Informix Product Family
Informix
Version 11.70

IBM Informix Administrator's Guide

Informix Product Family
Informix
Version 11.70

IBM Informix Administrator's Guide

IBM

Contents

Part 3. Logging and log administration

Chapter 11. Logging 11-1

Chapter 12. Manage the database-logging mode 12-1

Chapter 13. Logical log 13-1

Part 6. Distributed data

Chapter 25. Multiphase commit protocols 25-1

Chapter 26. Manually recovering from failed two-phase commit 26-1

Part 7. Overview of automatic monitoring and corrective actions

Chapter 27. The Scheduler 27-1

Part 8. Appendixes

Introduction

In this introduction

This introduction provides an overview of the information in this publication and describes the conventions it uses.

About this publication

This publication describes concepts and procedures for configuring, administering, and using IBM® Informix®.

A companion volume, the *IBM Informix Administrator's Reference*, contains reference material for using IBM Informix database servers. If you need to tune the performance of your database server and SQL queries, see your *IBM Informix Performance Guide*.

Types of users

This publication is written for the following users:
- Database users
- Database administrators
- Database server administrators
- Performance engineers
- Programmers in the following categories
  - Application developers
  - DataBlade® module developers
  - Authors of user-defined routines

This publication is written with the assumption that you have the following background:
- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

Software dependencies

This publication is written with the assumption that you are using IBM Informix Version 11.70 as your database server.

Assumptions about your locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation and representation of numeric data, currency, date, and time that is used by a language within a given territory and encoding is brought together in a single environment, called a Global Language Support (GLS) locale.

The IBM Informix OLE DB Provider follows the ISO string formats for date, time, and money, as defined by the Microsoft OLE DB standards. You can override that default by setting an Informix environment variable or registry entry, such as **DBDATE**.

If you use Simple Network Management Protocol (SNMP) in your Informix environment, note that the protocols (SNMPv1 and SNMPv2) recognize only English code sets. For more information, see the topic about GLS and SNMP in the *IBM Informix SNMP Subagent Guide*.

The examples in this publication are written with the assumption that you are using one of these locales: en_us.8859-1 (ISO 8859-1) on UNIX platforms or en_us.1252 (Microsoft 1252) in Windows environments. These locales support U.S. English format conventions for displaying and entering date, time, number, and currency values. They also support the ISO 8859-1 code set (on UNIX and Linux) or the Microsoft 1252 code set (on Windows), which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

You can specify another locale if you plan to use characters from other locales in your data or your SQL identifiers, or if you want to conform to other collation rules for character data.

For instructions about how to specify locales, additional syntax, and other considerations related to GLS locales, see the *IBM Informix GLS User's Guide*.

Demonstration databases

The DB-Access utility, which is provided with your IBM Informix database server products, includes one or more of the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in IBM Informix publications are based on the **stores_demo** database.
- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines.

For information about how to create and populate the demonstration databases, see the *IBM Informix DB-Access User's Guide*. For descriptions of the databases and their contents, see the *IBM Informix Guide to SQL: Reference*.

The scripts that you use to install the demonstration databases are in the $INFORMIXDIR/bin directory on UNIX platforms and in the %INFORMIXDIR%\bin directory in Windows environments.

What's new in administration for Informix, 11.70

This publication includes information about new features and changes in existing functionality.

For a complete list of what's new in this release, see the release notes or the information center at http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.po.doc/new_features.htm.

Table 1. What's new in the IBM Informix Administrator's Guide for version 11.70xC3

Overview	Reference
Automatic read-ahead operations You can enable the database server to use read-ahead operations automatically to improve performance. Most queries can benefit from processing the query while asynchronously retrieving the data required by the query. The database server can automatically use asynchronous operations for data or it can avoid them if the data for the query is already cached. Use the AUTO_READAHEAD configuration parameter to configure automatic read-ahead operations for all queries, and use the SET ENVIRONMENT AUTO_READAHEAD statement to configure automatic read-ahead operations for a particular session. The RA_THRESHOLD configuration parameter is deprecated with this release.	"Automatic read-ahead operations" on page 6-24 AUTO_READAHEAD configuration parameter (Administrator's Reference)
Configuring the server response to low memory You can configure the actions that the server takes to continue processing when memory is critically low. You can specify the criteria for terminating sessions based on idle time, memory usage, and other factors so that the targeted application can continue and avoid out-of-memory problems. Configuring the low memory response is useful for embedded applications that have memory limitations.	"Configure the server response when memory is critically low" on page 7-8 "Scenario for maintaining a targeted amount of memory" on page 7-8
Reserving memory for critical activities You can enable the server to reserve a specific amount of memory for use when critical activities (such as rollback activities) are needed and the server has limited free memory. If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, the critical activities can complete even when you get out-of-memory errors. You can also dynamically adjust the value of the LOW_MEMORY_RESERVE configuration parameter with the **onmode -wm** or **-wf** command.	"Reserve memory for critical activities" on page 7-7

Table 1. What's new in the IBM Informix Administrator's Guide for version 11.70xC3 (continued)

Overview	Reference
Connection Manager enhancements You can configure a single Connection Manager instance to automatically manage client connection requests for a combination of high availability clusters, grids, server sets, and Enterprise Replication (ER) servers. Before version 3.70.xC3 you had to use a separate Connection Manager instance to manage each type of connection unit. For example, you had to use one Connection Manager instance for database servers that were in a grid and another Connection Manager instance for database servers that were in a high-availability cluster. To make configuration easier, most of the command-line options are deprecated and options are set using the configuration file. In addition, the format of the configuration file is more intuitive than before. Because a single Connection Manager instance supports multiple connection units, you should configure backup Connection Manager instances in case the primary instance fails.	Chapter 23, "Connection Management," on page 23-1
Non-root installations support shared-memory and stream-pipe connections You can use shared-memory and stream-pipe connections for client communications with database servers that are installed without root privileges. In the sqlhosts file, you set the new cfd option to specify a directory for storing required communication files for shared-memory and stream-pipe connections.	See "sqlhosts file and SQLHOSTS registry key options" on page 2-25 Connections to a non-root installation (UNIX, Linux) (Security Guide)

Table 2. What's new in the IBM Informix Administrator's Guide for version 11.70xC2

Overview	Reference
Improve network connection performance and scalability You can improve the performance and scalability of network connections on UNIX operating systems by using the NUMFDSERVERS configuration parameter. Use the configuration parameter to adjust the maximum number of poll threads for the server to use when migrating TCP/IP connection across virtual processors (VPs). Adjusting the number of poll threads, for example, by increasing the number of poll threads beyond the default number, is useful if the database server has a high rate of new connect and disconnect requests or if you find a high amount of contention between network shared file (NSF) locks. For example, if you have multiple CPU VPs and poll threads and this results in NSF locking, you can increase the value of NUMFDSERVERS (and you can increase the number of poll threads specified in the NETTYPE configuration parameter) to reduce NSF lock contention.	"Connection information set in the NUMFDSERVERS configuration parameter" on page 2-44 NUMFDSERVERS configuration parameter (Administrator's Reference)

