| | End | USR2 | -- | Table | USR1.T1 | Table access (SELECT) |
| | | USR2 | -- | Table | USR2.T2 | Table access (SELECT) |

Legend:

Privilege: Audit trail is collected at time of privilege checking.

End: Audit trail is collected when the event terminates.

--: Not applicable

**Example 2: Collecting an audit trail when a table is defined or deleted**

The schema owner privilege, table owner privilege, and RDAREA owner privilege are used when a table is defined or deleted, so an audit trail is collected.

| Item searched (SQL specification) | | Contents of audit trail | | | | |
|---|---|---|---|---|---|---|
| | | User | Privilege used | Type of manipulated object | Name of manipulated object | Operation type |
| A user (USR1) issues the following CREATE TABLE:<br>CREATE TABLE<br>T1(C1 INT) IN RDAREA1 | Privilege | USR1 | RDAREA usage privilege | RDAREA | RDAREA1 | Definition creation |
| | | USR1 | Owner | Schema | USR1 | Definition creation |
| | | USR1 | Owner | Table | USR1.T1 | Definition creation |
| | End | USR1 | -- | Table | USR1.T1 | Definition creation |
| A user (USR2) issues the following DROP TABLE:<br>DROP TABLE T1 | Privilege | USR2 | Owner | Table | USR2.T1 | Definition deletion |
| | End | USR2 | -- | Table | USR2.T1 | Definition deletion |

Legend:

Privilege: Audit trail is collected at time of privilege checking.

End: Audit trail is collected when the event terminates.

--: Not applicable

## (4) Information collected in audit trails

The following table lists and describes the information collected in audit trails.

Table 9–2: Information collected in audit trails

| Information collected | Description |
|---|---|
| User identifier | Authorization identifier of the executor of the audit event |
| Event execution date | Year, month, and date the event was executed |
| Event execution time | Time the event was executed |
| Event execution duration | Amount of time it took for the event to execute (in microseconds) |
| Event type | Event type |
| Event subtype | Event subtype |
| Event result | Execution results of the event (whether the privilege check was successful) |
| Privilege used | Privilege used when the event was executed |
| UAP name | UAP name specified in the PDCLTAPNAME operand of the client environment definition |
| Service name | Service name requested by the UAP that issued the event.<br>This is the item that corresponds to the service name when an OpenTP1 SUP (service using program) requests a service from an SPP (service providing program), or when TP1/Message Control requests a service from an MHP (message handling program). |
| IP address | Client IP address at which the UAP that issued the event is running# |
| Process number | Process ID from the UAP that issued the event# |
| Thread number | Thread ID from the UAP that issued the event# |
| Host name | Name of the host to which the UAP that issued the event is connected |
| Unit identifier | Identifier of the unit to which the UAP that issued the event is connected |
| User name | Name of the front-end server or single server to which the UAP that issued the event is connected |
| Connection sequence number | Connection sequence number of the event issuer |
| SQL sequence number | SQL sequence number of the event |
| Object owner name | Name of the owner of the object on which the event privilege check is performed |
| Object name | Name of the object on which the event privilege check is performed |
| Object type | Type of the object on which the event privilege check is performed |
| Assigned, revoked, or modified privilege | Privilege that was assigned, revoked, or modified due to the event |
| Identifier of the user who assigned, revoked, or modified a privilege, and the user identifier for the event | Identifier of the user who assigned, revoked, or modified the privilege with the event and the authorization identifier for the event |

| Information collected | Description |
|---|---|
| Values of security audit facility-related operands | Values of operands related to the security audit facility (values at HiRDB startup) |
| Audit trail type | Indicator of privilege check or event end |
| SQL code or end code | Code issued when the SQL, utility, or command ends |
| Swap source audit trail file name | Name of audit trail file at swap source when a swap occurs |
| Swap target audit trail file name | Name of audit trail file at swap target when a swap occurs |
| Configuration change type of connection security facility | Configuration change type set in the connection security facility (a change type is also set when the password is changed) |
| Values of operands related to connection security facility (before change) | Values of operands related to the connection security facility before they have been changed |
| Values of operands related to connection security facility (after change) | Values of operands related to the connection security facility after they have been changed |
| Audit trail table options | Flag for handling events that target an audit trail table, a view base table of an audit trail table, or a list base table of an audit trail table |
| Access count | Number of rows that are retrieved from, inserted into, updated in, or deleted from objects (base tables, view tables, and lists) by an event |
| SQL statement | SQL statement that was executed |
| SQL data | Data of the executed SQL |
| User-added information 1 | Additional information that is set up by users |
| User-added information 2 | |
| User-added information 3 | |
| Related product-added information 1 | Additional information that is set up by products related to Cosminexus or Hitachi Application Server. |

**Note**

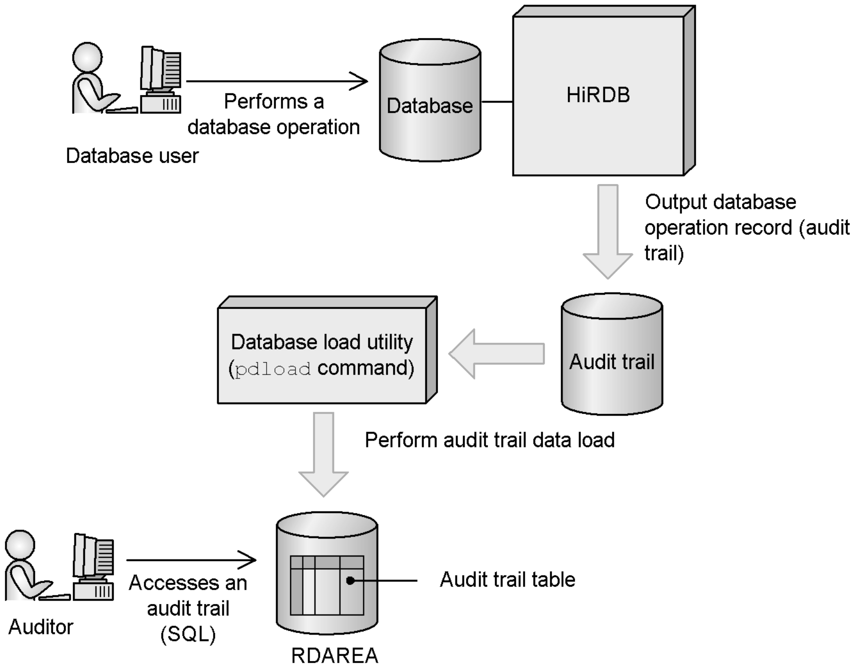
The information items that are collected depend on the event. For a list of the types of information that are collected for each event, see the *HiRDB Version 9 System Operation Guide*.

#: For events provided via an application running under OpenTP1 or provided via a Web server or similar product, information is collected from the application to which HiRDB is connected, not from the application that the end user is running.

## (5) Accessing an audit trail

Audit trails are output to an *audit trail file*. Data in an audit trail file can be accessed using SQL once the data has been loaded into an *audit trail table* by the database load utility (`pdload` command). Note that the auditor can access (but not update) this audit trail table. Users other than the auditor can access (but not update) an audit trail table only if they are granted access privilege by the auditor. The following figure shows how to access audit trails.

Figure 9–3: Accessing audit trails


Database user
Performs a database operation
Database
HiRDB
Output database operation record (audit trail)
Database load utility (pdload command)
Audit trail
Perform audit trail data load
Auditor
Accesses an audit trail (SQL)
Audit trail table
RDAREA

**Explanation**

1. When an audit event occurs, an audit trail is output to an audit trail file. The audit trail file is created in a HiRDB file system area allocated for audit trail files. For details about audit events, see *9.2.2 Audit events*.
2. Using as the input audit trails that were output to the audit trail file, the database load utility (`pdload` command) is used to load the data into a table. When the facility for automatically loading audit trail table data is applied, HiRDB automatically executes the database load utility.
3. The auditor uses the audit trail table to perform an audit.

## 9.2.2 Audit events

Operations that are collected in audit trails are called audit events. The following table lists and describes audit events.

Table 9–3: Audit events

| Event type | Description and list of audit events | Selectable? |
|---|---|---|
| System administrator security event | 1. Audits security events generated by HiRDB administrators and DBA privilege holders.<br>2. Audits changes to connection security facility settings.<br>3. Audits security events generated automatically by the system.<br>An audit trail is output when any of the following events is generated:<br>• HiRDB startup (`pdstart` command)#1<br>• HiRDB termination (`pdstop` command)#1, #2<br>• Auditor registration (`pdmod` command)<br>• Audit trail table creation (`pdmod` command)<br>• Audit trail file deletion (`pdaudrm` command)#3<br>• Start of audit trail collection#5<br>• End of audit trail collection#6<br>• Start of audit trail file overwrite | No (an audit trail is always output) |

| Event type | Description and list of audit events | Selectable? |
|---|---|---|
| | • Transition to consecutive certification failure account lock state<br>• Release of consecutive certification failure account lock state<br>This occurs at the following times:<br>● When CONNECT is executed after the account lock period expires<br>● When DROP CONNECTION SECURITY is executed<br>● When the pdacunlck command is executed<br>• Transition to password invalid account lock state<br>• Release of password invalid account lock state<br>• Change in a connection security facility setting:<br>● Permitted number of consecutive certification failures<br>● Account lock period<br>● Items set with password character string restrictions (including pre-checking)<br>• Execution of the pdacunlck command | |
| Auditor security event | Audits security events generated by the auditor. An audit trail is output when any of the following events is generated:<br>• Loading of data into an audit trail table (pdload command)<br>• Swapping of audit trail files (pdaudswap command)<br>• Definition of an audit event (CREATE AUDIT)#4<br>• Deletion of an audit trail event (DROP AUDIT)#4<br>• Changing an auditor password (GRANT AUDIT)#4<br>• Output of data to the audit log output file of JP1/NETM/Audit (pdaudput command) | No (an audit trail is always output) |
| Session security event | Audits events generated by user authentication based on an authorization identifier and password. An audit trail is output when either of the following events is generated:<br>• Connection to HiRDB (CONNECT statement)<br>• Changing users (SET SESSION AUTHORIZATION statement)<br>• Disconnection from HiRDB (DISCONNECT statement)#9 | Yes |
| Privilege control event | Audits events generated by granting and revoking user privileges. An audit trail is output when either of the following events is generated:<br>• Granting a user privilege (GRANT statement)<br>• Revoking a user privilege (REVOKE statement) | Yes#7 |
| Object definition event | Audits events generated by definition, deletion, or modification of objects. An audit trail is output when any of the following events is generated:<br>• Definition of an object; audits the following SQL statements:<br>CREATE FUNCTION<br>CREATE INDEX<br>CREATE PROCEDURE<br>CREATE PUBLIC VIEW<br>CREATE SCHEMA<br>CREATE SEQUENCE<br>CREATE TABLE<br>CREATE TRIGGER<br>CREATE TYPE<br>CREATE VIEW<br>• Deletion of an object; audits the following SQL statements:<br>DROP DATA TYPE | Yes#7 |

| Event type | Description and list of audit events | Selectable? |
|---|---|---|
| | DROP FUNCTION<br>DROP INDEX<br>DROP PROCEDURE<br>DROP PUBLIC VIEW<br>DROP SCHEMA<br>DROP SEQUENCE<br>DROP TABLE<br>DROP TRIGGER<br>DROP VIEW<br>• Modification of an object; audits the following SQL statements:<br>ALTER INDEX<br>ALTER PROCEDURE<br>ALTER ROUTINE<br>ALTER TABLE<br>ALTER TRIGGER<br>COMMENT | |
| Object operation event | Audits events generated by object manipulation. An audit trail is output when any of the following events is generated:<br>• Searching a table (SELECT statement)<br>• Insertion of rows in a table (INSERT statement)<br>• Updating of rows in a table (UPDATE statement)<br>• Deletion of rows from a table (DELETE statement)<br>• Deletion of all rows from a table (PURGE TABLE statement)<br>• Execution of a stored procedure (CALL statement)<br>• Locking a table (LOCK TABLE statement)<br>• Creation of a list (ASSIGN LIST statement)<br>• Return of values generated by the sequence generator (NEXT VALUE expression) | Yes#7 |
| Utility operation event | Audits security events generated from operations on objects by utilities or commands. An audit trail is output when any of the following events is generated:<br>• Database load command (pdload command)<br>Object: TABLE, SEQUENCE<br>• pddefrev command<br>Object: PROCEDURE, TABLE, TRIGGER, and VIEW<br>• Database reorganization utility (pdrorg command)<br>Object: TABLE<br>• Dictionary import/export utility (pdexp command)<br>Object: PROCEDURE, TABLE, TRIGGER, and VIEW<br>• Integrity check utility (pdconstck command)<br>Object: TABLE | Yes#7, #8 |

#1: In the case of a HiRDB parallel server configuration, startup and termination of a single server are not audit events.

#2: Normal termination and planned termination are audit events; forced termination and abnormal termination are not audit events. For these cases, check the messages output by HiRDB or the operating system.

The following termination commands are not monitored:

- `pdstop -f`
- `pdstop -f -q`

- `pdstop -f -x` *host-name*
- `pdstop -f -u` *unit-identifier*
- `pdstop -f -s` *server-name*
- `pdstop -f -u` *unit-identifier* `-s` *server-name*
- `pdstop -z`
- `pdstop -z -q`
- `pdstop -z -c`
- `pdstop -z -s` *server-name*

#3: Creation of an audit trail file is not an audit event. To audit creation of audit trail files, use the audit facility provided by the OS.

#4: You can also output an audit trail by executing the database definition utility (`pddef` command) or HiRDB SQL Executer.

#5: An audit trail is output when the `pdaudbegin` command is executed or when an audit trail is collected at HiRDB startup.

#6: An audit trail is output when the `pdaudend` command is executed or when an audit trail is collected during performance of normal or planned termination of HiRDB.

#7: An audit trail is output unconditionally when the event terminates in the case of privilege control events, object definition events, object operation events, and utility operation events that target an audit trail table, a view base table of an audit trail table, or a list base table of an audit trail table. You can select whether to collect an audit trail when a privilege check is performed.

#8: An audit trail is output unconditionally when the database reorganization utility (`pdrorg` command) is used to reload a dictionary table.

#9: Audit trail events include the following:

- The server process of a single server or front-end server detects a `DISCONNECT` statement.
- The server process of a single server or front-end server internally executes a `DISCONNECT` statement.

# 9.3 Connection security facility

This section provides a brief description of the connection security facility. For details about operating the connection security facility, see the *HiRDB Version 9 System Operation Guide*.

## 9.3.1 Overview of the connection security facility

One means of enhancing system security is to use passwords. HiRDB enables you to set a password for each user. However, if simple, easy-to-guess passwords are used (for example, one's authorization identifier or birth date), there is an increased possibility that a fraudulent user could use that password to gain access to the system. To prevent fraudulent use of passwords, use of the connection security facility is recommended. The following table provides an overview of the connection security facility.

Table 9–4: Overview of connection security facility

| Function | Description |
|---|---|
| Restrictions on password character strings | You can place restrictions on the character strings that can be specified as passwords. For example, you can prohibit passwords such as `012345` or `aaaaa`. Prohibiting simple passwords tends to increase password security. |
| Restrictions on the number of consecutive certification failures | If an incorrect password is entered a specified number of times in succession, you can bar that user from connecting (`CONNECT`) to HiRDB. To do so, you set the maximum number of times an incorrect password can be entered in consecutive connection attempts, and any user who exceeds the set number of attempts can no longer connect to HiRDB.<br>For example, you might permit a user three attempts to enter the correct password. On the fourth unsuccessful attempt, the user would be prohibited from connecting to HiRDB. |

By combining these two functions, fraudulent use of passwords based on ease of discovery can be made more difficult, which enhances system security.

## 9.3.2 Restrictions on password character strings

### (1) Restrictions that can be specified for passwords

The following table lists the restrictions that can be specified for passwords.