Table 3. What's New in IBM Informix Administrator's Guide for version 11.70.xC1

Overview	Reference
New editions and product names IBM Informix Dynamic Server editions were withdrawn and new Informix editions are available. Some products were also renamed. The publications in the Informix library pertain to the following products: • IBM Informix database server, formerly known as IBM Informix Dynamic Server (IDS) • IBM OpenAdmin Tool (OAT) for Informix, formerly known as OpenAdmin Tool for Informix Dynamic Server (IDS) • IBM Informix SQL Warehousing Tool, formerly known as Informix Warehouse Feature	For more information about the Informix product family, go to http://www.ibm.com/software/data/informix/.
Transaction completion during cluster failover Active transactions on secondary servers in a high-availability cluster now run to completion if the primary server encounters a problem and fails over to a secondary server. Previous versions of Informix rolled back all active transactions when a failover occurred.	"Transaction completion during cluster failover" on page 22-43 FAILOVER_TX_TIMEOUT Configuration Parameter
Quickly clone a primary server You can now use the **ifxclone** utility to clone a primary server with minimal setup and configuration. Previously to clone a server it was necessary to create a level-0 backup, transfer the backup to the new system, restore the image, and initialize the instance. The **ifxclone** utility starts the backup and restore processes simultaneously and there is no need to read or write data to disk or tape. You can use the **ifxclone** utility to create a standalone server or a remote standalone secondary server. For example, you can quickly, easily, and securely clone a production system to a test system. The **ifxclone** utility requires the DIRECT_IO configuration parameter to be set to 0 on both the source and target servers.	"Cloning a primary server" on page 22-29
Running DDL statements on secondary servers You can automate table management in high-availability clusters by running Data Definition Language (DDL) statements on all servers. You can run most DDL statements such as CREATE, ALTER, and DROP on secondary servers. In previous releases, only Data Manipulation Language (DML) statements could be run on secondary servers.	"Database updates on secondary servers" on page 22-32

Table 3. What's New in IBM Informix Administrator's Guide for version 11.70.xC1 (continued)

Overview	Reference
Automatic storage provisioning ensures space is available You can configure Informix to automatically expand an existing storage space if the space is full. You can also configure Informix to expand the space before it is full, when its free pages fall below a specified threshold. When you enable and configure automatic storage provisioning, you do not need to manually add storage space to avoid out-of-space errors. Even if you prefer to add space manually, automatic storage provisioning simplifies the process of adding space, because you do not need to determine where to get the space. You enable or disable the automatic expansion of storage spaces by setting the new SP_AUTOEXPAND configuration parameter. You can expand a storage space in one of two ways: • Enable chunk extensions by marking specific chunks as extendable, using a new SQL administrative API command. • Enable automatic chunk creation by using another new SQL administrative API command to add valid entries to the Informix storage pool. Valid entries can include available directories, cooked files, and raw devices. In addition, you can use new SQL administrative API commands to: • Manually extend a chunk. • Modify the amount by which the data server can automatically expand a particular storage space. • Disable automatic expansion for a storage space. • Manually create storage spaces and chunks from the storage pool. • Return space to the storage pool, while dropping chunks and storage spaces. • Manage the storage pool by adding, modifying, and deleting entries. Automatic storage provisioning is supported in a high-availability cluster. In this environment, any storage pool entry (directory, cooked file, or raw device) on the primary server must also be available through the same path on all secondary servers.	"The storage pool" on page 8-34 "Extendable chunks" on page 8-4 "Automatic space management" on page 9-26 "Creating and managing storage pool entries" on page 9-27 "Marking a chunk as extendable or not extendable" on page 9-29 "Modifying the create or extend size of a storage space" on page 9-29 "Changing the threshold and wait time for the automatic addition of more space" on page 9-30 "Manually expanding a space or extending an extendable chunk" on page 9-31 "Creating a space or chunk from the storage pool" on page 9-38 "Returning empty space to the storage pool" on page 9-39
Automatically terminate idle sessions You can automatically terminate sessions with clients that have been idle for a specified time by enabling the **idle_user_timeout** Scheduler task. The default idle time out value is 60 minutes. You enable and configure the threshold and frequency for the task by updating the task in the **sysadmin** database.	"Automatically terminating idle connections" on page 1-16 "Built-in tasks and sensors" on page 27-3

Table 3. What's New in IBM Informix Administrator's Guide for version 11.70.xC1 (continued)

Overview	Reference
Notification of corrupt indexes If an index is corrupted and needs to be rebuilt, an alert with information about the index is added to the ph_alert table in the sysadmin database and displayed in the IBM OpenAdmin Tool (OAT) for Informix. Previously, you had to manually run the **oncheck** utility to determine if an index was corrupted.	"Validate indexes" on page 19-2 "Built-in tasks and sensors" on page 27-3
Defragmenting partitions A frequently updated table can become fragmented over time and this fragmentation degrades performance when the table is accessed by the server. You can improve performance by defragmenting partitions to merge noncontiguous extents. Defragmenting a table brings data rows closer together and avoids partition header page overflow problems. Use the SQL administration API **task()** or **admin()** function with the `defragment` argument and specify the table name or partition number that you want to defragment.	"Defragment partitions" on page 9-57
Automatically add CPU virtual processors When the database server starts, it checks that the number of CPU virtual processors is at least half the number of CPU processors on the database server computer. This ratio of CPU processors to CPU virtual processors is a recommended minimum to ensure that the database server performs optimally in most situations. If necessary, the database server automatically increases the number of CPU virtual processors to half the number of CPU processors.	"Built-in tasks and sensors" on page 27-3 Automatic addition of CPU VPs
Automatically optimize data storage You can automatically compress, shrink, repack, and defragment tables and fragments by enabling the **auto_crsd** Scheduler task. You can disable or update the thresholds for each of these operations.	"Built-in tasks and sensors" on page 27-3 "Automatically optimizing data storage" on page 9-56
Compression functionality automatically enabled You can now compress data in a table or table fragment without first enabling compression. Previously you needed to run an SQL administration API command with the enable compression argument to enable the compression of data in a table or table fragment.	"Data compression" on page 9-58

Table 3. What's New in IBM Informix Administrator's Guide for version 11.70.xC1 (continued)

Overview	Reference
Improve name service connection time You can use the new NS_CACHE configuration parameter to define the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache. If you disable one or more of these caches by setting the retention time to 0, the database server queries the operating system for the host, service, user, or group information. Getting information from the name service cache instead of querying the operating system decreases the amount of time needed to establish connections. The database server also now supports multiple listener threads for one service (port) for the **onsoctcp** and the **onimcsoc** protocols.	"Name service maximum retention time set in the NS_CACHE configuration parameter" on page 2-43 "Multiple listen threads" on page 4-24 "Add listen threads" on page 4-25
Prevent the accidental disk initialization of your instance or another instance You can use the new FULL_DISK_INIT configuration parameter to prevent the major problems that can occur if you or someone else accidentally initializes your instance or another instance when the first page of the first chunk (page zero) exists at the root path location. Page zero, which is created when Informix is initialized, is the system page that contains general information about the server. The FULL_DISK_INIT configuration parameter specifies whether or not the disk initialization command (**oninit -i**) can run on your Informix instance when a page zero exists at the root path location. When this configuration parameter is set to 0, the **oninit -i** command runs only if there is not a page zero at the root path location. If you change the setting of the FULL_DISK_INIT configuration parameter to 1, the **oninit -i** command runs under all circumstances, but also resets the FULL_DISK_INIT configuration parameter to 0 after the disk initialization.	"Initialize disk space" on page 3-4

Example code conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement. If you are using DB–Access, you must delimit multiple statements with semicolons.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional documentation

Documentation about this release of IBM Informix products is available in various formats.

You can access or install the product documentation from the Quick Start Guide CD that is shipped with Informix products. To get the most current information, see the Informix information centers at ibm.com®. You can access the information centers and other Informix technical information such as technotes, white papers, and IBM Redbooks® publications online at http://www.ibm.com/software/data/sw-library/.

Compliance with industry standards

IBM Informix products are compliant with various standards.

IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

The IBM Informix Geodetic DataBlade Module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)—Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).

Syntax diagrams

Syntax diagrams use special components to describe the syntax for statements and commands.