Table 9–5: Restrictions that can be specified for passwords

| Item | Description |
|---|---|
| Set a minimum password length | You can set a minimum length for a password (minimum number of bytes). |
| Prohibit specification of authorization identifier | You can prohibit use of one's authorization identifier as a password. |
| Prohibit only one type of characters# | You can prohibit passwords that consist of only one type of characters (for example, only uppercase alphabetic characters, or only numeric characters, etc.). |

#: Only the following types of characters can be used in passwords:

- Uppercase alphabetic characters: A to Z, #, @, \
- Lowercase alphabetic characters: a to z
- Numeric characters: 0 to 9

Reference note

You cannot set password character string restrictions for individuals users. The settings you specify here apply to all users of HiRDB (including DBA privilege holders and auditors). Note, however, that the password character string restrictions do not apply to simple authenticated users.

### (2) Effects on existing users

When you set restrictions on passwords, the account of any user who violates a restriction is placed in *password-invalid account lock state*. Users placed in this status can no longer connect (`CONNECT`) to HiRDB.

Such a user's password must be changed in order to release the password-invalid account lock state.

Before setting restrictions on passwords, you can check how many user accounts are likely to be placed in password-invalid account lock state due to restriction violations.

### (3) Effects on new users

If a password set for a new user with a `GRANT DBA`, `GRANT AUDIT`, or `GRANT CONNECT` statement violates a restriction, the `GRANT` statement will not execute.

### (4) Specification method

You use `CREATE CONNECTION SECURITY` to set restrictions on password character strings.

## 9.3.3 Restrictions on the number of consecutive certification failures

### (1) Restrictions that can be specified

You can specify that when an incorrect password is entered a specified number of times in succession, that user is to be prohibited from connecting (`CONNECT`) to HiRDB. You make this specification by setting a maximum number of consecutive connection failures (*permitted number of consecutive certification failures*). Any user who exceeds this number of unsuccessful password entry attempts will be barred from connecting to HiRDB.

For example, if you specify `3` for the permitted number of consecutive certification failures, any user's fourth consecutive failed attempt to gain user certification will place that user in *consecutive certification failure account lock state* Users placed in this status can no longer connect (`CONNECT`) to HiRDB.

Reference note

You cannot set restrictions on the number of consecutive certification failures for individual users. The settings you specify here apply to all users of HiRDB (including DBA privilege holders and auditors).

You can also set a time period during which an account will be kept in consecutive certification failure account lock state (*account lock period*). For example, if you set the account lock period at 1 hour, a consecutive certification failure account lock state will remain in effect for one hour. After one hour, the consecutive certification failure account lock state will be released and the user will be permitted once again to connect to HiRDB.

Reference note

- You can also set the account lock period to be indefinite (permanent).
- You can release an account from consecutive certification failure account lock state before the account lock period has expired.

### (2) Specification method

You use `CREATE CONNECTION SECURITY` to set restrictions on the number of consecutive certification failures.

# 10 Plug-ins

This chapter provides an overview of and explains the functions of the HiRDB plug-ins; it also explains the HiRDB preparations for using plug-ins.

# 10.1 Overview of HiRDB plug-ins

HiRDB is based on a *plug-in architecture*, in which *abstract data types* for multimedia data, such as documents and spatial data, and *facilities that enable simple, high-speed operations on data* can be used as a packaged product. With a plug-in architecture, facilities can be expanded to build a system that can handle multimedia data simply by registering plug-ins into HiRDB.

Moreover, application programs can handle multimedia data easily and at high speed without the programmer having to be concerned with complex operational details typical of multimedia data, and application programs can thus provide information services to customers efficiently. The following table lists the HiRDB plug-ins.

Table 10–1: HiRDB plug-ins

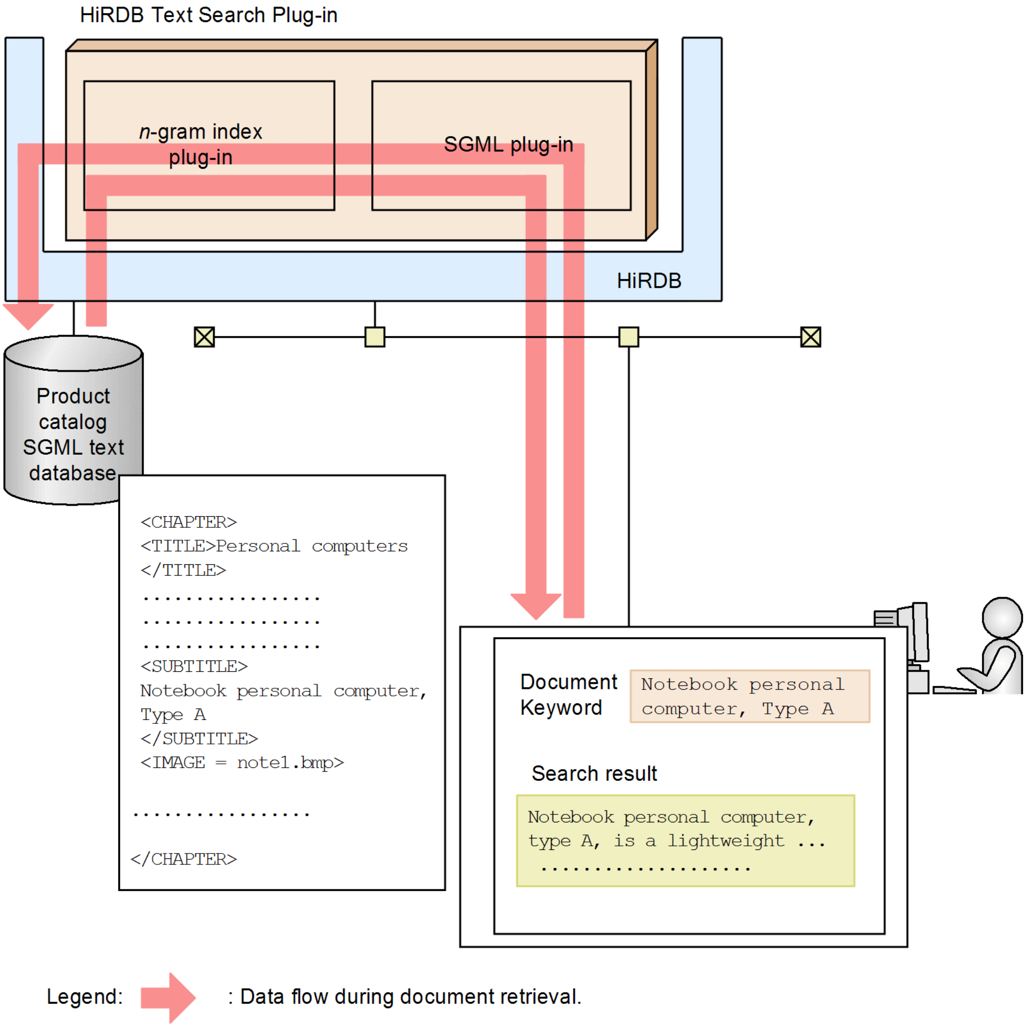
| Plug-in | Explanation |
|---|---|
| HiRDB Text Search Plug-in | Provides a function for retrieving a large volume of document information accurately and quickly. |
| HiRDB Spatial Search Plug-in | Provides a function for retrieving spatial data (two-dimensional data), such as map information. |
| HiRDB XML Extension | Provides a function for storing XML documents in the database while their structure is retained. |

For details about the HiRDB plug-in products, see the manuals for the applicable HiRDB plug-ins.

# 10.2 Applying a plug-in to a job

Assume for example that a company needs to use a database to manage data for its personal computer products catalog. The company's personal computer products catalog might contain document data on desktop machines, notebook computers, CPUs, and memory devices. When this document data is stored in a HiRDB database, and the HiRDB Text Search Plug-in is registered in HiRDB, it becomes possible to use keywords to retrieve document data very quickly. The following figure shows how to apply plug-ins to a job.

Figure 10–1: Application of plug-ins to a job


HiRDB Text Search Plug-in
*n*-gram index plug-in
SGML plug-in
HiRDB
Product catalog SGML text database
Document
Keyword
Notebook personal computer, Type A
Search result
Legend:
: Data flow during document retrieval.

```
<CHAPTER>
<TITLE>Personal computers
</TITLE>
.................
.................
.................
<SUBTITLE>
Notebook personal computer,
Type A
</SUBTITLE>
<IMAGE = note1.bmp>

.................

</CHAPTER>
```

```
Notebook personal computer,
type A, is a lightweight ...
 ....................
```

**Explanation**

- This example uses the HiRDB Text Search Plug-in (SGML plug-in and n-gram index plug-in), which is a plug-in package product.
- It is possible to retrieve documents using the keyword `notebook personal computer, type A`.

# 10.3 HiRDB plug-in facilities

HiRDB provides the following plug-in products:

- HiRDB Text Search Plug-in
- HiRDB Spatial Search Plug-in
- HiRDB XML Extension

The benefits of each of these plug-ins are explained as follows.

## 10.3.1 HiRDB Text Search Plug-in

The HiRDB Text Search Plug-in provides the following text structure search facilities:

- SGML and XML document registration
- Flat document registration
- Retrieval with structure specification
- Synonym and spelling variation retrieval
- Score retrieval
- Retrieval result text data extraction
- Embedding of retrieval hit location highlighting tag

Each of these facilities is explained as follows.

### (1) SGML and XML document registration

A utility provided by the HiRDB Text Search Plug-in can be used to register into the HiRDB database a DTD file, which defines tag names that indicate the structure and elements of an SGML and XML document. When the SGMLTEXT constructor facility based on the registered DTD file is used, the SGML and XML document together with the document structure information can be registered into the HiRDB database.

### (2) Flat document registration

Flat documents that do not have a structure can be registered into the HiRDB database.

### (3) Retrieval with structure specification

A full-text search for SGML and XML documents can be performed by using the `contains` abstract data type function to specify the columns to be retrieved and the retrieval condition (document structure name to be retrieved, a conditional expression that specifies the keyword to be retrieved).

### (4) Synonym and spelling variation retrieval

A utility provided by the HiRDB Text Search Plug-in can be used to register synonyms and spelling variations into a local file. Based on the registered synonyms and spelling variations, synonyms or spelling variations of a keyword used for a search can also be retrieved during a full-text search for SGML documents. For example, a search for the keyword *computer* can also retrieve a synonym such as *electronic computing machine* and spelling variations such as *COMPUTER* and *Computer*.

### (5) Score retrieval

The `contains_with_score` and score abstract data type facilities provided by the HiRDB Text Search Plug-in can be used to compute points (scores) based on the frequency of occurrence of the keyword and to display the retrieval results according to the scores.

## 10.3.2 HiRDB Spatial Search Plug-in

The HiRDB Spatial Search Plug-in provides the following facilities:

- Spatial data registration
- Spatial data retrieval

### (1) Spatial data registration

The GEOMETRY abstract data type facility provided by the HiRDB Spatial Search Plug-in makes it possible to define locations such as train stations and banks in a map as spatial data (two-dimensional data with X and Y coordinates specified). The data can also be registered in the HiRDB database.

### (2) Spatial data retrieval

The Within abstract data type facility provided by the HiRDB Spatial Search Plug-in can be used to evaluate the spatial relationships between the graphic object specified in a retrieval condition expression and a GEOMETRY-type graphic object, and to determine whether the specified graphic object is included. For example, one could retrieve *banks located within 100 m of the train station*.

The IntersectsIn abstract data type facility can also be used to evaluate the spatial relationships between the graphic object specified in a retrieval condition expression and a GEOMETRY-type graphic object, and to determine whether the specified graphic object is included or intersects. For example, one could retrieve *roads, train tracks, and rivers that pass through a certain area*.

## 10.3.3 HiRDB XML Extension

The XML Extension provides the XML type, an abstract data type that lets you use data directly in XML format, which is a standard format for exchanging content and interfacing to services.

Because the XML type does not require data structure definitions, you do not need to change table definitions even if you change the XML type data structure. This extension facilitates management of data by letting you take advantage of the high flexibility of XML. It allows you to minimize system changes and thus reduce the impact on business operations even when changes are made to the massive stocks of documentation owned by companies and various kinds of research institutes as intellectual property or to the data structures of content that is used on a regular basis.

Furthermore, with respect to searching, the extension supports key features of SQL/XML that use XQuery, the standard XML query language.

# 10.4 Preparations for using plug-ins in HiRDB

The preparations explained in this section must be made in order to use plug-ins in HiRDB.

- Setup and registration of a plug-in
  To use a plug-in, it must be set up and registered into HiRDB.
- Initialization of the registry facility
  The registry facility must be initialized before registry information can be registered.
- Table definition for plug-in usage
  An object relational database must be created using the abstract data types and index types provided by the plug-in.
- Delayed batch creation of plug-in index
  Instead of adding (updating) plug-in index data when row data is added (updated), it is possible to use the database reorganization utility later to add (update) the plug-in index data in a batch.

Each of these preparatory items for using plug-ins is explained as follows.

## 10.4.1 Setup and registration of a plug-in

To use a plug-in, the plug-in must first be installed, and then it must be set up and registered in the HiRDB environment (in the Windows edition, setup of the plug-in is not required).

A plug-in is set up in the HiRDB environment with the `pdplgset` command, and it is registered with the `pdplgrgst` command.

For details about how to set up and register plug-ins, see the *HiRDB Version 9 Installation and Design Guide*. For details about installing plug-ins, see the documentation for the applicable plug-ins.

## 10.4.2 Initialization of the registry facility

An SGML document might have a structure such as chapter, section, item, and list in the document itself. If the individual items of data have structures, the user must enable the plug-in to recognize each data item's structure when the document is registered. This type of information that is specific to a plug-in and that is to be used by the plug-in for data manipulation is called *registry information*. The facility by which HiRDB maintains registry information is called the *registry facility*.

Before the registry facility can be used for a plug-in, it must be initialized. HiRDB's registry facility initialization utility (`pdreginit`) is used for this purpose.

Initializing the registry facility creates a registry RDAREA and a registry LOB RDAREA and stores in HiRDB RDAREAs a table for managing registry information (registry management table) and data manipulation stored procedures (such as for registering information into the registry management table).

The following table shows the RDAREAs used for storing information related to the registry facility.

Table 10–2: RDAREAs for storing information related to registry facility

| Registry facility information | Storage RDAREA |
|---|---|
| Registry management table | Registry RDAREA or registry LOB RDAREA# |
| Data manipulation stored procedures, such as for registering information into the registry management table | Data dictionary LOB RDAREA |

#: The RDAREA to be used for storing information is determined automatically on the basis of the length of the data to be registered (up to 32,000 bytes is stored in the registry RDAREA; 32,000 bytes or longer is stored in the registry LOB RDAREA).

For details about registry facility initialization, see the *HiRDB Version 9 Installation and Design Guide*. For details about the registry facility setup utility (`pdreginit`), see the manual *HiRDB Version 9 Command Reference*.

## 10.4.3 Table definition for plug-in usage

Once a plug-in has been set up and registered, it provides abstract data types and index types, thus enabling the user to easily build an object relational database that can handle complex data, such as multimedia information, without having to define a complex data structure or operations using abstract data types.

To define a table, an abstract data type provided by the plug-in is specified as the data type of the column in the table. To define an index, an index type provided by the plug-in is specified as the index. An index for which an index type provided by a plug-in is specified is called a *plug-in index*. The specifics of the plug-in index facility depend on the particular plug-in; for details, see the manual for the applicable plug-in.

For details about designing and creating an object relational database to be used with plug-ins, see the *HiRDB Version 9 Installation and Design Guide*.

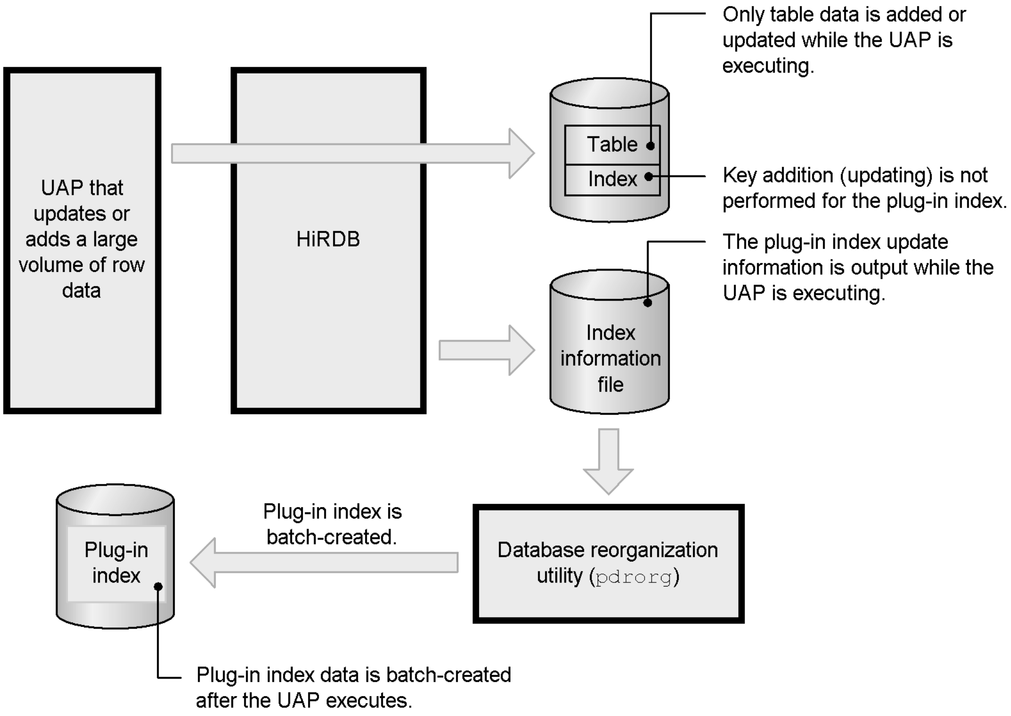
## 10.4.4 Delayed batch creation of plug-in index

Addition or updating of row data in a table for which a plug-in index is defined requires that keys be added to the plug-in index. When a large amount of row data is added or updated, the resulting addition of keys to the plug-in index can degrade performance.

Instead of adding the plug-in index data at the time the row data is added, the database reorganization utility (`pdrorg`) can be used at a later time to add the plug-in index data in a batch; this is called *delayed batch creation of a plug-in index*. The figure below shows delayed batch creation of a plug-in index.

When the system switchover facility is being used, the index information file must be created on the shared disk. In addition, when the system switchover facility is being used in the UNIX edition, the index information file must be created in a HiRDB file system area (character special file) on a shared disk.

Figure 10–2: Delayed batch creation of plug-in index


Only table data is added or updated while the UAP is executing.
UAP that updates or adds a large volume of row data
HiRDB
Table
Index
Key addition (updating) is not performed for the plug-in index.
The plug-in index update information is output while the UAP is executing.
Index information file
Plug-in index
Plug-in index is batch-created.
Database reorganization utility (pdrorg)
Plug-in index data is batch-created after the UAP executes.

**Prerequisite**

It must be determined whether the plug-in being used supports delayed batch creation of plug-in index. Delayed batch creation of plug-in index cannot be used for a plug-in that does not support this facility.

The HiRDB Text Search Plug-in supports delayed batch creation of plug-in index.

**Advantage**

Because data is not added to the plug-in index, the execution time (table data creation time) for a UAP that adds or updates a large volume of data can be reduced considerably.

**Operating procedure**

For details about the operating procedure for using delayed batch creation of plug-in index, see the *HiRDB Version 9 System Operation Guide*.

# Appendixes

# A. Functional Differences Between HiRDB Versions on Different Platforms

Table A-1 lists the functional differences between HiRDB versions on different platforms. Note that some facilities cannot be used, depending on the operating environment. For details, check the description of each facility.

Table A–1: Functional differences between HiRDB versions on different platforms

| Item | Facility, operand, or command | HP-UX | Solaris | AIX | Linux | Windows |
|---|---|---|---|---|---|---|
| Facilities | Facility for monitoring the memory size of server processes | Y | Y | Y | N | N |
| | Utility special unit | Y | Y | Y | Y | N |
| | Sharing of HiRDB system definitions | Y | Y | Y | Y | N |
| | Shell script for creating an unload statistics log file on a specific server machine (`pdstjacm`) | Y | Y | Y | Y | N |
| | Facility for parallel output of system logs | N | N | Y | N | N |
| | Start and termination of extended units of active HiRDB | N | N | N | Y | N |
| Option program product related | Inner replica facility (HiRDB Staticizer Option) | Y | Y | Y | Y | N |
| | Updatable online reorganization (HiRDB Staticizer Option) | Y | Y | Y | Y | N |
| Linkage to related products | Real Time SAN Replication | Y | N | Y | N | N |
| | Online distribution of HiRDB by using JP1/NETM/DM | Y | Y | N | N | Y |
| | Linkage to OLTP (TPBroker for C++) | Y | Y | N | N | Y |
| | Linkage to OLTP (TUXEDO) | N | Y | N | N | Y |
| | Linkage to OLTP (WebLogic Server) | Y | Y | N | N | Y |
| | Linkage to OLTP (TP1/EE) | N | N | Y | Y | N |
| | EasyMT, MTguide | Y | N | Y | N | N |
| HiRDB system definitions | Specification of the maximum number of consecutive abnormal terminations during HiRDB restart processing<br>`pd_term_watch_count` | Y | Y | Y | Y | N |
| | Specification of the maximum number of files and pipes that can be accessed by HiRDB processes<br>`pd_max_open_fds` | Y | Y | Y | Y | N |
| | Specification of the maximum memory size to use for a server process<br>`pd_svr_castoff_size` | Y | Y | Y | N | N |
| | Specification of the maximum size of the communication buffers used for server intra-unit communication (TCP-UNIX domain)<br>`pd_ipc_unix_bufsize` | Y | Y | Y | Y | N |

| Item | Facility, operand, or command | HP-UX | Solaris | AIX | Linux | Windows |
|---|---|---|---|---|---|---|
|  | Specification of the maximum size of the communication buffers used for communication with the HiRDB clients (TCP-UNIX domain)<br>`pd_tcp_unix_bufsize` | Y | Y | Y | Y | N |
| Commands | `pddbchg` | Y | Y | Y | Y | N |
|  | `pddivinfgt` | N | N | N | Y | N |
|  | `pdgeter` | Y | Y | Y | Y | N |
|  | `pditvstop` | Y | Y | Y | Y | N |
|  | `pditvtrc` | Y | Y | Y | Y | N |
|  | `pdlodsv` | Y | Y | Y | N | N |
|  | `pdmemsv` | Y | Y | Y | Y | N |
|  | `pdopsetup` | Y | Y | Y | Y | N |
|  | `pdorbegin` | Y | Y | Y | Y | N |
|  | `pdorcheck` | Y | Y | Y | Y | N |
|  | `pdorchg` | Y | Y | Y | Y | N |
|  | `pdorcreate` | Y | Y | Y | Y | N |
|  | `pdorend` | Y | Y | Y | Y | N |
|  | `pdplgset` | Y | Y | Y | Y | N |
|  | `pdrpause` | Y | Y | Y | Y | N |
|  | `pdsetup` | Y | Y | Y | Y | N |
|  | `pdkill` | N | N | N | N | Y |
|  | `pdntenv` | N | N | N | N | Y |
|  | Acquisition of a core file with the `pdcancel` command | Y | Y | Y | Y | N |
|  | EasyMT date check with the `pdload` command | Y | N | Y | N | N |
|  | Use or fixed-length tape with the `pdcopy`, `pdload`, `pdrstr`, and `pdrorg` commands | Y | Y | Y | N | Y |
| Client | ODBC Driver | N | N | N | Y | Y |
|  | OLE DB Provider | N | N | N | N | Y |
|  | HiRDB.NET Data Provider | N | N | N | N | Y |
|  | Third-party COBOL support facilities | N | Y | N | N | N |
|  | XA interface for the XDM/RD E2 connection facility | Y | Y | Y | Y | Y |
|  | Multi-connection facility (kernel threads) | Y | Y | Y | Y | N |

Legend:

Y: Supported

N: Not supported

# B. Data Dictionary Tables

HiRDB creates and manages the data dictionary tables that store table, index, and other types of definition information. Data manipulation SQL statements are used to reference data dictionary tables in order to check and confirm table, index, and other types of definition information.

The following table lists the HiRDB data dictionary tables. For examples of SQL coding for referencing data dictionary tables and columns in the data dictionary tables, see the *HiRDB Version 9 UAP Development Guide*.

Table B–1: Data dictionary tables

| No. | Table name | Contents | Information coverage (per row) |
|---|---|---|---|
| 1 | SQL_PHYSICAL_FILES | HiRDB file information (correspondence between HiRDB file system name and RDAREA names) | One HiRDB file |
| 2 | SQL_RDAREAS | Information such as RDAREA name, definition information, RDAREA type, number of tables stored, and number of indexes | One RDAREA |
| 3 | SQL_TABLES | Owner and table names of tables in a database (including dictionary tables) | One table |
| 4 | SQL_COLUMNS | Definition information related to a column, such as column name and data type | One column |
| 5 | SQL_INDEXES | Owner and index names of indexes in a database (including dictionary tables) | One index |
| 6 | SQL_USERS | Authorization identifier of the user who granted a user execution privilege and database access | One user |
| 7 | SQL_RDAREA_PRIVILEGES | RDAREA usage privilege status | One RDAREA of One authorization identifier |
| 8 | SQL_TABLE_PRIVILEGES | Table access privilege granting status | One table of 1 authorization identifier |
| 9 | SQL_VIEW_TABLE_USAGE | Name of base table on which a view table is based | One view table |
| 10 | SQL_VIEWS | View definition information | One view table |
| 11 | SQL_DIV_TABLE | Table division information (division condition and name of RDAREA for storage, as specified in CREATE TABLE) | One table per *n* rows |
| 12 | SQL_INDEX_COLINF | Name of column to which an index is assigned | One index per *n* rows |
| 13 | SQL_DIV_INDEX | Index division information (name of RDAREA for storage) | One index per *n* rows |
| 14 | SQL_DIV_COLUMN | BLOB type column division information (name of RDAREA for storage, as specified in CREATE TABLE) | One column per *n* rows |
| 15 | SQL_ROUTINES | Routine definition information | One routine per row |
| 16 | SQL_ROUTINE_RESOURCES | Information on resources used in a routine | One routine per *n* rows |
| 17 | SQL_ROUTINE_PARAMS | Parameter definition information in a routine | One routine per *n* rows |
| 18 | SQL_TABLE_STATISTICS | Table statistical information | One table |
| 19 | SQL_COLUMN_STATISTICS | Column statistical information | One column |
| 20 | SQL_INDEX_STATISTICS | Index statistical information | One index |

| No. | Table name | Contents | Information coverage (per row) |
|---|---|---|---|
| 21 | SQL_DATATYPES | User-defined type information | One user-defined type |
| 22 | SQL_DATATYPE_DESCRIPTORS | Information on an attribute that comprises a user-defined type | One attribute |
| 23 | SQL_TABLE_RESOURCES | Resource information to be used in a table | One resource |
| 24 | SQL_PLUGINS | Plug-in information | One plug-in |
| 25 | SQL_PLUGIN_ROUTINES | Plug-in routine information | One plug-in routine |
| 26 | SQL_PLUGIN_ROUTINE_PARAMS | Parameter information of a plug-in routine | One parameter information item |
| 27 | SQL_INDEX_TYPES | Index type information | One index type |
| 28 | SQL_INDEX_RESOURCES | Resource information to be used for an index | One resource information item |
| 29 | SQL_INDEX_DATATYPE | Index item information | One applicable information item (one step) |
| 30 | SQL_INDEX_FUNCTION | Information on an abstract data type facility to be used for an index | Information on One abstract data type facility |
| 31 | SQL_TYPE_RESOURCES | Resource information to be used in a user-defined type | One resource information item |
| 32 | SQL_INDEX_TYPE_FUNCTION | Information on an abstract data type facility that can be used by an index for which index type has been defined | One index per *n* rows |
| 33 | SQL_EXCEPT | Information on index exception key values | Exception key group of one index per *n* rows |
| 34 | SQL_IOS_GENERATIONS | UNIX edition:<br>Generation information on the HiRDB file system areas when the inner replica facility is being used<br>Windows edition:<br>Information used by the system (containing no data) | UNIX edition:<br>1 HiRDB file system area per row<br>Windows edition:<br>None |
| 35 | SQL_TRIGGERS | Trigger information in schemas | One trigger per row |
| 36 | SQL_TRIGGER_COLUMNS | Event column list information on UPDATE triggers | One event column data item per row |
| 37 | SQL_TRIGGER_DEF_SOURCE | Definition source information on triggers | One trigger definition source data item per *n* rows |
| 38 | SQL_TRIGGER_USAGE | Resource information referenced by trigger action conditions | One resource name referenced by the trigger action condition per row |
| 39 | SQL_PARTKEY | Information on partitioning keys of matrix-partitioned tables | One partitioning key data item per row |
| 40 | SQL_PARTKEY_DIVISION | Information on partitioning condition values of matrix-partitioned tables | One partitioning condition value data item per row |

| No. | Table name | Contents | Information coverage (per row) |
|---|---|---|---|
| 41 | `SQL_AUDITS` | Information on audit targets | One event data item for an object or one user per row |
| 42 | `SQL_REFERENTIAL_CONSTRAINTS` | Match state of referential constraints | One constraint per row |
| 43 | `SQL_KEYCOLUMN_USAGE` | Information on columns comprising foreign keys | One column per row |
| 44 | `SQL_TABLE_CONSTRAINTS` | Information on integrity constraints in schemas | One integrity constraint per row |
| 45 | `SQL_CHECKS` | Information on check constraints | One check constraint per row |
| 46 | `SQL_CHECK_COLUMNS` | Information on columns used by check constraints | One column used by a check constraint per row |
| 47 | `SQL_DIV_TYPE` | Partitioning key information on partitioning keys in matrix-partitioned tables, combining key range partitioning and hash partitioning | One partitioning key per row |
| 48 | `SQL_SYSPARAMS` | Information on the number of consecutive certification failures permitted and on password character string restrictions | One setting item per row, or one permitted number of consecutive certification failure specifications, or one password character string restriction specification per *n* rows |
| 49 | `SQL_INDEX_XMLINF` | Information on the substructure path comprising the index of substructure indexes | One index per row |
| 50 | `SQL_SEQUENCES` | Sequence generator information | Data on one sequence generator per row |

# C. HiRDB Client and HiRDB Server Connectivity

HiRDB clients and HiRDB servers can be connected even though their versions or platforms are different. Tables C-1 and C-2 show HiRDB client and server connectivity.

A HiRDB server and a HiRDB client that have the right connectivity might not be able to actually connect if the client does not support compatible character encodings.

Table C–1: HiRDB client and HiRDB server connectivity (HiRDB server Version 5.0 or later)

| HiRDB client version (type) | | HiRDB server version (type) | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Version 7 or later | | | | | | Version 6 | | | | | | Version 5.0 | | | | | |
| | | E | H | S | A | L | W | E | H | S | A | L | W | E | H | S | A | L | W |
| Version 7 or later | E | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | S | Y | Y | Y | Y | Y | Y | Y | Y | D | Y | Y | Y | Y | Y | D | Y | -- | -- |
| | A | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | L | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Version 6 | E | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | S | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | A | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- |
| | L | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Version 5.0 | E | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | S | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | L | Y | Y | Y | Y | Y | -- | Y | Y | Y | Y | Y | -- | Y | Y | Y | Y | Y | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Version 4.0<br>04-03 or later | E | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | S | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Version 4.0<br>04-02 or earlier | E | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Version<br>03-03 or earlier | E | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |

| HiRDB client version (type) | | HiRDB server version (type) | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Version 7 or later | | | | | | Version 6 | | | | | | Version 5.0 | | | | | |
| | | E | H | S | A | L | W | E | H | S | A | L | W | E | H | S | A | L | W |
| | S | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- | Y | Y | Y | Y | -- | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| VOS3 client | V | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | -- | -- | -- | -- |

E: HI-UX/WE2 edition

H: HP-UX edition

S: Solaris edition

A: AIX edition

L: Linux edition

W: Windows edition

V: VOS3 edition

Y: Connectable

D: Depends on the HiRDB client version, as follows:

- Earlier than Version 07-00-/C
  Connectable
- Version 07-00-/C or later
  Connectable when the HiRDB client and the HiRDB server are on separate server machines. Not connectable if they are on the same server machine.