Table 4. Syntax Diagram Components

Component represented in PDF	Component represented in HTML	Meaning
▶▶───────────	>>----------------------	Statement begins.
──────────────▶	---------------------->	Statement continues on next line.
▶────────────	>----------------------	Statement continues from previous line.

Table 4. Syntax Diagram Components (continued)

Component represented in PDF	Component represented in HTML	Meaning
►	`---------------------><`	Statement ends.
────SELECT────	`--------SELECT----------`	Required item.
LOCAL	`--+-----------------+---` `  '------LOCAL------'`	Optional item.

ALL
DISTINCT
UNIQUE
	`---+-----ALL-------+---`
	`   +--DISTINCT-----+`
	`   '---UNIQUE------'`
		Required item with choice. Only one item must be present.

FOR UPDATE
FOR READ ONLY
	`---+-----------------+---`
	`   +--FOR UPDATE-----+`
	`   '--FOR READ ONLY--'`
		Optional items with choice are shown below the main line, one of which you might specify.

NEXT
PRIOR
PREVIOUS
	`    .---NEXT---------.`
	`----+---------------+---`
	`    +---PRIOR--------+`
	`    '---PREVIOUS-----'`
		The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line is used by default.

,
index_name
table_name
	`    .-------,----------.`
	`    V               |`
	`---+----------------+---`
	`    +---index_name---+`
	`    '---table_name---'`
		Optional items. Several items are allowed; a comma must precede each repetition.

►►─┤Table Reference├─►◄	`>>-| Table Reference |-><`	Reference to a syntax segment.

Table Reference
view
table
synonym
	`Table Reference`
	`|--+-----view--------+--|`
	`   +------table------+`
	`   '----synonym------'`
		Syntax segment.

How to read a command-line syntax diagram

Command-line syntax diagrams use similar elements to those of other syntax diagrams.

Some of the elements are listed in the table in Syntax Diagrams.

Creating a no-conversion job

►►──onpladm create job─*job*───────────── -n─ -d─*device*─ -D─*database*──────►
                          └─ -p─*project*─┘

```
►►─ -t─table──────────────────────────────────────────────────────────────────►
```

```
        ┌─────────────────────────────────────────────────┐
►─────────┴──────────────────────────────────────────────────┴──────────────────►◄
                                                        (1)
          ┌─ -S─server─┐   ┌─ -T─target─┐   ├ Setting the Run Mode ┤
```
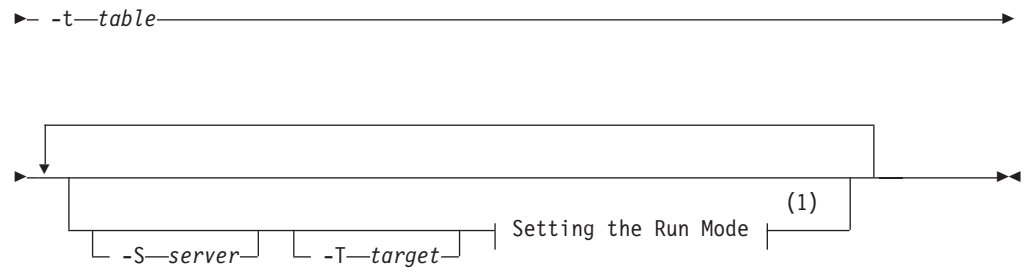
Notes:

1   See page Z-1

This diagram has a segment named "Setting the Run Mode," which according to the diagram footnote is on page Z-1. If this was an actual cross-reference, you would find this segment on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

Setting the run mode:

```
                     ┌─l─┐
                     │ └─c─┘ │
├── -f ──┬───────┬──┼───────┼──┬───┬──┬───┬──────────────────────────────┤
         ├─d─┤     └─u─────┘   └─n─┘  └─N─┘
         ├─p─┤
         └─a─┘
```
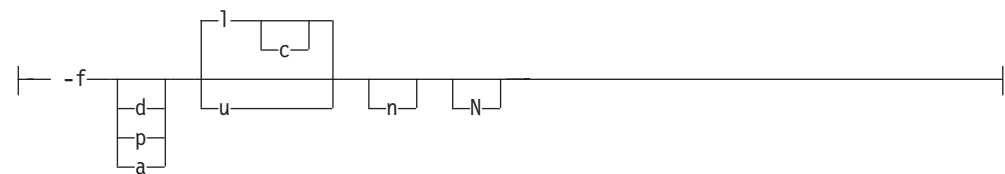
To see how to construct a command correctly, start at the upper left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diagram are case-sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case-sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:
1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
   - **-n**
   - **-d** and the name of the device
   - **-D** and the name of the database
   - **-t** and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   - **-S** and the server name
   - **-T** and the target server name
   - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Keywords and punctuation

Keywords are words reserved for statements and all commands except system-level commands.

When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples.

You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.

►►──SELECT──*column_name*──FROM──*table_name*──────────────────────────────────────►◄

When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

How to provide documentation feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:

- Send email to docinf@us.ibm.com.
- In the Informix information center, which is available online at http://www.ibm.com/software/data/sw-library/, open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.
- Add comments to topics directly in the information center and read comments that were added by other users. Share information about the product documentation, participate in discussions with other users, rate topics, and more!

Feedback from all methods is monitored by the team that maintains the user documentation. The feedback methods are reserved for reporting errors and omissions in the documentation. For immediate help with a technical problem, contact IBM Technical Support at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Part 1. The database server

Chapter 1. Database server installation and configuration

You have various options to choose from when you install and initially configure the database server. You can also install IBM OpenAdmin Tool (OAT) for Informix to monitor and administer database server instances from a single location.

When you install IBM Informix, follow the installation instructions to ensure that all prerequisites are met (for example, the permissions of all key files and directories are set appropriately). The installation instructions are in the *IBM Informix Installation Guide for UNIX, Linux, and Mac OS X* and the *IBM Informix Installation Guide for Windows*.

After you install a new version of the database server, you must configure it. *Configuration* is setting specific parameters that customize the database server for your data-processing environment: quantity of data, number of tables, types of data, hardware, number of users, and security requirements.

Configuration parameters are sometimes changed or removed from the database server. For information about changed or removed configuration parameters, see Configuration parameters that have been changed or removed.

Plan for the database server

Configuring a database management system requires many decisions, such as where to store the data, how to access the data, and how to protect the data. How you install and configure IBM Informix can greatly affect the performance of database operations.

You can customize the database server so that it functions optimally in your particular data-processing environment. For example, using a database server to serve 1000 users who execute frequent, short transactions is very different from using a database server to serve a few users making long and complicated searches.

When you are planning for your database server, consider your priorities and your environment.

Consider your priorities

As you prepare the initial configuration and plan your backup strategy, keep in mind the characteristics of your database server:
- Do applications on other computers use this database server instance?
- What is the maximum number of users that you can expect?
- To what extent do you want to control the environment of the users?
- Are you limited by resources for space, CPU, or availability of operators?
- Do you require the database server to run unsupervised?
- Does the database server usually handle many short transactions or fewer long transactions?
- Which new database server features or related products do you plan to use?

Considering your environment

Before you start the initial configuration of your database server, you must obtain information about the computers in your environment.

Collect as much of the following information as possible. You might require the assistance of your system administrator to obtain this information:

- The host names and IP addresses of the other computers on your network
- Does your UNIX platform support the Network Information Service (NIS)?
- The disk-controller configuration

  How many disk drives are available? Are some of the disk drives faster than others? How many disk controllers are available? What is the disk-controller configuration?

- What are the requirements, features, and limitations of your storage manager and backup devices?

  For more information, see the *IBM Informix Storage Manager Administrator's Guide* or your storage-manager documentation.

- Are you upgrading the hardware and operating system? Is your operating system 32-bit or 64-bit?
- Operating-system shared memory and other resources

  How much shared memory is available? How much of it can you use for the database server?