--: Not connectable

Table C–2: HiRDB client and HiRDB server connectivity (HiRDB server Version 4.0 or older)

| HiRDB client version (type) | | HiRDB server version (type) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Version 4.0 | | | | | | | | Version 03-03 or older | | | |
| | | Version 04-03 or later | | | | | Version 04-02 or older | | | | | | |
| | | E | H | S | A | W | E | H | W | E | H | S | W |
| Version 7 or later | E | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | S | Y | Y | D | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | L | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | W | Y | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- |
| Version 6 | E | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | S | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | L | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | W | Y | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- |

| HiRDB client version (type) | | HiRDB server version (type) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Version 4.0 | | | | | | | | Version 03-03 or older | | | |
| | | Version 04-03 or later | | | | | Version 04-02 or older | | | | | | |
| | | E | H | S | A | W | E | H | W | E | H | S | W |
| Version 5.0 | E | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | S | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | L | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | W | Y | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- |
| Version 4.0 04-03 or later | E | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | H | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | S | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | A | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- |
| | W | Y | Y | Y | Y | Y | -- | -- | -- | -- | -- | -- | -- |
| Version 4.0 04-02 or older | E | Y | Y | Y | Y | -- | C | C | -- | -- | -- | -- | -- |
| | H | Y | Y | Y | Y | -- | C | C | -- | -- | -- | -- | -- |
| | W | Y | Y | Y | Y | Y | C | C | C | -- | -- | -- | -- |
| Version 03-03 or older | E | Y | Y | Y | Y | -- | Y | Y | -- | C | C | C | -- |
| | H | Y | Y | Y | Y | -- | Y | Y | -- | C | C | C | -- |
| | S | Y | Y | Y | Y | -- | Y | Y | -- | C | C | C | -- |
| | W | Y | Y | Y | Y | Y | Y | Y | Y | C | C | C | C |
| VOS3 client | V | Y | Y | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |

E: HI-UX/WE2 edition

H: HP-UX edition

S: Solaris edition

A: AIX edition

L: Linux edition

W: Windows edition

V: VOS3 edition

Y: Connectable

D: Depends on the HiRDB client version, as follows:

- Earlier than Version 07-00-/C
  Connectable
- Version 07-00-/C or later
  Connectable when the HiRDB client and the HiRDB server are on separate server machines. Not connectable if they are on the same server machine.

C: Limited connectivity; can be connected if the version of the HiRDB server is the same as or higher than the version of the HiRDB client. In other cases, cannot be connected except in the following cases:

- HiRDB client 04-01 can be connected to HiRDB server 04-00
- HiRDB client 03-02 can be connected to HiRDB server 03-01
- HiRDB client 02-06 can be connected to HiRDB server 02-05
- HiRDB client 02-04 can be connected to HiRDB server 02-03 (but not in the Windows NT edition)
- HiRDB clients 02-05 and 02-06 can be connected to HiRDB server 02-03 (Windows NT edition only)

--: Not connectable

# D. List of Facilities That Are No Longer Supported

HiRDB no longer supports the facilities listed below. Therefore, explanations of these facilities have been deleted from the respective manuals.

- HiRDB External Data Access facility
- Development of external Java stored routines using JBuilder
- JP1/OmniBack II linkage facility
- System generator (`pdgen`)
- Directory Server linkage facility
- Linkage to data-mining products (`DATAFRONT`)
- HiRDB Image Search Plug-in
- Distributed database facility
- HiRDB File Link

# E. Glossary

This Appendix defines terms used in the HiRDB manuals.

### abstract data type

An abstract data type is a unique data type defined by the user so that data with a complex structure and operations involving it can be handled in SQL.

### abstract data type column structure base table

A table containing abstract data type columns and from which the abstract data type columns have been removed is called an abstract data type column structure base table.

### accepting unit

A unit in which is located a guest BES when the standby-less system switchover (effects distributed) facility is used.

### access privilege

An access privilege is a permission that enables you to access a table; access privileges are set on a table-by-table basis. There are four access privileges:

- SELECT privilege
- INSERT privilege
- DELETE privilege
- UPDATE privilege

### account lock period

Period during which a consecutive certification failures account lock state is to be in effect (will be detected by the connection security facility).

### ADT (Abstract Data Type)

See *abstract data type*.

### aio library

An aio library is an OS-provided library that houses a group of APIs for asynchronously inputting or outputting files. In AIX, Asynchronous I/O Subsystem is its aio library; in Linux, libaio is its aio library. The facility for parallel output of system logs uses the API provided by the aio library to issue a request for parallel output of duplexed system log files.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### alias IP address

A technique by which different IP addresses can share the same LAN adapter through assignment of multiple IP addresses to the LAN adapter.

### all asynchronous method

One of the processing methods used in Real Time SAN Replication. When an update of a database file or system file occurs at the main site, copying to the corresponding file at the remote site is performed asynchronously.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### all synchronous method

One of the processing methods used in Real Time SAN Replication. When an update of a database file or system file occurs at the main site, copying to the corresponding file at the remote site is performed synchronously.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### alternate BES

A back-end server that takes over processing when an error occurs while the standby-less system switchover (1:1) facility is being used. Similarly, a unit containing an alternate BES is called an alternate BES unit.

### asynchronous copy

One of the processing methods used to update-copy data to a remote site. Update-copying at the main site is completed without waiting for update-copying at the remote site to be completed.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### asynchronous DB state

In in-memory data processing, synchronous DB state means that data in the in-memory RDAREA is not synchronized with data in the in-memory buffer.

### asynchronous group

A group made up entirely of asynchronous pair volumes.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### asynchronous pair volume

A pair volume that has been created by specifying `async` as the fence level. When data is written to the P-VOL, it is not mirrored synchronously onto the S-VOL. Thus, differences might arise between the data on the P-VOL and the data on the S-VOL, even though they are paired.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### asynchronous READ facility

When the prefetch facility is used, the asynchronous READ facility provides two buffer sectors for batch input. While DB processing is using one buffer, an asynchronous READ process prefetches data from the other buffer asynchronously with respect to the DB processing. By overlapping the DB processing with the prefetch input, you can reduce the processing time.

### audit privilege

A privilege required for auditors. A user needs the audit privilege to perform operations such as the following:

- Loading data to audit trail tables
- Swapping audit trail files
- Searching, updating, and deleting audit trail files

### audit trail

Operations performed on HiRDB such as audit commands and SQL execution records are output to a file. This record is called an audit trail, and the file to which this record is output is called an audit trail file.

### audit trail table

A table used by the auditor to perform audits. It is created by loading into it information saved in an audit trail file.

### auditor

A person who audits a HiRDB system. An auditor must be assigned the audit privilege. Only one auditor can be registered.

### authorization identifier

A name for identifying a set of privileges. The privilege of operating a database is assigned to an authorization identifier.

In HiRDB, the name that identifies a HiRDB user is used as an authorization identifier, and during a database operation, HiRDB checks whether the executing user's authorization identifier has the privilege necessary for the operation.

### automatic log unloading facility

Normally, when system log information is unloaded by the HiRDB administrator using the `pdlogunld` command, it is also necessary to unload any system log file that is waiting to be unloaded. The facility that automates the process of unloading system log files is called the automatic log unloading facility.

### automatic numbering facility

When it executes a numbering job, the automatic numbering facility automatically assigns numbers generated by the sequence generator.

### automatic reconnect facility

When a connection to a HiRDB server is lost due to a server process going down, system switchover, network failure, or other cause, this capability automatically re-establishes the connection. By using the automatic reconnect facility, a user can continue executing the UAP without having to be aware that the connection to the HiRDB server was lost.

### back-end server

A back-end server is a constituent element of a HiRDB parallel server configuration that performs database access, locking, and operational processing in accordance with execution instructions received from a front-end server.

### backup acquisition mode

The backup acquisition mode allows you to specify whether referencing and updating requests from other UAPs for an RDAREA being backed up are to be accepted while a backup is being made by the database copy utility. The backup acquisition mode is specified in the -M option of the database copy utility; there are three modes:

- Referencing/updating-impossible mode (-M x specification)
  During backup, an RDAREA being backed up cannot be referenced or updated.
- Referencing-permitted mode (-M r specification)
  During backup, an RDAREA being backed up can be referenced but not updated.
- Updatable mode (-M s specification)
  During backup, an RDAREA being backed up can be both referenced and updated.

### backup file

A backup file stores a copy of an RDAREA that can be used to recover the RDAREA after a failure.

### backup-hold

When a backup is made using a command other than the `pdcopy` command (backup using a function provided by another product), place the RDAREA to be backed up on backup-hold. An RDAREA on backup-hold can be backed up by the backup facility of another product while HiRDB is running. To apply backup-hold to an RDAREA, specify the `-b` option in the `pdhold` command. There are four types of backup-hold, as shown in the following table.

Table E–1: Types of backup-hold

| Types of backup-hold | Explanation |
|---|---|
| Referencing-permitted backup-hold | During backup-hold, RDAREAs that have been backed up and held can be referenced; an updating attempt will result in an SQL error (-920). |
| Referencing-permitted backup-hold (update WAIT mode) | During backup-hold, RDAREAs that have been backed up and held can be referenced; an updating attempt goes onto lock-release wait status until the backup-hold is released. |
| Updatable backup-hold | During backup-hold, RDAREAs that have been backed up and held can be referenced and updated. Even while an updating transaction is being executed, an RDAREA is placed immediately on updatable backup-hold status without placing the `pdhold` command on wait status. |
| Updatable backup-hold (WAIT mode) | During backup-hold, RDAREAs that have been backed up and held can be referenced and updated. If an updating transaction is being executed, the `pdhold` command is kept waiting until the transaction terminates. |

The following types of backup-hold are referred to collectively as *committing a database*:

- Referencing-permitted backup hold
- Referencing-permitted backup hold (update WAIT mode)

### base row

The row that contains the base data for all columns. If a branch row has been created, the base row contains the branch target information.

### base table

A table that is not a viewed table. A base table can be either a permanent base table or a temporary table.

### BLOB (Binary Large Object)

BLOB is large data, such as large text, image, and audio data.

### BLOB data file output facility

With this facility, instead of retrieved BLOB data being returned to the client, it is output to a file in a unit that contains either a single server or a front-end server, and only the name of the file is returned to the client. Use of this facility prevents large increases in memory use during data retrieval operations.

### BLOB or BINARY data, partial update or retrieval of

The facility for partial update or retrieval of `BLOB` or `BINARY` data consists of the three functions described below. Data with a defined length of 32,001 bytes or more is handled as `BINARY` data.

- Function for specifying a concatenation operation in the `UPDATE` statement's `SET` clause in order to add new data to registered `BLOB` or `BINARY` data

- Function for specifying the SUBSTR scalar function in order to extract only a specified portion of BLOB or BINARY data
- SUBSTR scalar function in which the constant 1 is specified for the processing target column and start position in the SET clause of the UPDATE statement. This allows you to delete only the back-end portion of BLOB or BINARY data.

This facility can reduce the memory requirement to the amount of BLOB or BINARY data that is to be added or extracted.

## block transfer facility

The block transfer facility transfers data from any number of rows from a HiRDB server to a HiRDB client.

## branch row

Data is partitioned onto another page when its length exceeds one page. Note that HiRDB sometimes partitions the data in this way even when the no-split option is not specified. Rows that are stored on the other page in this manner are called *branch rows*.

## buffer error

In in-memory data processing, a buffer error means that an error has occurred at the in-memory data buffer.

## C stored function

A stored function whose processing procedure is coded in C language.

## C stored procedure

A stored procedure whose processing procedure is coded in C language.

## changing the partition storage conditions

Use of ALTER TABLE to change the partition storage conditions for a key range partitioned table. By changing a table's partition storage conditions, RDAREAs containing old data can be reused, thereby saving time.

Note:

You can use ALTER TABLE to change the partition storage conditions of tables that have been partitioned using one of the following methods:

- Boundary value specification
- Storage condition specification (when the equal sign (=) was used as the storage condition comparison operator)

## character code set

Character codes are the codes assigned to individual characters that allow computers to process text. A character code set is a particular mapping between characters and codes (an organized system of character codes). The character code sets that can be used in the UNIX edition of HiRDB and in the Windows edition of HiRDB are somewhat different. The following table lists the character code sets that can be used on each platform:

| Character code set | UNIX edition | Windows edition |
|---|---|---|
| sjis (Shift JIS Kanji Code) | Y | Y |
| ujis (EUC Japanese Kanji Code) | Y | N |
| chinese (EUC Chinese Kanji Code) | Y | Y |
| utf-8 (Unicode) | Y | Y |
| utf-8_ivs (Unicode (IVS-compatible UTF-8)) | Y | Y |
| chinese-gb18030 (Chinese Kanji Code GB18030) | Y | Y |
| lang-c (Single Byte Character Code | Y | Y |

Legend:

Y: Can be used

N: Cannot be used

## character set

A character set is an attribute of character data, and has the following three attributes:

- Usage format (usage rules for expressing characters)
- Character repertoire (a set of characters that can express a particular character set)
- Predefined comparison sequence (rules for comparing two sets of character string data)

HiRDB can use the following character sets:

- EBCDIK
- UTF16

### check constraint

A constraint used to maintain data integrity in tables by checking specified constraint conditions when data is inserted or updated and suppressing operations on data that does not meet the conditions.

### check pending status

HiRDB limits the data operations that can be performed on referencing tables and check constraint tables when the integrity of table data can no longer be guaranteed due to execution of SQL code or a utility on a table in which a referential constraint or a check constraint is defined. This status indicates such a state, in which data operations have been limited because data integrity cannot be guaranteed.

### client for primary facility-providing server

The HiRDB client that accesses the primary facility-providing server is called the client for primary facility-providing server. The client for primary facility-providing server can access only disk databases. It can execute definition SQLs.

### closing RDAREAs

Process of prohibiting accesses from HiRDB to RDAREAs (closing the RDAREAs). To execute a utility that re-creates RDAREAs, the RDAREAs must first be closed.

When an RDAREA is closed, some RDAREA definition information retained in memory by HiRDB and data storage information are discarded from memory and must be re-created the next time the RDAREA is opened.

For example, when `pdmod` is to be used to re-initialize or delete RDAREAs or `pdrstr` is to be used to recover a database, the RDAREAs must first be closed.

To close an RDAREA whose open attribute is not `SCHEDULE`, the RDAREA must be shut down before it can be closed.

### cluster key

A cluster key is a table column that is specified as the key for storing rows in ascending or descending order of the values in the specified column.

### comment

Comments include simple comments and enclosed comments.

A simple comment in an SQL statement starts with `--` and ends with a line break character.

An enclosed comment in an SQL statement is specified by enclosing it between `/*` and `*/`.

### communication trace

A communication trace is a type of troubleshooting information used for investigating errors, and it records event information on facilities or the like when HiRDB processes carry out communication. Many HiRDB processes record information on the facilities used for communication in the process private memory, to be used in troubleshooting. If a process terminates abnormally, the content of the process private memory is output to the core file. Therefore, you can acquire communication traces from the core file.

### compatibility mode

One of the operand default behaviors. In this mode, when any of the following specifications are omitted, the default value for a particular version is assumed.

- HiRDB system definition operands
- Utility options
- SQL operands
- Environment variables in the client environment definition
- Preprocessing options

Note that the mode that assumes the default values of version 09-04 is called the *0904 compatibility mode*.

### compiling

Compiling is the process of converting a post-source program created by an SQL preprocessor in order to create a machine language program (object code).

### compressed column

A column for which the specification to compress data when storing it in a table (`COMPRESSED` option) is defined.

### compressed table

A table in which compressed columns are defined.

### concurrent connections, maximum number of

The number of connections that can be established at the same time to a HiRDB server (single front-end server in a HiRDB parallel server configuration system). This value is specified in the `pd_max_users` operand. Connection requests that exceed the maximum number of concurrent connections result in a CONNECT error.

### connection frame guarantee facility for client groups

Facility that groups clients connected to HiRDB and guarantees the maximum number of HiRDB connections allowed for each client group.

### connection security facility

Facility for enhancing password security. This facility enables the user to set a minimum size for passwords (in bytes) in order to prohibit use of simple or easily-guessed passwords. It can also set a maximum number of consecutive unsuccessful attempts to enter a password (consecutive certification failures).

### consecutive certification failure account lock state

If user authentication fails because of too many consecutive failures to enter the correct password, that user is placed in consecutive certification failure account lock state. A user in this status can no longer establish connection with HiRDB (`CONNECT`).

### consistency group

A group that guarantees maintenance of the update order integrity of S-VOLs. Integrity is maintained between the order in which data is written on the P-VOL and the order in which data is updated to the S-VOL on all asynchronous pair volumes that make up the consistency group.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### constructor function

The constructor function is used to generate an abstract data type value.

### critical-state process

A critical-state process is a process for which the critical state, *m*, is `Y` in the execution result of the command `pdls -d rpc`.

### cursor

A cursor is used during data retrieval or updating to maintain the most recent extraction position, in order to extract multiple rows of a retrieval result one row at a time. `DECLARE CURSOR` is used to declare a cursor, the `OPEN` statement is used to open a cursor, the `FETCH` statement is used to retrieve search results and advance the cursor to the next row, and the `CLOSE` statement is used to close a cursor.

### database

A database is a system that organizes systematically tables, indexes, user-defined types, stored procedures, stored functions, and access privileges that are created for processing jobs.

### data definition language

A language for defining the structure and contents of a database.

### data dictionary

A data dictionary stores database design specifications, including database table structures, column definitions, and index definitions. In the case of a HiRDB single server configuration, the single server provides centralized management of the data dictionary. With a HiRDB parallel server configuration, centralized management of the data dictionary is provided by the dictionary server.

### data dictionary LOB RDAREA

A data dictionary LOB RDAREA stores either stored procedures or stored functions. The two data dictionary LOB RDAREAs are one for storing the definition sources and one for storing the objects of stored procedures or stored functions.

### data directory RDAREA

A data directory RDAREA stores dictionary tables for managing the analysis results of a definition SQL and the dictionary table indexes.

### data loading

Data loading is the process of storing data in a table. Data loading is performed by the database load utility (`pdload`).

### data loading with synchronization points set

Normally, while data is being loaded, transactions cannot be reconciled until the storage processing of all the data has been completed. This means that synchronization point dumps cannot be obtained during execution of the database load utility. If HiRDB terminates abnormally during loading of a large quantity of data, it will take a long time to restart HiRDB. To resolve this problem, you can set synchronization points for any number of data items during data loading in order to reconcile transactions. This is called data loading with synchronization points set.

To perform data loading with synchronization points set, you must specify a synchronization point lines count, which is the number data items to be stored before a synchronization point is set. This value is specified in the `option` statement of the database load utility. When a large amount of data is to be loaded into a table, consider whether to execute data loading with synchronization points set.

Synchronization points can also be specified in the database reorganization utility; this is called reorganization with synchronization points set.

### data manipulation language

A language for specifying database operations so that a business application program can perform operations on the data in a database.

### data warehouse

A data warehouse is the database system concept that enables databases residing at a mainframe, UNIX servers, and PC servers to be linked, so that they can be accessed in a manner transparent to the various different operating systems. With a data warehouse, end-users can access a database from PCs and the system administrator can access it by means of a system analysis tool, so that the data warehouse provides to each the necessary database management functions. The replication functions (HiRDB/Datareplicator and HiRDB Dataextractor) that can be used in data warehouse databases are available in HiRDB.

### deadlock

Deadlock occurs when multiple transactions are competing for the same resources and are waiting for each other to release their resources.

### default constructor function

The default constructor function is created automatically by HiRDB and identified with the same name as an abstract data type; the default constructor function does not take any arguments.

### deferred write processing

Deferred write processing writes pages that have been updated in the global buffer onto the disk when the number of updated pages reaches a certain value, rather than when a `COMMIT` statement is issued. The point at which the number of updated pages reaches the certain value (as determined by HiRDB) is called the deferred write trigger.

### delayed batch creation of plug-in index

Instead of adding plug-in index data when row data is added to a table, the plug-in index delayed batch creation facility can be used to add the plug-in index data in a batch later with the database reorganization utility.

This facility is used for adding (or updating) a large volume of row data to a table for which a plug-in index is defined.

### delayed rerun

Delayed rerun is the mode that enables rollback and acceptance of new transactions to be started simultaneously during system recovery.

### dictionary server

The server that centrally manages data dictionaries that contain database definition information is called the dictionary server.

### differential backup facility

The differential backup facility backs up only the information that has changed since the previous backup was made, which reduces the amount of time required to make a backup. Consider using the differential backup facility when the database is large and the amount of data updating has been relatively small.

### differential backup management file

A file required for using the differential backup facility. A differential backup management file stores information obtained when differential backups are made. HiRDB uses this file when backups are made, and during recovery of a database using backups.

### Directory Server linkage facility

A facility (such as Sun Java System Directory Server) that is used to manage and authenticate HiRDB users. This capability is provided by a process called the Directory Server linkage facility. By using a directory server, you can centrally manage organizational and user information (user IDs, passwords, affiliations, and job titles) that is otherwise managed separately in HiRDB, Groupmax, and other systems.

### dynamic SQL

A method of generating SQL while a UAP is executing. Contrast with *static SQL*, which is a method of coding SQL within a program when a UAP is created.

### dynamic update of global buffer

Dynamic update of global buffer refers to the ability to add, change, and delete global buffers with the `pdbufmod` command while HiRDB is running.

### embedded UAP

An embedded UAP is a UAP in which an SQL is written directly into a source program that is written in a high-level language (C or COBOL).

### encapsulation

The process of separating an abstract data type interface from its implementation method and keeping this implementation method hidden from the user is called encapsulation.

### exceptional key value

An exceptional key value is a key value that is to be excluded in order not to create a useless index key during index definition. If an index has the null value in all rows of the column or has key values that are duplicated multiple times, the null value can be specified as the exceptional key value.

### excised column

A column that is excised from a reserved column and added to a table. An excised column is specified in the column addition definition of the `ALTER TABLE` statement.

### facility for conversion to a decimal signed normalized number

The facility for conversion to a decimal signed normalized number converts the sign part of signed packed-format data (decimal, date interval, time interval) that has been input; the conversion is performed in accordance with the rules described in Section *7.7 Facility for conversion to a decimal signed normalized number*. The process of converting a sign is called *normalization of the sign part*.

### facility for monitoring abnormal process terminations

A facility that monitors the number of times a server process terminates abnormally over a given period of time and terminates HiRDB (or the unit in the case of a HiRDB parallel server configuration) abnormally when the number of server abnormal terminations exceeds the value specified in the `pd_down_watch_proc` operand. We recommend that you use this facility in conjunction with the system switchover facility. Because this facility terminates HiRDB abnormally when a specified number of server process abnormal terminations occurs, the system switchover facility will provide rapid system switchover. If you do not use this facility, the system will not be switched over in such a situation because HiRDB will not terminate abnormally.

### facility for parallel writes in deferred write processing

A facility for performing deferred write processing by executing multiple processes in parallel, which reduces the amount of time required for write processing.

### facility for predicting reorganization time

Typically, the user must determine the necessity for and timing of table or index reorganization or RDAREA expansion on the basis of the results of executing the `pddbst` command. This sometimes results in tables being reorganized unnecessarily or in tables not being reorganized when needed because of human error (such as failure to read messages that have been output).

To simplify these operations, the facility for predicting reorganization time enables HiRDB to predict when reorganization will be performed.

### facility for simple authentication of OS login users

The ability to connect to the HiRDB server by simple authentication of users logged in to the OS on which the HiRDB client is running.

### facility using arrays

Array-type variables enable multiple processes to be executed by a single execution of an SQL statement. Use of this facility reduces the number of communications between HiRDB client and server.

The facility using arrays can be applied to the `FETCH`, `INSERT`, `UPDATE`, and `DELETE` statements.

### falsification prevention facility

Facility that prevents updating of table data other than the data in updatable columns in order to protect table data from errors or unauthorized tampering. Tables on which the falsification prevention facility is applied are called falsification-prevented tables. When the falsification prevention option is specified, the only operations permitted on the table are row addition, retrieval, deletion of data whose deletion-prevented duration has expired, and updating of updatable columns. A table definition can also be changed to convert an existing falsification-unprevented table into a falsification-prevented table. If no deletion-prevented duration is specified, the table and its data cannot ever be deleted.

### fence level

A pair volume copying method or a level for guaranteeing the integrity of data on pair volumes when an update-copy error occurs. The fence levels are `data`, `never`, and `async`. For details about the levels, see the RAID Manager documentation.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### FIX attribute

`FIX` is the attribute assigned to a table whose row length is fixed.

To realize improvements in access performance, specify the `FIX` attribute for a table to which no columns will be added, that has no columns with the null value, or that has no variable-length columns.

### FIX hash partitioning

FIX hash partitioning is a method of row-partitioning a table using a hash function that divides the table by row and stores the values of the columns that comprise the table evenly among RDAREAs. The column specified as the condition for row partitioning of the table is called the partitioning key. When FIX hash partitioning is used, HiRDB recognizes the portion of the table that is partitioned into each RDAREA. Consequently, during retrieval processing, only those back-end servers in which the applicable data is presumed to be located must become retrieval targets.

### flexible hash partitioning

Flexible hash partitioning is a method of row-partitioning a table using a hash function and dividing the values of the columns that comprise the table evenly among RDAREAs. The column specified as the condition for row partitioning of the table is called the partitioning key. When flexible hash partitioning is used for partitioning and storage of a table in RDAREAs, it is not possible to tell which portion of the table is in which RDAREA. Consequently, during retrieval processing, all back-end servers in which the applicable table is located must become retrieval targets.

### floatable server

A floatable server is a back-end server that is dedicated to sort and merge operations that involve a heavy processing load; it is not used to access table data. Use of floatable servers improves the overall performance of HiRDB. Table data is not stored in a back-end server that is assigned as a floatable server. A back-end server that does not perform data accesses due to the nature of a particular transaction can be used temporarily as a floatable server. A floatable server can be set up only on a HiRDB parallel server configuration.

### foreign key

One or more columns defined for a referencing table. The value of a foreign key is restricted in such a manner that it takes either the same value as the primary key that references it or a value that includes the null value.

### free page

A page in which no data is stored. A free page might be either a used free page or an unused free page.

### free page reuse mode

A method of searching for free space in an RDAREA when HiRDB is saving data. When a segment becomes full, a search is conducted to find free space in used pages before an unused segment is allocated. The next search start position is also remembered, so that, the next time, searching for free space begins at this position.

### free segment

A segment in which no data is stored. A free segment might be either a used free segment or an unused free segment.

### free space reusage facility

A facility by which free space in used segments is utilized once the number of segments specified by the user is reached and the last segment becomes full. This is done by switching the page search mode to the free page reuse mode. Once the free space in all the segments of the specified number of segments runs out, the free space reuse facility activates the new page allocate mode, and a new unused segment is allocated.

### front-end server

A front-end server is a constituent element of a HiRDB parallel server configuration that determines the method of database access to be used from an executed SQL and provides execution instructions to back-end servers.

### global buffer

A buffer that is used for input and output of data stored in RDAREAs on disk. Global buffers are allocated in common memory. Global buffers are always allocated to an RDAREA or an index.

A LOB global buffer is allocated in the following cases:

- When a plug-in index is being stored
- When benefits can be expected from buffering because there is a relatively small amount of data

- When LOB data that will be accessed frequently is being stored

### global buffer pre-writing

A facility that writes specified tables and indexes into global buffers in advance. This improves the buffer hit rate.

### global deadlock

A form of deadlock that occurs between servers in a HiRDB parallel server configuration.

### guest area

Resources for a back-end server that is associated with a guest BES.

### guest BES

A back-end server whose processing has been moved to another unit by the standby-less system switchover (effects distributed) facility in the aftermath of an error. The unit for a guest BES is called the accepting unit.

### HA Monitor

A cluster software program for implementing a system switchover facility. It is packaged with the Hitachi HA Toolkit Extension.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### hash facility for hash row partitioning

The addition of RDAREAs to accommodate an increase in the amount of data in a hash-partitioned table (increasing the number of row partitions in a table) can create an imbalance in the amount of data in the existing RDAREAs and in the new RDAREAs. The hash facility for hash row partitioning can correct any such imbalance resulting from an increase in the number of table row partitions.

The hash facility for hash row partitioning can be applied to both FIX hashing and flexible hashing.

### HiRDB administrator

Of the users who log onto the operating system using a system administrator user ID, the HiRDB administrator is the user who has the privilege to manipulate HiRDB itself. The HiRDB administrator's privilege permits execution of HiRDB commands and conveys ownership of HiRDB directories and files.

### HiRDB client

A workstation or personal computer on which HiRDB/Developer's Kit or HiRDB/Run Time has been installed.

### HiRDB file

A file that is used by HiRDB. A HiRDB file is composed of contiguous segments in a HiRDB file system area. It stores tables, indexes, and information necessary to recover the status of the system in the event of an error.

### HiRDB file system area

An area in which HiRDB files are created. Separate HiRDB file system areas are created for different application purposes, for system files, for work table files, and for RDAREA files.

### HiRDB.ini file

A file that contains information required by utilities that execute on HiRDB clients and for the HiRDB SQL Executer to connect to a HiRDB server. This file must be available on both the HiRDB server PC and the HiRDB client PC.

### HiRDB.NET data provider

Data provider required in order to access databases from applications compatible with ADO.NET. The HiRDB.NET data provider complies with the ADO.NET specifications.

### HiRDB system definition files

The HiRDB system definition files store the HiRDB system definition information that is used to determine the HiRDB operation environment (system configuration, control information, and each server's execution environment).

### host BES

A back-end server that is defined on an applicable unit by the standby-less system switchover (effects distributed) facility. The unit for a host BES is called a regular unit.

### hybrid method

One of the processing methods used in Real Time SAN Replication. When an update of a system file at the main site occurs, copying to the corresponding file at the remote site is performed synchronously. When an update of a database file at the main site occurs, copying to the corresponding file at the remote site is performed asynchronously.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### incorporation during commit

Incorporation during commit is the process of writing pages that have been updated in the global buffer onto the disk when a `COMMIT` statement is issued.

### index

An index is assigned to a column as a key for retrieving the table. An index consists of a key and key values. The key is the column name that describes the contents of the column, and the key values are the actual values in the column.

An index can be either a single-column index or a multicolumn index. A single-column index is based on a single column of a table; a multicolumn index is based on more than one column of a table.

### index page split

HiRDB indexes have a B-tree structure. Consequently, an index page split occurs when there is no free area available in an index page to which a key is to be added. An index page split occurs when HiRDB splits the index information of an index page in order to allocate a free area and moves the second half of this information to a new page.

### index reorganization

Rearrangement of an index based on information derived by retrieving index key information to create an index information file. Index reorganization can be performed by index or by index RDAREA.

Over time, deleting (`DELETE`) and updating (`UPDATE`) data degrades an index's storage efficiency, which reduces performance when the index is used for searches. Index reorganization is performed to minimize such performance degradation.

### inheritance

Inheritance is the process by which a lower-order abstract data type inherits attributes and routines defined in a higher-order abstract data type.

### inheriting a database

The operations after a site switchover has occurred that involve shutting down HiRDB after the log application processing has been completed and the site status has been changed. Inheriting a database is performed with the `pdrisedbto` command.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### in-memory copying

In in-memory data processing, loading all data in an RDAREA into memory and keeping this data resident is called in-memory copying.

### in-memory data buffer

A data buffer used during in-memory data processing.

### in-memory data processing

A processing method that keeps all data in an RDAREA resident in memory (in-memory data buffer) to reduce the number of disk input/output operations.

### in-memory RDAREA

An RDAREA targeted for in-memory data processing. All data in this RDAREA is kept resident in memory.

### inner replica facility

A facility by which duplicate RDAREAs (replica RDAREAs) are defined and manipulated. Using a disk system or software that provides mirroring, the inner replica facility allows you to access the duplicated database. For details about the inner replica facility, see the *HiRDB Staticizer Option Version 9 Description and User's Guide*.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### integrity constraint

An integrity constraint is a constraint that is available for guaranteeing the validity of data in a database.

There are two integrity constraints:

- `NOT NULL` constraint (does not permit the null value to be set in a specified column)
- Uniqueness constraint (requires that the values in a specified column be unique and does not permit value duplication in the column)

### interface area

An area used for exchanging information between HiRDB and UAPs. HiRDB provides the following seven interface areas:

- SQL Communications Area
- SQL descriptor area
- Column Name Descriptor Area (SQLCNDA)
- Type Name Descriptor Area (SQLTNDA)
- Embedded variables
- Indicator variables
- Parameters

### inter-process memory communication facility

When a HiRDB server and a HiRDB client are running on the same server machine, memory is used for inter-process communication to increase communication speed between the HiRDB server and the HiRDB client. This functionality is provided by the *inter-process memory communication facility*, which is used by specifying `MEMORY` in the `PDIPC` operand of the client environment definition.