UNIX only: The machine notes file indicates which parameters are applicable for each UNIX platform.

Configure the operating system

Before you can start configuring the database server, you must configure the operating system appropriately. You might require the assistance of the system administrator for this task.

The 32-bit version of IBM Informix runs on a 64-bit or 32-bit operating system. The 64-bit version of Informix must run on a 64-bit operating system. For more information, see "Memory use on 64-bit platforms" on page 6-32.

Configure Windows memory

On Windows, you must create NTFS partitions and configure memory. Also see "Provide NTFS partitions in Windows" on page 1-4.

Insufficient memory for the database server can result in excessive buffer-management activity. When you set the Virtual Memory values in the System application on the Control Panel, ensure that you have enough paging space for the total amount of physical memory.

Maximum address space

On Windows, the maximum address space per computer system is:

- 1.7 gigabytes if the `boot.ini` file is not modified to 3 gigabytes.
- 2.7 gigabytes if the `boot.ini` file is modified to 3 gigabytes.

You can use the **/userva=xxxx** switch for more precise tuning of user and kernel virtual memory space. Use this switch with the 3-gigabyte switch in the `boot.ini`

file to tune the user-mode space to a value that is between 2 and 3 gigabytes, with the difference (3,072 less xxxx) given back to the kernel mode. Note that xxxx is a value expressed in megabytes.

The following sample `boot.ini` file shows how to use the new switch to tune a computer to allocate 2,900 megabytes of user-mode virtual memory and 1,196 megabytes of kernel-mode virtual memory. This increases the available kernel space by 172 megabytes.

```
[Boot Loader]
Timeout=30
Default=multi(0)disk(0)rdisk(0)partition(2)\WINNT
[Operating Systems]
multi(0)disk(0)rdisk(0)partition(2)\WINNT="Microsoft Windows Server 2003"
/fastdetect /3GB /Userva=2900
```

Modify UNIX kernel parameters

The machine notes file contains recommended values for configuring operating-system resources. Use these recommended values when you configure the operating system.

If the recommended values for the database server differ significantly from your current environment, consider modifying your operating-system configuration. For more information, see your *IBM Informix Performance Guide*.

On some operating systems, you can specify the amount of shared memory allocated to the database server. The amount of available memory influences the values that you can choose for the shared-memory parameters in your configuration file. In general, increasing the space available for shared memory enhances performance. You might also be required to specify the number of locks and semaphores.

For background information about the role of the UNIX kernel parameters, see Chapter 7, "Manage shared memory," on page 7-1.

Configure disk space

Configure your disks to obtain optimum performance with data marts and data warehouses. Disk I/O is the longest portion of the response time for an SQL operation. The database server offers parallel access to multiple disks on a computer. Before you allocate the disk space, study the information about disk space in your operating-system administration guide and see "Disk allocation for chunks" on page 8-3.

Use large chunks

By default, the size of chunks for dbspaces is 4 terabytes for a 2-KB page. Chunks can be located anywhere in a 64-bit address space.

If you have upgraded from a version before version 10.00, you can activate the creation of large chunks by running the **onmode -BC 2** command.

Creating chunk files on UNIX

On UNIX, you can store data in chunks that use either unbuffered (*raw*) disks or operating-system files, also known as *buffered* or *cooked* files.

Raw or unbuffered disk access

UNIX provides unbuffered disk access using character-special devices (also known as *raw* disk devices). To create raw disk devices on UNIX, follow the instructions provided with your operating system.

The database server uses raw disk access to improve the speed and reliability of disk I/O operations. Raw disk access bypasses the file-buffering mechanism that the operating system provides. The database server itself manages the data transfers between disk and memory. The database server optimizes table access by guaranteeing that rows are stored contiguously.

Important: While you must use raw disk devices on UNIX to achieve better performance, recent advances in I/O caching for cooked writes can provide similar if not better performance. To determine the best device performance, perform benchmark testing on the system with both types of devices for the dbspace and table layout.

To allocate disks for the database server:
1. Configure a raw disk device for each disk.
2. Create standard device names or file names.
3. Set permissions, ownership, and group for each raw disk device.

Cooked files

If optimum performance is unimportant, you can configure the database server to store data in *cooked* files. Cooked files are easier to set up than raw disk devices.

Provide NTFS partitions in Windows

On Windows, install the database server on a New Technology File System (NTFS) or File Allocation Table (FAT) partition. However, you must store all dbspaces, blobspaces, and sbspaces in NTFS files or on a physical drive or logical disk partition. Use NTFS files to simplify disk administration. For more information about NTFS files, see your Windows documentation and "Disk access on Windows" on page 8-3.

If all of your partitions are FAT partitions, you must convert at least one partition to NTFS. You can use the Windows **convert** utility, as the following example shows:

```
convert /fs:ntfs
```

Set permissions, ownership, and group

Files or raw disk devices that the database server uses must have the appropriate ownership and permissions.

UNIX only: On UNIX, the owner and group must be **informix**, and the permissions must be set to read and write for both the user and the group (but not for others).

If you want users other than **informix** or **root** to run ON-Bar commands, create a **bargroup** group. Only members of **bargroup** can run ON-Bar commands. The **bargroup** group is not created automatically during database server installation. For instructions on creating a group, see the *IBM Informix Installation Guide for UNIX, Linux, and Mac OS X* or your UNIX documentation.

Windows only: On Windows, files must be owned by a member of the **Informix-Admin** group. The **Informix-Admin** group is created automatically when you install the database server.

Create standard device names (UNIX)

Use symbolic links to assign abbreviated standard device names for each raw disk device. If you have symbolic links, you can replace a disk that has failed with a new disk by assigning the symbolic name to the new disk.

To create a link between the character-special device name and another file name, use the UNIX link command (usually **ln**).