### IP address

An address used by the IP protocol.

### IVS

IVS (Ideographic Variation Sequence) is a Unicode standard for distinguishing stylistic variants of the same character (glyph variants). In IVS, glyph variants are represented by a character combination consisting of the underlying (base) character followed by a code (the Variation Selector (VS)) that represents the glyph variant. Note that separate IVS-compatible fonts are required to actually display the glyph variants.

### Java stored function

A Java stored function is a stored function in which procedures are coded in Java.

### Java stored procedure

A Java stored procedure is a stored procedure in which procedures are coded in Java.

### JDBC driver

A driver created for HiRDB based on the interface defined in the JDBC standard. The JDBC driver is needed to execute Java stored procedures and Java stored functions. It is installed by selecting it when you install a HiRDB client.

### join

Processing that combines two or more tables.

### JP1

JP1 is a series of products that provide functions for batch job operations, automatic system operation, document output control, and file backup.

JP1 is used to automate system operations and reduce labor costs.

### key range partitioning

Key range partitioning is a method of table partitioning that divides a table into groups of rows, using ranges of values in a column of the table as the partitioning condition. The column specified as the condition for row partitioning is called the partitioning key. Key range partitioning can be implemented with a storage condition specified or a boundary value specified.

### LAN adapter

Hardware that converts data for connecting a computer to a LAN.

### language compiler

A program that converts a post-source program created by an SQL preprocessor into a machine-language program (object).

### large file

A HiRDB file system area of at least 2 gigabytes in size is called a large file. Normally, the maximum size of a HiRDB file system area is 2,047 megabytes (approximately 2 gigabytes). A HiRDB file system area whose size needs to exceed this limit must be created as a large-file HiRDB file system area. The maximum size of a HiRDB file system area created as a large file is 1,048,575 megabytes.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### linkage

The process of creating a single executable program file (load module) by combining the object modules created by a language compiler.

### linker

A program that combines the object modules created by a language compiler into a single executable program file (load module).

### list

A set of data items saved under a single name (list name). Lists include sets of data items that might be saved with temporary names at intermediate stages during the process of filtering information in a series of hierarchical steps based on specification of conditions that are designed to retrieve an appropriate number of data items.

### list RDAREA

Lists created by the `ASSIGN LIST` statement are stored in a list RDAREA. A list RDAREA is created when a narrowed search is performed.

### load module

A UAP source that has been preprocessed, compiled, and linked into a single executable file.

### LOB column structure base table

A table containing LOB columns from which the LOB data has been removed is called a LOB column structure base table. Data can be loaded and the database can be reorganized by LOB column structure base table or LOB column.

### LOB data

Large variable object data such as documents, images, and sounds is called LOB data. LOB data is stored in user LOB RDAREAs. Data loading and database reorganization can be performed separately from the user RDAREA that stores the LOB column structure base table.

### local buffer

A buffer that is used for input and output of data stored in RDAREAs on disk. Local buffers are allocated in process private memory.

### locator facility

Facility that reduces a client's memory resource overload and data transfer volume during `BLOB` or `BINARY` data searches.

A locator is a 4-byte data value that is used to identify data at a server. When a locator-embedded variable is specified, such as in the `INTO` clause of a single-row `SELECT` statement or `FETCH` statement, the value of the locator indicating the data, not the data itself, is received as the search result. Processing that handles the data identified by a locator can be performed when a locator-embedded variable is specified in other SQL statements.

### lock

A form of control managed by HiRDB to maintain database integrity.

### locking

Locking is a type of control managed by HiRDB in order to maintain database integrity.

### log acquisition mode

The log acquisition mode is one of the modes for acquiring a database update log during execution of a UAP or utility. This mode acquires the database update log that will be needed for rollback or rollforward when the UAP or utility is updating the contents of an RDAREA.

### log application

In the log-only synchronous method, update processing of a database based on system logs copied from the transaction execution site. Log application is performed at the log application site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### log application not possible status

The state in which integrity of the databases at the transaction execution site and the log application site is not established or the information required for log application cannot be copied correctly. If a natural disaster occurs while the system is in this status, data will be lost if the transaction execution site goes down.

When the preparations for system log application are performed while the system is in log application not possible status, the system is placed in log application possible status.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### log application possible status

The state in which integrity of the databases at the transaction execution site and the log application site is established and the information required for log application can be copied correctly. If a natural disaster occurs while the system is in this status, job processing can be transferred to the log application site without loss of data, even if the transaction execution site goes down.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### log application site

In the log-only synchronous method, the site at which update processing of databases is performed based on system logs copied from the transaction execution site. As with the transaction execution site, the log application site must be operational at all times.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### loopback address

A loopback address is an IP address between 127.0.0.0 and 127.255.255.255 (for example, 127.0.0.1). The IP addresses that can be used as loopback addresses depends on the OS specification. Furthermore, because HiRDB handles `localhost` as a regular host name, you must resolve the naming conflict in advance if you plan to specify `localhost` as the host name in the system definition.

### LVM

A disk device management facility that can create and manage multiple virtual devices from a set of one or more real devices. LVM enables the user to achieve disk device management with high availability and management simplicity, such as by dividing a single real device into multiple small virtual devices or by grouping multiple real devices into a single large virtual device. LVM also provides functions for improving performance and achieving redundancy.

### main site

Site where the system normally used by Real Time SAN Replication is located.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### master directory RDAREA

The master directory RDAREA stores information on the RDAREAs that store dictionary tables and user-created tables and indexes; it also stores RDAREA registration location (server) information.

### matrix partitioning

A method of partitioning a table by a combination of partitioning methods using two of the table columns as the partitioning key. The first column used as the partitioning key is called the*first dimension partitioning column*, and the second column used as the partitioning key is called the *second dimension partitioning column*. Matrix partitioning always involves key range partitioning with boundary values specified for the first dimension partitioning column and then partitioning the resulting data further by the second dimension partitioning column. A table partitioned by the matrix partitioning method is called a matrix partitioning table.

### message queue monitoring facility

A facility that monitors for the server process not responding state. HiRDB uses the message queue during server process allocation processing. If a server process not responding state occurs, messages can no longer be fetched from the message queue. If messages cannot be fetched from the message queue after a certain amount of time has elapsed (called the message queue monitoring time), HiRDB outputs a warning message or error message (KFPS00888-W or KFPS00889-E). Output of one of these messages indicates the possibility that a server process not responding state might have occurred.

### module trace

A type of troubleshooting information used for isolating errors. The executed functions and macro history of most HiRDB processes are recorded in a process-specific memory. If a process terminates abnormally and a core file is output, the contents of the process-specific memory are output to the core file, which allows a module trace to be obtained from the core file.

### monitor mode

The system switchover facility can be operated in either the monitor mode or the server mode. In the monitor mode, only system failures are monitored. In the server mode, both system failures and server failures (such as HiRDB abnormal termination) are monitored. System switchover can take less time in the server mode than in the monitor mode because a standby HiRDB is started beforehand in the server mode. The system switchover facility must also be operating in the server mode to use the following functions:

- user server hot standby
- rapid system switchover facility
- standby-less system switchover facility

### monitoring the free space remaining for system log files, facility for

A facility that outputs a warning message or suspends scheduling of new transactions and forcibly terminates all transactions on the server when the percentage of free space for the system log files drops below a warning value.

### multi-connection facility

The multi-connection facility enables a UAP running on a HiRDB client to connect simultaneously to multiple HiRDB servers.

### multi-HiRDB

Multi-HiRDB is the mode in which multiple HiRDB systems run on a single server machine.

### multiple front-end servers

The configuration in which multiple front-end servers are set up is used when the CPU load for SQL processing is too heavy for a single front-end server.

### narrowed search

A narrowed search is the process of narrowing in steps the records retrieved in successive searches.

A narrowed search requires creation of a list using the `ASSIGN LIST` statement of the data manipulation SQL. When such a list is created under appropriate conditions, its use can reduce processing time. If there are multiple conditions, multiple lists can be combined for the search.

### NetBackup linkage facility

A facility designed to create backup files for use by the database copy utility (`pdcopy`) or database recovery utility (`pdrstr`) on a medium managed by NetBackup Server.

### new page allocate mode

A method of searching for free space in an RDAREA when HiRDB is saving data. When a segment becomes full, first, a new unused segment is allocated. If the RDAREA runs out of unused pages, a search is then conducted to find free space in a segment already allocated to that table (in a used segment), starting at the beginning of the first used segment. Once free space is found, the data is saved to that free space.

### no-log mode

The no-log mode is one of the modes for acquiring a database update log during execution of a UAP or utility. This mode does not acquire a database update log when the UAP or utility is updating the contents of an RDAREA.

### normal BES

A back-end server whose processing is transferred to another server in the event of an error, through use of the standby-less system switchover (1:1) facility. Similarly, a unit containing a normal BES is called a normal BES unit.

### no-split option

Normally, the data part of a variable-size character string that is at least 256 bytes in length is stored in different pages. The no-split option allows you to store one row of data in one page even when the actual data size of a variable-size character string is 256 bytes or greater. The no-split option improves the data storage efficiency. The no-split option is used by specifying `NO SPLIT` in the `CREATE TABLE` or `CREATE TYPE` statement.

### NOT NULL constraint

The `NOT NULL` constraint is a limitation that does not permit the null value to be set in a column.

### no transaction loss (no data loss)

With no transaction loss, update processing on transactions committed at the main site is guaranteed to be mirrored to the database at the remote site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### null value

The null value is a special value that indicates that no value has been set.

### object

A program that has been translated into machine language by a language compiler.

### object relational database

An object relational database is a database that has been expanded by incorporating the concept of object orientation into a relational database model. HiRDB can handle data that has a complex structure, such as multimedia data, and the operations for such data as a unified object and can manage it in a database.

### operation releasing check of unload status

A system log file can become a swap target only if it is in one of the following statuses:

- Overwriting enabled status
- Extraction completed status (HiRDB Datareplicator)
- Overwriting permitted status for online reorganization (HiRDB Staticizer Option) (UNIX edition only)

The unloading status is unrelated to the statuses required in order for a system log file to be a swap target. Performing this operation eliminates the need to unload or release system log files.

### operation unloading system log

The operation of unloading system logs. When a database is recovered, unload logs that have been unloaded from the system logs are used as the input to the database recovery utility.

### operation without unloading system log

Operating mode in which system log files are not unloaded. When a database is recovered, system logs are entered directly as input information to the database recovery utility, without unload logs being used.

### optimizing based on cost

When multiple indexes have been created for a table, HiRDB selects and uses the index that has the least access cost and that it evaluates to be optimal based on the conditions specified for the table retrieval. This process in which HiRDB selects the index it evaluates to be optimal is called *optimizing based on cost*.

### overload

Multiple stored functions can be defined with the same name as long as their parameter counts and data types are different. Stored functions that have the same name are said to be *mutually overloaded*.

### override

Override is the process of overwriting the definition of a lower-order abstract data type with a routine that has the same name as the routine defined for the higher-order abstract data type.

### page

The page is one of the data storage units; it is the smallest unit for database input/output operations. There are three types of pages:

- Data page: Page for storing rows of a table
- Index page: Page for storing index key values
- Directory page: Page for storing RDAREA status management information

### pair logical volume group

A group of pair logical volumes. Pair logical volumes on which files have been distributed are handled in units of pair logical volume groups.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### pair logical volumes

Volumes paired between servers that have been logically named and configured.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### pair status

A designation that indicates the status of a pair volume. The pair statuses include `PAIR`, `SMPL`, and `PSUS`. For details about these statuses, see the RAID Manager documentation.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### pair volumes

With Hitachi Disk Array System volumes, a volume at the main site and a volume at the remote site that correspond to each other. Pair volumes consist of the primary volume and a secondary volume.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### paired

A status in which a pair logical volume or pair logical volume group is paired.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### partitioning key index

An index that satisfies a particular condition becomes a partitioning key index. An index that does not satisfy the condition is called a non-partitioning key index. The condition is explained as follows.

The condition is whether the table involved is partitioned on the basis of single-column partitioning or multicolumn partitioning. When only one column is used in the table partitioning condition, the partitioning is said to be *single-column partitioning*; when multiple columns are used in the table partitioning condition, the partitioning is said to be *multicolumn partitioning*.

## E.1 Single-column partitioning

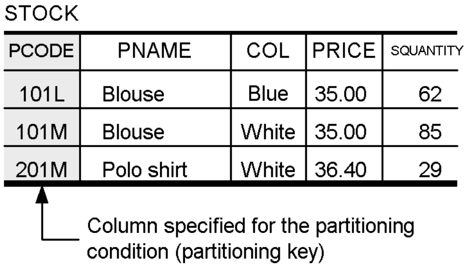
An index satisfying one of the following conditions is a partitioning key index:

**Conditions**

- Single-column index is defined for the column (partitioning key) on the basis of which storage conditions were specified when the table was row partitioned
- Multicolumn index is defined, and the column (partitioning key) on the basis of which storage conditions were specified when the table was row partitioned is the first constituent column

The following figure shows an example (based on the STOCK table) of an index that becomes a partitioning key index.

Figure E–1: Partitioning key index: Single-column partitioning



STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |

Column specified for the partitioning condition (partitioning key)

**Explanation**

```
CREATE INDEX A12 ON STOCK (PCODE ASC)                   1
CREATE INDEX A12 ON STOCK (PCODE ASC, PRICE DESC)       2
CREATE INDEX A12 ON STOCK (PRICE_DESC, PCODE ASC)       3
```

1. If the PCODE column, which is a partitioning key, is used as an index, the index becomes a *partitioning key index*. If any other column is used as an index, the resulting index becomes a non-partitioning key index.
2. Specifying the PCODE column, which is the partitioning key, as the first constituent column of the index makes the resulting multicolumn index a *partitioning key index*.
3. Specifying the PCODE column, which is the partitioning key, as a column other than the first constituent column of the index makes the resulting multicolumn index a *non-partitioning key index*.

## E.2 Multicolumn partitioning

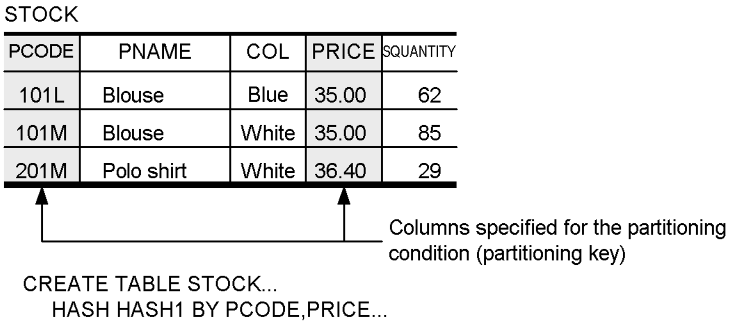
An index satisfying the following condition is a partitioning key index:

**Condition**

- Index is created on the basis of multiple columns, beginning with the partitioning key and containing all the columns specified for partitioning from the beginning and without any change in their order.

The following figure shows an example (based on the STOCK table) of an index that becomes a partitioning key index.

Figure E–2: Partitioning key index: Multicolumn partitioning



STOCK

| PCODE | PNAME | COL | PRICE | SQUANTITY |
|---|---|---|---|---|
| 101L | Blouse | Blue | 35.00 | 62 |
| 101M | Blouse | White | 35.00 | 85 |
| 201M | Polo shirt | White | 36.40 | 29 |

Columns specified for the partitioning condition (partitioning key)

```
CREATE TABLE STOCK...
  HASH HASH1 BY PCODE,PRICE...
```

**Explanation**

```
CREATE INDEX A12 ON STOCK (PCODE ASC, PRICE DESC)       1
CREATE INDEX A12 ON STOCK
            (PCODE ASC, PRICE DESC, SQUANTITY ASC)      2
CREATE INDEX A12 ON STOCK (PRICE DESC, PCODE ASC)       3
CREATE INDEX A12 ON STOCK
            (PCODE ASC, SQUANTITY DESC,PRICE DESC)      4
```

1. All partitioning keys (PCODE and PRICE columns) are specified, and these keys are specified in the same order as in the table definition. Therefore, this multicolumn index is a *partitioning key index*.
2. All partitioning keys (PCODE and PRICE columns) are specified, and these keys are specified in the same order as in the table definition. Therefore, this multicolumn index is a *partitioning key index*.
3. All partitioning keys (PCODE and PRICE columns) are specified, but these keys are specified in an order that differs from the table definition. Therefore, this multicolumn index is a *non-partitioning key index*.
4. All partitioning keys (PCODE and PRICE columns) are specified, but these keys are specified in an order that differs from the table definition. Therefore, this multicolumn index is a *non-partitioning key index*.

## password-invalid account lock state

When limitations are set on password character strings by the connection security facility, a user violating any of the limitations is placed in password-invalid account lock state. A user in this status can no longer establish connection with HiRDB (CONNECT).

## permanent base table

A base table that is not a temporary table. The term *table*, as used in Hitachi manuals, refers to a permanent base table unless otherwise specified.

## plug-in

A plug-in is a HiRDB package product that provides *abstract data types* in which documents, spatial data, and other multimedia entities are defined, and a facility for rapid manipulation of complex data.

## plug-in index

A plug-in index is a type of index provided by a plug-in.

## POSIX library edition

A HiRDB execution environment required for using a Java stored procedure or Java stored function. To use the POSIX library edition, you specify the -l option of the pdsetup command.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

## post-source

The source program generated by preprocessing embedded SQL statements.

## predefined data type

A data type provided by the HiRDB system is called a predefined data type. The predefined data types include INTEGER, CHARACTER, DATE, etc.

### prefetch facility

A facility that reads multiple pages at a time from a global buffer or a local buffer. It can reduce the I/O time when searches on large volumes of data are performed in HiRDB file system areas that use the raw I/O facility (or, in the UNIX edition, when searches are performed in character special files). It is particularly effective for searches that do not use an index, or for ascending-order searches on large data sets in tables that do use an index.

### preparations for system log application

A procedure that enables performance of log application in which database integrity is established and information in files required to perform log application is set in the proper state by temporarily synchronizing the databases at the transaction execution site and the log application site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### preprocessing

Processing performed on code before it is compiled by a language compiler. Preprocessing converts SQL statements to high-level language code.

### pre-update log acquisition mode

Pre-update log acquisition mode is one of the modes for collecting a database update log when a UAP or utility executes. This mode acquires only the database update log needed for rollback when a UAP or utility is updating the contents of an RDAREA.

### PRF trace

Trace information for troubleshooting that is output for a series of HiRDB operations. A PRF trace lets you trace the flow of processing across multiple machines and multiple processes for the purpose of performance verification and more effective troubleshooting.

PRF traces are output to the PRF trace information file. The PRF trace collection level can be selected when a PRF trace is collected.

### PRF trace collection level

The PRF trace collection levels are minimum level (the default), standard level, detail level, maintenance level, and suppression level. By changing the PRF trace collection level, you can adjust the amount of PRF trace information to be output to the PRF trace information file.

The PRF trace collection level is specified in the `pd_prf_level` operand of the system definition. If you want to change the PRF trace collection level while HiRDB is running, use the `pdprflevel` command.

### primary facility-providing server

The primary facility-providing server manages the disk database.

On a HiRDB single server configuration, SDS is called the primary facility-providing server. On a HiRDB parallel server configuration, MGR, FES, DS and BES are collectively referred to as primary facility-providing servers.

### primary key

The primary key enables you to uniquely identify a row in a table. Columns for which the primary key is defined are subject to the *uniqueness constraint* and the *NOT NULL constraint*. The uniqueness constraint is the limitation that does not permit duplicate data in a key (a column or a set of columns); in other words, all data in the key must be unique. The NOT NULL constraint is the limitation that does not permit the null value in the key columns.

### process

An instance of execution of a program. Each process is allocated a virtual space and CPU resources on a time-sharing basis. Executing more than one process in parallel (multi-processing) is one way to increase throughput.

### protection mode

A protection mode can be used to select a processing method when synchronous copying to a remote site cannot be performed. A protection mode can be selected when the all synchronous method or the hybrid method is being used. The protection modes include `data` and `never`, and their processing methods are explained below:

- `data`: Stops update processing at the main site (update processing of volumes containing files that cannot be copied synchronously).
- `never`: Continues update processing at the main site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### public routine

A stored routine whose owner is defined as `PUBLIC`, indicating all users, is called a public routine. Once a stored routine is defined as a public routine, there is no need to specify the owner's authorization identifier for a routine defined by another user to be invoked from a UAP (only the routine identifier is specified). You can define a public routine by using the `CREATE PUBLIC PROCEDURE` or `CREATE PUBLIC FUNCTION` statement.

## RAID Manager instance

A suite of facilities that identify the scope of operations and management that can be performed by RAID Manager. Each instance is identified by an instance number.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

## range specification recovery

One of the methods used by the database recovery utility for recovering a database. It is a technique by which the database can be recovered to a user-specified point between the time a backup was made and the time the failure occurred. The `-T` option of the database recovery utility is used to specify the recovery time.

## rapid grouping facility

When the `GROUP BY` clause of an SQL is used for grouping, sorting is performed before grouping. This is the rapid grouping facility, and it combines grouping with hashing.

## rapid system switchover facility

A facility whereby server processes are started on a standby HiRDB beforehand, so that startup processing of server processes is not performed when the system is switched over. The system switchover time is reduced by the amount of time required to perform startup processing of server processes. In addition to the rapid system switchover facility, user server hot standby is available for reducing system switchover time. The rapid system switchover facility can reduce the system switchover time more than user server hot standby can (the rapid system switchover facility includes the user server hot standby functionality).

## RDAREA

The RDAREA is the basic unit for storing data; an RDAREA can store 1 to 16 HiRDB files. Following are the types of RDAREAs:

- Master directory RDAREA
- Data dictionary RDAREA
- Data directory RDAREA
- Data dictionary LOB RDAREA
- User RDAREA
- User LOB RDAREA
- Registry RDAREA
- Registry LOB RDAREA
- List RDAREA

## RDAREA automatic extension

When RDAREA automatic extension is being used and a space shortage occurs in an RDAREA, segments are added automatically to expand the capacity of the RDAREA, provided that there is adequate unused free space in the HiRDB file system area.

## RDAREA error

An RDAREA error means that an error has occurred in the in-memory RDAREA.

## RDAREAs, shutting down

To limit accesses by UAPs and utilities to an RDAREA, the RDAREA can be shut down (*command shutdown*). If an error (such as an I/O error) occurs on an RDAREA, HiRDB might shut down the RDAREA automatically (*error shutdown*).

There are various shutdown modes, such as one that allows referencing and updating; the user selects the appropriate shutdown mode.

For example, when `pdload` is to be used to load data or `pdrorg` is to be used to re-organize a table, it is best to shut down the applicable RDAREAs first because such processing usually takes a considerable amount of time.

## Real Time SAN Replication

A facility that enables jobs to continue being processed on a standby system set up at a remote site in the event of a disaster (earthquake, fire, etc.) that makes it physically impossible to quickly restore the system that is normally used.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

## recommended mode

One of the operand default behaviors. In this mode, when any of the following specifications are omitted, the recommended value is assumed:

- HiRDB system definition operands
- Utility options

- SQL operands
- Environment variables in the client environment definition
- Preprocessing options

### recovery-unnecessary front-end server

If an error occurs on the unit where the front-end server is located, transactions that were executing from that front-end server might be placed in uncomplete status for transaction determination. Because a transaction in this status has locked the database, some database referencing or updating becomes restricted. Normally, to complete a transaction that is in uncomplete status for transaction determination, the front-end server must be recovered from the error and restarted. However, if the abnormally terminated front-end server is a recovery-unnecessary front-end server, HiRDB automatically completes any transaction that was placed in uncomplete status for transaction determination. This enables other front-end and back-end servers to be used to restart database update processing. A unit containing a recovery-unnecessary front-end server is called a recovery-unnecessary front-end server unit.

### reduced activation

Reduced activation is the process of starting HiRDB using only the normal units when there are units that cannot be started. In such a case, the reduced activation facility enables the HiRDB parallel server configuration to be started using only the remaining units even if some units cannot be activated due to errors.

### reference buffer

When data is referenced, it is referenced in a global buffer. A buffer that is used for referencing data, or a buffer for a database that has been updated, is called a *reference buffer*.

### reference-only back-end server

A back-end server in which shared tables and shared indexes of shared RDAREAs can be referenced but not updated.

### referential constraint

A constraint that is defined for a specific column (foreign key) during table definition in order to maintain referential conformity in data between tables. A table for which referential constraints and a foreign key are defined is called a referencing table, and a table that is referenced from the referencing table using the foreign key is called a referenced table. For the referenced table, the primary key that is referenced by the foreign key must be defined.

### registry facility

The registry facility enables HiRDB to maintain information unique to plug-ins; such information is used by the plug-ins for data manipulation.

### registry LOB RDAREA

The registry LOB RDAREA stores a table for managing registry information (registry table). This RDAREA is required in order to use the registry facility. Not all plug-ins use the registry facility. The system determines automatically whether data is to be stored in the registry LOB RDAREA or the registry RDAREA on the basis of the length of the data. Stored procedures for data manipulation, such as ones that register information in the registry management table, are also stored in this RDAREA.

### registry RDAREA

The registry RDAREA stores a table for managing registry information (registry table). This RDAREA is required in order to use the registry facility. Not all plug-ins use the registry facility.

### regular expression

A method of representation that allows different character strings, such as any character strings, repetitions of the same character string, or any number of characters, to be expressed in a specific sequence (pattern) of characters forming part of a larger character string. Regular expressions are specified in a pattern character string of the `SIMILAR` predicate.

### regular unit

The unit where a host BES is located when the standby-less system switchover (effects distributed) facility is used.

### remote site

Site where the backup system is located for Real Time SAN Replication.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### reorganization by schema

Reorganization by schema is the process of reorganizing in a batch all the tables in a schema. To reorganize in a batch all the tables you own, perform reorganization by schema by specifying the authorization identifier of the schema to be reorganized in the `-t` option of the database reorganization utility. The specification format is `-t` *authorization-identifier*`.all`.

In addition to reorganization by schema, the other units of reorganization are as follows:

- Reorganization by table
- Reorganization by RDAREA

### reorganization with synchronization points set

Normally, while a table is being reorganized, transactions cannot be reconciled until the storage processing of all the data has been completed. This means that synchronization point dumps cannot be obtained during execution of the database reorganization utility. If HiRDB terminates abnormally during reorganization of a large quantity of data, it will take a long time to restart HiRDB. To resolve this problem, you can set synchronization points for any number of data items during storage of the data (reload processing) in order to reconcile transactions. This is called reorganization with synchronization points set.

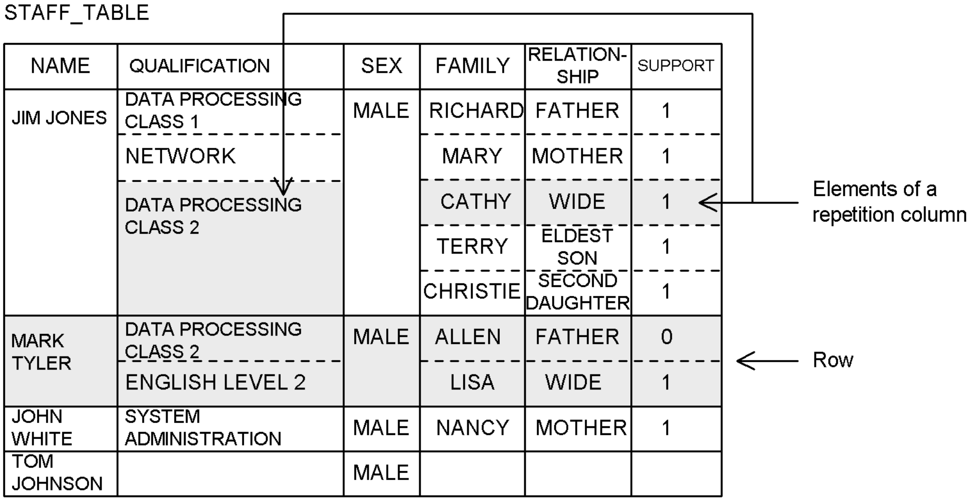
To perform reorganization with synchronization points set, you must specify a synchronization point lines count, which is the number data items to be stored before a synchronization point is set. This value is specified in the `option` statement of the database reorganization utility. When a table containing a large quantity of data is to be reorganized, consider whether to perform reorganization with synchronization points set.

Synchronization point setting can also be specified in the database load utility; this is called data loading with synchronization points set.

### repetition column

A column that consists of multiple elements in a single row is called a repetition column. An element is one of the multiple items in the repetition column row. A repetition column is defined by `CREATE TABLE` with the maximum number of elements specified. However, the number of elements can be increased later with `ALTER TABLE`. The following figure shows an example of a table that contains repetition columns.

Figure E–3: Example of a table containing repetition columns



STAFF_TABLE

| NAME | QUALIFICATION | SEX | FAMILY | RELATION-SHIP | SUPPORT |
|---|---|---|---|---|---|
| JIM JONES | DATA PROCESSING CLASS 1<br>NETWORK<br>DATA PROCESSING CLASS 2 | MALE | RICHARD<br>MARY<br>CATHY<br>TERRY<br>CHRISTIE | FATHER<br>MOTHER<br>WIDE<br>ELDEST SON<br>SECOND DAUGHTER | 1<br>1<br>1<br>1<br>1 |
| MARK TYLER | DATA PROCESSING CLASS 2<br>ENGLISH LEVEL 2 | MALE | ALLEN<br>LISA | FATHER<br>WIDE | 0<br>1 |
| JOHN WHITE | SYSTEM ADMINISTRATION | MALE | NANCY | MOTHER | 1 |
| TOM JOHNSON |  | MALE |  |  |  |

Note: Blank cells contain the null value.

### replication facility

The replication facility enhances the efficiency of data utilization in a data warehouse by linking a database in a Hitachi mainframe with a HiRDB database or a HiRDB database with another HiRDB database.

### reserved column

A reserved column is defined in a FIX table to be used in future column additions. By defining reserved columns, you can add columns while data is stored in the FIX table.

### reserved word

A character string that has been registered as a keyword used in SQL statements.

Reserved words cannot be used as table names or column names. Otherwise, a reserved word can be used as a name if it is enclosed in double quotation marks (").

A keyword that has been registered as a reserved word can be deleted by the SQL reserved word deletion facility; however, some functions might become disabled if a reserved word is deleted.

### rollback

Rollback is the process of invalidating the database processing performed by a transaction when an error has occurred in the transaction.

### row identifier

The information (row address) that the system assigns to uniquely identify a row of data stored in HiRDB.

### row partitioning

Row partitioning is the process of storing a table, index, or LOB column by dividing it into multiple user RDAREAs or user LOB RDAREAs. When a table is row-partitioned, its indexes can also be row partitioned in correspondence with the row-partitioned table. When a table contains a LOB column, it can be partitioned and stored in multiple user LOB RDAREAs in correspondence with the row-partitioned table. When a table is to be row-partitioned, it is necessary to specify storage conditions for the row partitioning; this is done with the `CREATE TABLE` statement of the definition system SQL. When an index is row-partitioned, it is necessary to specify the user RDAREAs in which the row-partitioned index is to be stored; this is done with the `CREATE INDEX` statement of the definition system SQL.

### schema

A concept that encompasses tables, indexes, abstract data types (user-defined types), index types, stored procedures, stored functions, triggers, and access privileges.

### schema definition privilege

The privilege required to define a schema.

### security facility

The security facility prevents anyone who does not have the required authorization from accessing the database.

### segment

The segment is a unit of data storage consisting of multiple contiguous pages. It is the unit of allocation for storage of tables and indexes. A segment can store either one table or one index.

### sequence generator

A sequence generator returns a serial integer (called the *current value*) every time data is called up within a database. For example, the sequence generator can be used to automatically increment the value of a specific column every time data is inserted. When you use a sequence generator, the current value is not changed, even when a rollback occurs. Therefore, you can identify unique data or verify the correct processing sequence regardless of the transaction status.

### server machine

A workstation or personal computer that runs HiRDB server software.