Run the UNIX command **ls -l** on your device directory to verify that both the devices and the links exist. The following example shows links to raw disk devices. If your operating system does not support symbolic links, you can use hard links.

```
ls -l
crw-rw--- /dev/rxy0h
crw-rw--- /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Set environment variables

To start, stop, or access a database server, each user must hold the required database access privileges, and must set the appropriate environment variables. Some environment variables are required; others are optional.

Required environment variables

The *IBM Informix Guide to SQL: Reference* contains a complete list of environment variables. For information about how setting environment variables can affect performance, see your *IBM Informix Performance Guide*.

The following table shows the environment variables that you must set before you access the database server or perform most administrative tasks.

Table 1-1. Required environment variables

Environment variable	Description
CLASSPATH	If you are using J/Foundation, specifies the location of `jvphome/krakatoa.jar` file so that Java™ Development Kit (JDK) can compile the Java source files.
INFORMIXDIR	Specifies the directory where you installed your IBM Informix database server.
INFORMIXSERVER	Specifies the name of the default database server. It has the value specified for the DBSERVERNAME or DBSERVERALIASES configuration parameter.
JVPHOME	If you are using J/Foundation, specifies the directory where you installed the IBM Informix JDBC Driver.
ONCONFIG	Specifies the name of the active `onconfig` file. All users who use database server utilities such as **onstat** must set the **ONCONFIG** environment variable. Users running client applications are not required to set the **ONCONFIG** environment variable. If the **ONCONFIG** environment variable is not present, the database server uses configuration values from the `onconfig` file: On UNIX: `$INFORMIXDIR/etc/onconfig` On Windows: `%INFORMIXDIR%\etc\onconfig.std`

Table 1-1. Required environment variables (continued)

Environment variable	Description
PATH	Specifies the location of executable files. On UNIX: `$INFORMIXDIR/bin` On Windows: `%INFORMIXDIR%\bin`
TERM	Enables DB-Access to recognize and communicate with the terminal that you are using. This environment variable is not required for initialization or startup, but it must be set before you can run an application.
TERMCAP TERMINFO INFORMIXTERM	Specifies whether DB-Access uses the information in the `termcap` file or the `terminfo` directory. If required for your system, you might require assistance from the UNIX system administrator to set these variables because they are highly system-dependent.

Setting environment variables

Tip: Set the environment variables in the appropriate startup file for your shell file or Windows.

You can include the environment variable **$INFORMIXDIR** in the configuration file. It must be the first path name value in path name specifications.

To set the required environment variables:

1. Set **INFORMIXDIR** to the directory where you installed the IBM Informix products.
2. Set the **PATH** environment variable to include `$INFORMIXDIR/bin` (UNIX) or `%INFORMIXDIR%\bin` (Windows).
3. Set **INFORMIXSERVER** to the name of the database server.

Set GLS environment variables

Set Global Language Support (GLS) environment variables if you want to use a language other than the default, U.S. English.

Use the following environment variables to work with GLS:

- **CLIENT_LOCALE**
- **DB_LOCALE**
- **SERVER_LOCALE**
- **DBLANG**
- **C8BITLEVEL**
- **ESQLMF**
- **GLS8BITFSYS**
- **GL_DATE**
- **GL_DATETIME**

If you plan to use Unicode, set the **GL_USEGLU** environment variable to improve the support of UTF-8 encoding.

For more information, see the *IBM Informix GLS User's Guide*.

Set environment variables on UNIX

Set UNIX environment variables in one of the following ways:

- At the system prompt on the command line

  When you set an environment variable at the system prompt, you must reassign it the next time you log-in to the system.

- In an environment-configuration file such as `$INFORMIXDIR/etc/informix.rc` or `.informix`

  An environment-configuration file is a common or private file where you can set environment variables for each database server user. Use of an environment-configuration file reduces the number of environment variables that you must set at the command line or in a shell file.

- In your `.profile` or `.login` file

  When you set an environment variable in your `.login`, `.cshrc`, or `.profile` file, it is assigned automatically every time you log-in to the system. For information about these files, see your operating-system manuals.

To override environment variables that have been automatically set, use a private environment-variable file, `~/.informix`, or assign new values to environment variables individually.

To check the validity of environment variables, use the **chkenv** utility.

The following example shows a setup file that contains environment variables for the **miami** database server. **LD_LIBRARY_PATH** is set to the location of the database server and Informix ESQL/C library files.

```
setenv INFORMIXDIR /ix/informix93
setenv INFORMIXSQLHOSTS /ix/sqlhosts.unified
setenv ONCONFIG s.miami
setenv INFORMIXSERVER miami

# setup to use J/Foundation
setenv JVPHOME /ix/informix93/extend/krakatoa
setenv CLASSPATH $JVPHOME/krakatoa.jar:$JVPHOME/jdbc.jar:/usr/java/lib/classes.zip

# Include jar paths for Java; include /usr/ccs/bin for C compiler:
setenv PATH $INFORMIXDIR/bin:$INFORMIXDIR/extend/krakatoa/krakatoa.jar:
   $INFORMIXDIR/extend/krakatoa/jdbc.jar:/usr/ccs/bin:$PATH