### server mode

The system switchover facility can be operated in either monitor mode or server mode. In monitor mode, only system failures are monitored. In server mode, both system failures and server failures (such as HiRDB abnormal termination) are monitored. System switchover can take less time in server mode than in monitor mode because a standby HiRDB is started beforehand in server mode. In addition, the system switchover facility must be operating in server mode to use the following functions:

- user server hot standby
- rapid system switchover facility
- standby-less system switchover facility

### shared index

An index that is stored in a shared RDAREA and can be referenced from all back-end servers. In order to achieve SQL and UAP compatibility with HiRDB parallel server configurations, shared indexes can also be defined in HiRDB single server configurations. If defined, however, such shared indexes are stored in user RDAREAs because shared RDAREAs cannot be defined in a HiRDB single server configuration.

### shared RDAREA

A user RDAREA that can be referenced from all back-end servers. Only a HiRDB parallel server configuration can define shared RDAREAs.

### shared table

A table that is stored in a shared RDAREA and can be referenced from all back-end servers. In order to achieve SQL and UAP compatibility with HiRDB parallel server configurations, shared tables can also be defined in HiRDB single server configurations. If defined, however, such shared tables are stored in user RDAREAs because shared RDAREAs cannot be defined in a HiRDB single server configuration.

### simple authenticated user

A HiRDB user using the facility for simple authentication of OS login users is called a *simple authenticated user*.

### site status

The status of a site as recognized by the log-only synchronous method. There are four statuses: initial, ready, transaction, and log application.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### skipped effective synchronization point dump monitoring facility

With the `pd_spd_syncpoint_skip_limit` system common definition operand, you can specify the maximum number of times synchronization point dumps can be skipped during a transaction.

If, for example, an infinite loop occurs in a UAP, synchronization point dump processing might not be performed several times in succession (processing might be skipped a number of times). If the number of times this processing is skipped exceeds a value specified by the user, the affected transaction is forcibly suspended and rolled back. This facility keeps track of the number of times in succession that synchronization dumps are skipped.

### space conversion facility

The space conversion facility unifies coding of single-byte and double-byte spaces that are intermixed in table data. The codes for double-byte spaces are shown below. Two single-byte spaces are coded as `X'2020'`.

- Shift-JIS Kanji Code: `X'8140'`
- EUC Japanese Kanji Code: `X'A1A1'` (UNIX edition only)
- EUC Chinese Kanji Code: `X'A1A1'`
- Chinese Kanji Code (GB18030): `X'A1A1'`
- Unicode (UTF-8): `X'E38080'`

### SQL connection

The logical connection of a UAP to an RD node for the purpose of executing an SQL statement.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### SQL extension optimizing options

SQL extension optimizing options enable optimization of SQL at execution time by determining the most efficient access paths that can be specified, taking into consideration the status of the database.

The following SQL extension optimizing options are provided:

1. Application of optimizing mode 2 based on cost
2. Hash-execution of a hash join or a subquery
3. Facility for applying join conditions that include value expressions
4. Enabling the substructure index to the `XMLEXISTS` predicate, including the parameters

### SQL object

An SQL object is the object that is compiled by HiRDB when SQL statements are defined and executed.

### SQL optimization options

SQL optimization options enable optimization of SQL at execution time by determining the most efficient access paths that can be specified, taking into consideration the status of the database.

The following SQL optimization options are provided:

1. Forced nest-loop-join
2. Creating multiple SQL objects
3. Increasing the target floatable servers (back-end servers for fetching data)
4. Prioritized nest-loop-join
5. Increasing the number of floatable server candidates
6. Priority of OR multiple index use
7. Group processing, `ORDER BY` processing, and `DISTINCT` set function processing at the local back-end server
8. Suppressing use of AND multiple indexes
9. Rapid grouping facility
10. Limiting the target floatable servers (back-end servers for fetching data)
11. Separating data collecting servers
12. Suppressing index use (forced table scan)

13. Forcing use of multiple indexes
14. Suppressing creation of update-SQL work tables
15. Derivation of rapid search conditions
16. Application of scalar-operation-included key conditions
17. Batch acquisition from plug-in-provided functions
18. Facility for moving search conditions into derived tables

### SQL optimization specification

Optimizations to enhance SQL search efficiency can be specified for SQL statements. The following three SQL optimization specifications are available:

- SQL optimization specification for index utilization
- SQL optimization specification for the join method
- SQL optimization specification for the subquery execution method

SQL optimization specifications take precedence over any specifications of SQL optimization options and SQL extension optimizing options.

### SQL preprocessor

A program that converts SQL statements into high-level language code so that they can be compiled by a high-level language compiler.

### SQL runtime warning output facility

After SQL code is executed, HiRDB checks its runtime. If this check determines that the SQL code's runtime is greater than the time specified (set as a ratio of the `PDCWAITTIME` value), this facility is used to output the following warning information for that SQL code:

- SQL runtime warning information file
- Warning message (KFPA20009-W)

### SQL stored function

An SQL stored function is a stored function in which procedures are coded in SQL.

### SQL stored procedure

An SQL stored procedure is a stored procedure in which procedures are coded in SQL.

### standby system switchover facility

A standby HiRDB separate from the HiRDB that is actively processing jobs is deployed, and if a failure occurs on the server machine or on HiRDB, job processing can be automatically switched over to the standby HiRDB. This is called the standby system switchover facility.

### standby-less system switchover (effects distributed) facility

A type of standby-less system switchover facility. In the event of an error, this facility distributes to multiple active units the processing requests for back-end servers (BESs) in the erroneous unit; this is called the standby-less system switchover (effects distributed) facility.

### standby-less system switchover facility

Unlike a standby system HiRDB that has been prepared for the standby system switchover facility, with the standby-less system switchover facility a standby HiRDB need not be prepared. If an error occurs, there is no system switchover to a standby HiRDB system, but another operating unit takes over the processing. This is called the standby-less system switchover facility.

The standby-less system switchover facility can be further subdivided as follows:

- Standby-less system switchover (1:1) facility
- Standby-less system switchover (effects distributed) facility

### standby-less system switchover (1:1) facility

Unlike the standby system switchover facility that provides a standby HiRDB system, the standby-less system switchover facility does not require that a standby HiRDB system be kept in reserve. In the event of an error, the standby-less system switchover facility transfers processing to another active unit without switching over to a standby HiRDB system. This is called the standby-less system switchover facility.

The standby-less system switchover (1:1) facility can perform 1-to-1 unit switchover in the event of an error and assign processing to another, designated back-end server.

### static SQL

A method of coding SQL within a program when a UAP is created. Contrast with *dynamic SQL*, which is a method of generating SQL code while a UAP is executing.

### statistics log file

A file that stores statistical information (statistics logs) output by HiRDB.

### status file

A file that stores system status information that might be needed to restart HiRDB is called a status file. There are two types of status files:

- Server status files
- Unit status files

### status file for log application processing

A status file required by the log-only synchronous method. It is used to obtain system status information when log application processing is performed at the log application site. Its counterpart, the status file for transaction processing, is required at the transaction execution site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### status file for transaction processing

A status file required by the log-only synchronous method. It is used when log application processing is performed at the log application site. Its counterpart, the status file for log application processing, is required at the transaction execution site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### stored function

A facility for registering a data process coded in SQL, Java, or C as a function in the database. Because input parameters can be assigned to a stored function and because a stored function can generate return values, a stored function can be called by specifying it as a value expression in an `SQL` statement. A stored function can be defined in a function in the `CREATE FUNCTION` or `CREATE TYPE` statement. Data processes coded in SQL, Java, or C are compiled during definition, and SQL object coding of the procedure is generated. The object, together with the definition information, is stored in the database.

### stored procedure

A stored procedure enables a sequence of database access operations coded in SQL, Java, or C to be stored as a procedure in a database. Output or input/output parameters can be assigned to a stored procedure, and a stored procedure can be called by the `CALL` statement in SQL. A stored procedure can be defined in a procedure in the `CREATE PROCEDURE` or `CREATE TYPE` statement. Database operations coded in SQL, Java, or C are compiled during definition, and SQL object coding of the access procedure is generated. The object, together with the definition information, is stored in the database.

### stored routine

Stored procedures and stored functions are collectively referred to as *stored routines* or *routines*.

### substitutability

Substitutability is the capability to substitute a value in a lower-order abstract data type for the value in the higher-order abstract data type.

### subtype

A subtype is an abstract data type that is created by customizing an abstract data type.

### Sun Java System Directory Server linkage facility

By using the Sun Java System Directory Server, you can manage and authenticate HiRDB users. This capability is provided by a process called the Sun Java System Directory Server linkage facility. By using this facility, you can centrally manage organizational and user information (user IDs, passwords, affiliations, job titles) that is otherwise managed separately in HiRDB, Groupmax, and other systems.

### supertype

An abstract data type that is higher in order than a specialized abstract data type (subtype) is called the supertype.

### synchronization point

The point at which a transaction is completed is called a synchronization point. Synchronization point processing that validates a transaction-induced update is called a commit, and synchronization point processing that invalidates a transaction is called a rollback.

### synchronization point dump file

In the event of abnormal termination of HiRDB, recovering it using system log information only will require system log information covering the entire time since HiRDB was startup, which will require a significant amount of time for recovery processing. Therefore, points can be established at fixed intervals during HiRDB operation, and required HiRDB management information can be saved at those points. This means that all system log information produced prior to the point to be used for recovery is no longer needed, thus reducing the recovery time. The file that stores the HiRDB management information obtained at each such point is called a synchronization point dump file.

### synchronization point dump file for log application processing

A synchronization point dump file that is required by the log-only synchronous method. It is used to obtain synchronization points when log application processing is performed at the log application site. Its counterpart, the synchronization point dump file for transaction processing, is required at the transaction execution site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### synchronization point dump file for transaction processing

A synchronization point dump file that is required by the log-only synchronous method. It is used when log application processing is performed at the log application site. Its counterpart, the synchronization point dump file for log application processing, is required at the transaction execution site.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### synchronous copy

One of the processing methods used to update-copy data to a remote site. After update-copy processing at the remote site is completed, update-copy processing at the main site is completed (update-copy processing at the main site waits for completion of the update-copy processing at the remote site).

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### synchronous DB state

In in-memory data processing, synchronous DB state means that data in the in-memory RDAREA is synchronized with data in the in-memory buffer.

### synchronous group

A group made up entirely of synchronous pair volumes.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### synchronous pair volume

A pair volume that has been created by specifying either `data` or `never` as the fence level. When data is written to the P-VOL, it is mirrored synchronously onto the S-VOL. When paired, no differences exist between the data on the P-VOL and the data on the S-VOL.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### system file

A system file stores information that will be needed if it becomes necessary to recover the system in the event of an error. *System file* is a generic term that includes the following files:

- System log files
- Synchronization point dump files
- Status files

### system log file

A system log file stores a history of database updates; it is also called a journal file. Historical information on database updates is called the system log (or system log information), and it is used by HiRDB to recover the database in the event of abnormal termination of either HiRDB or a UAP. The system log is also used as input information for recovery of the database.

### system RDAREA

System RDAREA is a generic term for the following types of RDAREAs:

- Master directory RDAREA
- Data directory RDAREA
- Data dictionary RDAREA

### system switchover facility

When a standby server machine is provided separately from the server machine that is processing jobs, job processing can be switched automatically by HiRDB to the standby server machine in the event the running server machine fails. This is called the system switchover facility.

### table reorganization

Over time, additions and updates of data in a table tend to fragment the arrangement of rows, giving rise to unusable free space. Table reorganization reorganizes table data in a user RDAREA or LOB data in a LOB RDAREA to eliminate unusable free space. The database reorganization utility (`pdrorg` command) is used to reorganize a table.

### tape device access facility

A facility that enables access to files on DAT, DLT, and LTO. The tape device access facility is used for the following types of files:

- Input data files (input data files specified in the `source` statement of the `pdload` command)
- Unload data files (unload data files specified in the `unload` statement of the `pdrorg` command)
- Unload data files (LOB data unload data files specified in the `lobunld` statement of the `pdrorg` command)
- Backup files (backup files specified with the `-b` option of the `pdcopy` or `pdrstr` command)

This term has no practical application for UNIX users, because it is related to a facility that cannot be used with a UNIX edition of HiRDB.

### temporary table

A base table that exists only while a transaction or SQL session is being executed. An index that is defined in a temporary table is called a *temporary table index*.

### temporary table RDAREA

A user temporary table RDAREA and temporary table indexes.

### transaction

A transaction is a unit of logical work; for example, a series of database operations. The transaction is also the basic unit for recovery and locking.

### transaction execution site

In the log-only synchronous method, the site that accepts transactions.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### trigger

By defining a trigger, you can have SQL statements automatically execute when an operation (updating, insertion, deletion) is performed on a particular table. When a particular table is updated, a trigger based on the associated event allows you to automatically perform operations such as updating another table as well.

### unbalanced index split

Unlike normal page split, the unbalanced index split method splits the data in an index page into two unequal parts, rather than into two equal parts. Index storage efficiency is improved by using this method when an ascending or descending middle key is to be added.

### uniqueness constraint

The uniqueness constraint is a limitation that prohibits a data value from appearing more than once in a column (all data values in the column must be unique).

### unit

A unit refers to the HiRDB operation environment within a single server machine.

### unit controller

A unit controller is a system that controls and monitors server execution in a unit and that controls communication between units.

### unload log file

A file created by unloading a system log file (system log information) is called an unload log file.

### unload statistics log file

A file created by unloading the contents of a statistics log file.

### unused page

A page that is not being used.

### unused segment

A segment that is not being used. Such a segment can be used by any table (or index) in the RDAREA.

### updatable back-end server

A back-end server on which shared tables and shared indexes in shared RDAREAs can be updated.

### updatable online reorganization

Refers to functionality that allows databases to be accessed and updated during database reorganization. Processes that access or update a database perform the operation on a replica database. To perform updatable online reorganization, you must install HiRDB Staticizer Option, and you must also specify related operands in the HiRDB system definitions.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### update buffer

When data is updated, the data is first updated into a global buffer, before being updated in the database. The buffer that stores this data before the database is updated is called an *update buffer*.

### update copy

An operation in which, when updating of data on a primary volume occurs, the updated data is also mirrored (copied) onto a secondary volume. Update copying maintains integrity between the primary volume and the secondary volume.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### used page

A page in which a table or an index is stored. A used page that is completely filled with data such that no more can be added is called a full page, and a used page from which data has been deleted so that it no longer contains data is called a used free page.

### used segment

A segment in which table or index data is stored. A used segment that is completely filled with data, such that no more can be added, is called a *full segment*. A used segment that is not full is called a *space-available segment*.

A space-available segment from which data has been deleted so that only free pages remain (used free pages or unused pages) is called a *used free segment*

### user-defined type

A data type that can be defined by the user is called a user-defined type. Abstract data types are user-defined types.

### user LOB RDAREA

A user LOB RDAREA stores large variable object data, such as documents, images, and sounds. The following types of data must be stored in a user LOB RDAREA:

- Column with the BLOB type specification (BLOB column)
- Attribute with BLOB specification in an abstract data type
- Plug-in index

### user RDAREA

A user RDAREA stores tables and indexes created by a user.

### user server hot standby

A facility that starts server processes on a standby HiRDB beforehand, so that startup processing of server processes is not performed when the system is switched over. The system switchover time is reduced by the amount of time required to perform startup processing of server processes.

In addition to user server hot standby, the rapid system switchover facility is available for reducing system switchover time. The rapid system switchover facility can reduce the system switchover time more than user server hot standby can (the rapid system switchover facility includes the user server hot standby functionality).

### utility special unit

A utility special unit is a server machine in which only input/output devices to be used for executing a utility are set up. A utility special unit can be set up only in a HiRDB parallel server configuration. The following utilities can use a utility special unit:

- Database load utility
- Database reorganization utility
- Dictionary import/export utility
- Database copy utility
- Database recovery utility

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### value (instance)

The term value refers to a specific value of an abstract data type.

### view table

A newly defined virtual table in which specific rows and columns are selected from a base table.

### volume attribute

There are three types of volumes, primary volumes (P-VOL), secondary volumes (S-VOL), and simplex volumes (SMPL). The volume attribute refers to the attribute indicating one of these three volume types.

This term has no practical application for Windows users, because it is related to a facility that cannot be used with a Windows edition of HiRDB.

### work table file

A file that stores temporary information needed in order to execute an SQL statement is called a work table file.

# Index