setenv LD_LIBRARY_PATH $INFORMIXDIR/lib:$INFORMIXDIR/lib/esql:/usr/lib
```

Related tasks:

⇨ Checking environment variables with the chkenv utility (Guide to SQL: Reference)

Related reference:

⇨ Database server files (Administrator's Reference)

⇨ informix.rc (UNIX) (Administrator's Reference)

Set environment variables on Windows

On Windows, the installation procedure prepares a file, `setenv.cmd`, that sets the environment variables to their correct values. The `setenv.cmd` file is stored in the `%INFORMIXDIR%` directory. You must run `setenv.cmd` before you can use any of the command-line utilities.

You can set environment variables or override environment variables that have been automatically set in the following places:

- On Windows, **System** > **Environment** > **User Variables**
- In a command-prompt session
- `%INFORMIXDIR%\`*dbservername.*`cmd` batch file

  Use this batch file to configure command-prompt utilities.

For more information, see the *IBM Informix Guide to SQL: Reference* and "Creating an onconfig file on Windows" on page 1-11.

Configure connectivity

The connectivity information allows a client application to connect to any IBM Informix database server on the network. The connectivity data for a particular database server includes the database server name, the type of connection that a client can use to connect to it, the host name of the computer or node on which the database server runs, and the service name by which it is known.

You must prepare the connectivity information even if the client application and the database server are on the same computer or node.

It is not necessary to specify all possible network connections in the `sqlhosts` file or registry before you start the database server. But to make a new connection available, you must take the database server offline and then bring it back to online mode again.

For detailed information about configuring connectivity, see Chapter 2, "Client/server communication," on page 2-1.

When you configure connectivity on UNIX, also consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTIONS configuration parameters. These parameters enable you to reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections. For more information, see the *IBM Informix Security Guide*.

The sqlhosts file on UNIX

On UNIX, the `sqlhosts` file contains connectivity information.

The default location of this file is $INFORMIXDIR/etc/sqlhosts. If you store the information in another location, you must set the **INFORMIXSQLHOSTS** environment variable.

If you set up several database servers to use distributed queries, use one of the following ways to store the `sqlhosts` information for all the databases:

- In one `sqlhosts` file, pointed to by **INFORMIXSQLHOSTS**
- In separate `sqlhosts` files in each database server directory

Network-configuration files

In addition to the `sqlhosts` files, Internet protocol network connections require entries in the /etc/hosts and /etc/services files.

Network-security files

IBM Informix database servers follow UNIX security requirements for making connections. Thus, the UNIX system administrator might be required to to make modifications to the /etc/passwd, /etc/hosts, ~/.rhosts, and other related files.

The network-configuration and network-security files are described in operating-system manuals.

Related concepts:

"The sqlhosts file and the SQLHOSTS registry key" on page 2-17

"Trusted-host information" on page 2-13

"Trusted-user information" on page 2-14

The sqlhosts registry on Windows

On Windows, the HKEY_LOCAL_MACHINE registry contains the sqlhosts information. The database server installation procedure prepares the registry information. You must not edit the HKEY_LOCAL_MACHINE registry.

Use **setnet32** to manage sqlhosts information. For information about **setnet32**, see the installation information in your client documentation. However, you cannot use **setnet32** to assign a database server to a database server group.

Configure the database server

You can customize the database server for your data-processing environment by setting configuration parameters. The configuration parameters for the database server are stored a configuration file.

When you create an Informix database server instance, a corresponding configuration file is created automatically with default values that are based on the config.std file. The new configuration file is called onconfig.*informixservername* and it is created in the %INFORMIXDIR%\etc\ directory by default. If you move your configuration file to another directory, set the ONCONFIG environment variable to the location of the file.

You can edit your configuration file to modify the configuration parameter values to improve performance and other characteristics of the instance. If you omit a parameter value in your configuration file, the database server either uses default values in the onconfig.std file or calculates values based on other parameter values.

The onconfig.std file is the template for creating onconfig.*informixservername* file. Do not modify or delete the onconfig.std file.

You can modify configuration parameter values by editing your onconfig file. The changes take effect after the next time the database server is shut down and restarted. However, you can change the value of some configuration parameters in other ways, when it is not convenient to restart the database server:

- Use the **onmode -wf** command to permanently update many configuration parameters while the database server is running.
- Use the **onmode -wm** command to update many configuration parameter values for the current session.

The online log identifies configuration parameters that were tuned dynamically.

You can use the IBM OpenAdmin Tool (OAT) for Informix to monitor your configuration.

Related concepts:

"Configuration file used during initialization" on page 3-3

Related tasks:

↪ Displaying the settings in the onconfig file (Administrator's Reference)

Related reference:

↪ onmode -wf, -wm: Dynamically change certain configuration parameters (Administrator's Reference)

↪ onconfig Portal: Configuration parameters by functional category (Administrator's Reference)

"Prepare your onconfig configuration file"

↪ AUTO_TUNE configuration parameter (Administrator's Reference)

↪ Database configuration parameters (Administrator's Reference)

Prepare your onconfig configuration file

The `onconfig.std` template file in the **etc** subdirectory of `INFORMIXDIR` contains initial values for many of the configuration parameters. You can copy the template and save the copy so that you can tailor the copy to your specific configuration.

The template files contain initial values for many of the configuration parameters.

Important: Do not modify or delete the template files. The database server provides these files as templates and not as functional configuration files.

The functional configuration file that you can change is your `onconfig` file (not the `onconfig.std` template file). You can use a text editor to change the configuration parameters in your `onconfig` file.

If you omit parameters in your copy of the `onconfig` file, the database server uses values in the `onconfig.std` file for the missing parameters when the server starts.

Tip: The **genoncfg** utility can expedite the process of customizing the configuration file to your hardware and to the anticipated usage of the database server if you do not want to work directly with all the `onconfig` file parameters.

Related concepts:

"Configure the database server" on page 1-9

"Configuration file used during initialization" on page 3-3

Related reference:

↪ The genoncfg Utility (Administrator's Reference)

Creating an onconfig file on UNIX

A new instance of the database server is created and initialized when you install the IBM Informix software for the first time. The installation script automatically creates the `onconfig.demo` file for you.

To prepare the `onconfig` file using a text editor:

1. Copy and rename the `$INFORMIXDIR/etc/onconfig.std` file and store it in the `etc` subdirectory.

2. Use a text editor to edit the onconfig configuration file.
3. Set the **ONCONFIG** environment variable to the name of your new onconfig file.
4. Initialize the database server if it is a new instance. Otherwise, shut down and restart the database server.

Creating an onconfig file on Windows

On Windows, a new instance of the database server is created and initialized when you install the IBM Informix software for the first time. You can also use the Instance Manager (**instmgr.exe**) to create a new database server instance. The Instance Manager automatically creates an onconfig file.

To prepare the onconfig file using the Instance Manager:

1. Use the Instance Manager to create a new database server instance.

   For details, see "Using the Instance Manager to create a new database server instance (Windows)."
2. Use a text editor to edit the onconfig configuration file.
3. Set your **ONCONFIG** environment variable to the name of your new onconfig file.
4. Shut down and restart the database server for the configuration changes to take effect.
   a. From the **Service** control application window, select the IBM Informix service and click **Stop**.
   b. Click **Start**.

To use **dbservername.cmd** to change the **ONCONFIG** environment variable.

1. Change the **ONCONFIG** environment variable for your session in the %INFORMIXDIR%\dbservername.cmd file.
2. Run **dbservername.cmd** for the changes to take effect.

View information about configuration parameters

You can view the contents of your configuration file with **onstat** commands or IBM OpenAdmin Tool (OAT) for Informix.

To view the contents of your configuration file, run the **onstat -c** command. If you change a configuration parameter while the server is running, and you do not shut down and restart the database server, the effective configuration differs from what the **onstat -c** option displays.

Related reference:

➡ onstat -c command: Print ONCONFIG file contents (Administrator's Reference)

➡ onstat -g cfg command: Print the current values of configuration parameters (Administrator's Reference)

Using the Instance Manager to create a new database server instance (Windows)

You can use the Instance Manager to create a new instance of the database server. The Instance Manager automatically creates an onconfig file for you. For more information about the Instance Manager and scheduling priority, see *IBM Informix Installation Guide for Windows*.

To use the Instance Manager:

1. Select **Server Instance Manager** from the Informix menu.

2. Click **Create New** to create a new instance of the database server and follow the instructions in the wizard.
3. Click **Delete Server** to delete a database server instance.

Using the Instance Manager to rename a database server instance (Windows)

You can also use the Instance Manager rename option, to change the name for a database server instance.

To rename a database server instance:

1. Shut down the database server instance that you want to rename.
2. Select **Server Instance Manager** from the Informix menu. The Instance Manager displays a list of available instances.
3. Select the instance to rename and click **Rename Server**.
4. In the dialog box that is displayed, enter your password and specify the new database server name.

After you rename the server instance, you can start the instance.

If an administrator changed the name of `onconfig` file or the `MSGLOG` file to a file with a custom name before the server instance was renamed, you must manually edit the file to change the name.

Configuring Java support

You can use IBM Informix Dynamic Server with J/Foundation to develop and run Java UDRs. Configure the database server without Java and then modify it to add Java support. Configuring the database server to support Java requires several additional steps.

To configure the database server to support Java user-defined routines

1. Create an sbspace to hold the Java JAR files.
2. Create the JVP properties file.
3. Add (or modify) the Java configuration parameters in the `onconfig` file.
4. Set environment variables.

For the setup instructions, see *J/Foundation Developer's Guide*.

Start and administer the database server

After you install and configure the database server, you must perform one or more of the following tasks:

- Prepare to connect to applications.
- Start the database server and initialize disk space.
- Create storage spaces.
- Set up your backup and restore system.
- Perform administrative tasks.

Starting the database server

Use the **oninit** utility to start the database server. You can start the database server in different modes. The default mode is online mode, which allows multiple users to connect to the database server.

Prerequisites:
- **UNIX, Linux, or Mac OS X:** You must be logged in as user **root** or **informix**.
- **Windows:** You must have appropriate permissions to start a Windows service.
- The disk space for the database server is initialized.

To start the database server:

UNIX, Linux, or Mac OS X: Run the **oninit** command. You can include options to the **oninit** command to customize start up.

Windows: Use one of the following methods. Include **oninit** options if you want to customize start up:
- In the **Services** control application, choose the database server service and type any **oninit** options that you want to use in the Startup parameters field. Then click **Start**.
- Use the **starts** command from the command line with the database server name and any **oninit** options you want to use: `starts dbservername -y`

Related concepts:

"Database server operating modes" on page 3-8

Related reference:

"Initializing disk space" on page 3-1

➦ The oninit utility (Administrator's Reference)

Preparing for automatic startup

Prepare the operating-system registry or scripts to automatically start and stop the database server. See "Preparing for automatic startup on Windows" or "Preparing the UNIX startup and shutdown scripts."

Preparing for automatic startup on Windows

Change the IBM Informix password in the **Service** control application when you change the IBM Informix password on Windows.

To start the database server automatically when Windows starts:
1. From the **Service** control application window, select the IBM Informix service and click **Startup**.
2. Select **Automatic** in the **Status Type** dialog box.
3. In the **Log On As** dialog box, select **This Account** and verify that `informix` is in the text box.

Preparing the UNIX startup and shutdown scripts

You can modify the UNIX startup script to initialize the database server automatically when your computer enters multiuser mode. You can also modify your UNIX shutdown script to shut down the database server in a controlled manner whenever UNIX shuts down.

ISA provides a sample UNIX script for startup and shutdown that you can customize in `$INFORMIXDIR/etc/ids-example.rc`.

Preparing the UNIX startup script:

You can prepare the UNIX startup script to start the database server.

To prepare the UNIX startup script, add UNIX and database server utility commands to the UNIX startup script so that the script performs the following steps.

To prepare the UNIX startup script:

1. Set the **INFORMIXDIR** environment variable to the full path name of the directory in which the database server is installed.
2. Set the **PATH** environment variable to include the $INFORMIXDIR/bin directory.
3. Set the **ONCONFIG** environment variable to the appropriate configuration file.
4. Set the **INFORMIXSERVER** environment variable so that the **sysmaster** database can be updated (or created, if necessary).
5. Run **oninit**, which starts the database server and leaves it in online mode.

   The **oninit** utility has a **-w** option that forces the server to wait until it successfully initializes before it returns to the shell prompt with a return code of 0. For information about the **oninit** utility, see The oninit Utility.

   If you plan to initialize multiple versions of the database server (multiple residency), you must reset **ONCONFIG** and **INFORMIXSERVER** and rerun **oninit** for each instance of the database server.
6. If you are using IBM Informix Storage Manager (ISM) (ISM) for managing database server backups, you must start the ISM server on each node.

   For information about how to start the ISM server, see your *IBM Informix Installation Guide*.

If different versions of the database server are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each different version.

Related reference:

⬐ Database configuration parameters (Administrator's Reference)

Preparing the UNIX shutdown script:

You can prepare to shut down the database server in a controlled manner whenever UNIX shuts down.

To shut down the database server in a controlled manner whenever UNIX shuts down, add UNIX and database server utility commands to the UNIX shutdown script so that the script performs the following steps.

To prepare the UNIX shutdown script:

1. Set the **INFORMIXDIR** environment variable to the full path name of the directory in which the database server is installed.
2. Set the **PATH** environment variable to include the $INFORMIXDIR/bin directory.
3. Set the **ONCONFIG** environment variable to the appropriate configuration file.
4. Run **onmode -ky**, which initiates an immediate shutdown and takes the database server offline.

   If you are running multiple versions of the database server (multiple residency), you must reset **ONCONFIG** and rerun **onmode -ky** for each instance.

If different versions of the database server are installed in different directories, you must reset **INFORMIXDIR** and repeat the preceding steps for each version.

In the UNIX shutdown script, the database server shutdown commands run after all client applications have completed their transactions and exited.

Prepare to connect to applications

When the database server is online, you can connect client applications and begin to create databases. Before you can access information in a database, the client application must connect to the database server environment. To connect to and disconnect from a database server, you can issue SQL statements from the following client programs

- DB-Access
- SQL Editor
- IBM Informix ESQL/C
- IBM Informix ODBC Driver
- IBM Informix JDBC Driver

For information about creating databases, see *IBM Informix Database Design and Implementation Guide* and *IBM Informix Guide to SQL: Tutorial*. For information about how to use client applications, see the *IBM Informix DB-Access User's Guide*, *IBM Informix ESQL/C Programmer's Manual*, *IBM Informix ODBC Driver Programmer's Manual*, or *IBM Informix JDBC Driver Programmer's Guide*.

Create storage spaces and chunks

You are responsible for planning and implementing the configuration of storage spaces and chunks. The way you distribute the data on disks affects the performance of the database server.

A *chunk* is the same as a logical volume, logical unit, or regular file that has been assigned to the database server. The maximum size of an individual chunk is 4 terabytes.

You can have up to 32766 chunks in an instance. You can put all those chunks in one storage space, or spread them among multiple storage spaces.

A logical *storage space* is composed of one or more chunks.

Tip: To take advantage of the large limit of 4 terabytes per chunk, assign a single chunk per disk drive. This way of distributing data increases performance.

After the database server is initialized, you can create storage spaces such as dbspaces, blobspaces, or sbspaces. Use the **onspaces** utility to create storage spaces and chunks.

After the database server is initialized, you can create storage spaces such as dbspaces, blobspaces, or sbspaces. Use the **onspaces** utility or ISA to create storage spaces and chunks.

You must create an sbspace if you are using the following functions:
- J/Foundation (to hold Java JAR files)
- Enterprise Replication (to hold spooled row data)
- Smart large objects (BLOB and CLOB data types)
- Multirepresentational data types (such as DataBlades or HTML data types)

For a description of storage spaces and other physical units such as tblspaces and extents, see Chapter 8, "Data storage," on page 8-1. For an explanation of the allocation and management of storage spaces, see Chapter 9, "Manage disk space," on page 9-1.

For more information, see the *IBM Informix Enterprise Replication Guide* and *J/Foundation Developer's Guide*.

Support for large chunks

If you just converted from a version of IBM Informix that is before Version 9.4, chunks that are greater than 2 gigabytes are not yet enabled. To support large chunks and large offsets to the maximum size of 4 terabytes and more than 2047 chunks, run **onmode -BC 1**.

You can test your data in the **onmode -BC 1** mode. When you are satisfied that your data has converted correctly, then you can run **onmode -BC 2** and thereby put the server in large -chunk-only mode.

After running **onmode -BC 2**, reversion is no longer supported. After support for large chunks is enabled, it cannot be disabled.

Set up your backup system and storage

You can use either the ON-Bar utility or the **ontape** utility to back up and restore your data.

Decide which utility to use and then prepare to back up your data:
- If you use **ontape** as your backup tool, you must set up storage devices (tape drives) before you can back up and restore data. The **ontape** utility does not require a storage manager.
- If you use ON-Bar as your backup tool, you must set up a storage manager and storage devices before you can back up and restore data.

  ON-Bar is packaged with IBM Informix Storage Manager (ISM) (ISM). The *storage manager* is an application that manages the storage devices and media that contain backups. The storage manager handles all media labeling, mount requests, and storage volumes. ISM can back up data to as many as four storage devices at a time. ISM stores data on simple tape drives, optical disk devices, and file systems. You can purchase a storage manager from another vendor if you want more sophisticated storage devices, backups to more than four storage devices at a time, or backups over a network. For information about configuring and managing storage devices and media, see the *IBM Informix Storage Manager Administrator's Guide* or to your vendor-acquired storage manager documentation.

For information about setting up and using ON-Bar or **ontape**, see the *IBM Informix Backup and Restore Guide*.

Automatically terminating idle connections

You can automatically terminate sessions with clients that have been idle for a specified time by enabling the **idle_user_timeout** Scheduler task.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To enable the **idle_user_timeout** task, run the following statement:
```
UPDATE ph_task
   SET tk_enable = 't'
   WHERE tk_name = 'idle_user_timeout';
```

By default, the **idle_user_timeout** task terminates user sessions that are idle for longer than 60 minutes. Sessions owned by user **informix** are not terminated. The **idle_user_timeout** task starts checking for idle sessions after two hours, which is the default frequency for the task.

Tip: When the system time changes on the database server computer, the amount of time user sessions have been idle is no longer accurate. For example, if a user session last did work at 3:14 PM and at 3:15 PM the system clock is moved forward by one hour, then to the database server, the user session has been idle for over an hour.

To change the idle timeout period, update the frequency of running the task and the value of the threshold. The shortest idle timeout period allowed is 5 minutes. For example, to change the timeout period to 5 minutes, run the following statements:

```
UPDATE ph_task
  SET tk_frequency = INTERVAL (5) MINUTE TO MINUTE
  WHERE tk_name = 'idle_user_timeout';

UPDATE ph_threshold
    SET value = '5'
    WHERE task_name = 'idle_user_timeout';
```

Configure session properties

You can change the properties of a database server session at connection or access time without changing the application that the session runs. This is useful if you cannot modify the source code of an application to set environment options or environment variables or to include session-related SQL statements, for example, because the SQL statements contain vendor-acquired code.

To change the properties of a session, design custom **sysdbopen( )** and **sysdbclose( )** procedures for various databases to support the applications of specific users or the PUBLIC group. The **sysdbopen( )** and **sysdbclose( )** procedures can contain a sequence of SET, SET ENVIRONMENT, SQL, or SPL statements that the database server executes for the user or the PUBLIC group when the database opens or closes.

For example, for *user1*, you can define procedures that contain SET PDQPRIORITY, SET ISOLATION LEVEL, SET LOCK MODE, SET ROLE, or SET EXPLAIN ON statements that execute whenever *user1* opens the database with a DATABASE or CONNECT TO statement.

Any settings of the session environment variables PDQPRIORITY and OPTCOMPIND that are specified by SET ENVIRONMENT statements within **sysdbopen( )** procedures persist for the duration of the session. SET PDQPRIORITY and SET ENVIRONMENT OPTCOMPIND statements, which are not persistent for regular procedures, are persistent when **sysdbopen( )** procedures contain them.

The *user*.**sysdbclose( )** procedure runs when the user who is the owner of the procedure disconnects from the database (or else when **PUBLIC.sysdbclose( )** runs, if it exists and no **sysdbclose( )** procedure is owned by the current user).

In custom **sysdbopen( )** and **sysdbclose( )** procedures, IBM Informix does not ignore the name of the owner of a UDR when a routine is invoked in a database that is not ANSI-compliant.

For more information, see "Configuring session properties" and the *IBM Informix Guide to SQL: Syntax.*

Configuring session properties

Only a DBA or user **informix** can create or alter **sysdbopen( )** or **sysdbclose( )** in the ALTER PROCEDURE, ALTER ROUTINE, CREATE PROCEDURE, CREATE PROCEDURE FROM, CREATE ROUTINE FROM, DROP PROCEDURE, or DROP ROUTINE statements of SQL.

You can set up **sysdbopen( )** procedures that change the properties of a session at connection or access time without changing the application that the session runs. This is useful if you cannot modify the source code of an application to set environment options or environment variables or to include session-related SQL statements, for example, because the SQL statements contain vendor-acquired code.

To set up a **sysdbopen()** and **sysdbclose()** procedure to configure session properties:

1. Set the **IFX_NODBPROC** environment variable to any value, including 0, to cause the database server to bypass and prevent the execution of the **sysdbopen( )** or **sysdbclose( )** procedure.
2. Write the CREATE PROCEDURE or CREATE PROCEDURE FROM statement to define the procedure for a particular user or the PUBLIC group.
3. Test the procedure, for example, by using **sysdbclose( )** in an EXECUTE PROCEDURE statement.
4. Unset the **IFX_NODBPROC** environment variable to enable the database server to run the **sysdbopen( )** or **sysdbclose( )** procedure.

Examples

The following procedure sets the role and the PDQ priority for a specific user:

```
create procedure oltp_user.sysdbopen()
    set role to oltp;
    set pdqpriority 5;
end procedure;
```

The following procedure sets the role and the PDQ priority for the PUBLIC group:

```
create procedure public.sysdbopen()
    set role to others;
    set pdqpriority 1;
end procedure
```

For more information, see information about **sysdbopen()** and **sysdbclose()** in the *IBM Informix Guide to SQL: Syntax*

Perform routine administrative tasks

Depending on the requirements of your organization, you might be responsible for performing the periodic tasks described in the following paragraphs. Not all of these tasks are appropriate for every installation. For example, if your database server is available 24 hours a day, 7 days a week, you might not bring the database server to offline mode, so database server operating mode changes would not be a routine task.

Change database server modes

The database server administrator is responsible for starting up and shutting down the database server by changing the mode. "Database server operating modes" on page 3-8 explains how to change database server modes.

Back up data and logical-log files

To ensure that you can recover your databases in the event of a failure, make frequent backups of your storage spaces and logical logs. You also can verify ON-Bar backups with the **archecker** utility.

How often you back up the storage spaces depends on how frequently the data is updated and how critical it is. A backup schedule might include a complete (level-0) backup once a week, incremental (level-1) backups daily, and level-2 backups hourly. You also must perform a level-0 backup after performing administrative tasks such as adding a dbspace, deleting a logical-log file, or enabling mirroring.

Back up each logical-log file as soon as it fills. You can back up these files manually or automatically. For information about using ON-Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

Monitor activity

The IBM Informix database server design lets you monitor every aspect of the database server. "Monitor database server activity" on page 1-22 provides descriptions of the available information, instructions for obtaining information, and suggestions for its use.

Check for consistency

Perform occasional checks for data consistency. For a description of these tasks, see Chapter 19, "Consistency checking," on page 19-1.

Perform additional administrative tasks

These topics cover various administrative tasks that you would perform on a production database server.

Disk mirroring

When you use disk mirroring, the database server writes data to two locations. Mirroring eliminates data loss due to storage device failures. If mirrored data becomes unavailable for any reason, the mirror of the data is available immediately and transparently to users. For information about mirroring, see Chapter 17, "Mirroring," on page 17-1. For instructions on tasks related to mirroring, see Chapter 18, "Using mirroring," on page 18-1.

Important: Mirror critical dbspaces that contain logical-log files, the physical log, and the root dbspace.

Manage database-logging status

You can specify whether each database uses transaction logging by default, whether the default logging mode for databases is buffered or unbuffered, and whether the logging mode is ANSI-compliant.

You can create the following table types in a logging database:

- STANDARD
- TEMP
- RAW

For more information, see "Temporary tables" on page 8-27 and Chapter 11, "Logging," on page 11-1. For information about how to change logging options, see Chapter 12, "Manage the database-logging mode," on page 12-1.

Manage the logical log

The database server contains several files called *logical logs* that record data transactions and administrative information such as checkpoint records and additions and deletions of chunks.

Typical logical-log administration tasks include backing up logical-log files, adding, freeing, and resizing logical-log files, and specifying high-watermarks.

The database server dynamically allocates logical-log files while online to prevent long transactions from hanging the database server.

For more information, see Chapter 13, "Logical log," on page 13-1. For instructions on creating and modifying the logical-log configuration, see Chapter 14, "Manage logical-log files," on page 14-1. For information about backing up the logical log, see the *IBM Informix Backup and Restore Guide*.

Manage the physical log

You can change the size and location of the physical log. For more information about the physical log, see Chapter 15, "Physical logging, checkpoints, and fast recovery," on page 15-1, and Chapter 16, "Manage the physical log," on page 16-1.

When the database server starts, it checks whether the physical log is empty, because that implies that it shut down in a controlled fashion. If the physical log is not empty, the database server automatically performs an operation called *fast recovery*. Fast recovery automatically restores databases to a state of physical and logical consistency after a system failure that might have left one or more transactions uncommitted.

Manage shared memory

Managing shared memory includes the following tasks:
- Changing the size or number of buffers (by changing the size of the logical-log or physical-log buffer, or changing the number of buffers in the shared-memory buffer pool)
- Changing shared-memory parameter values, if necessary
- Changing forced residency (on or off, temporarily or for this session)
- Tuning checkpoint intervals
- Adding segments to the virtual portion of shared memory
- Using the SQL statement cache to reduce memory usage and preparation time for queries

For information about how the database server uses shared memory, see Chapter 6, "Shared memory," on page 6-1. For additional information, see Chapter 7, "Manage shared memory," on page 7-1.

Manage virtual processors

The configuration and management of virtual processors (VPs) has a direct affect on the performance of a database server. The optimal number and mix of VPs for your database server depends on your hardware and on the types of applications that your database server supports.

For an explanation of virtual processors, see Chapter 4, "Virtual processors and threads," on page 4-1. For additional information, see Chapter 5, "Manage virtual processors," on page 5-1.

Manage parallel database query

You can control the resources that the database uses to perform decision-support queries in parallel. You must balance the requirements of decision-support queries against those of online transaction processing (OLTP) queries. The resources that you must consider include shared memory, threads, temporary table space, and scan bandwidth. For information about parallel database query (PDQ) and how fragmentation affects the performance of PDQ, see your *IBM Informix Performance Guide*.

Data replication

Data replication is the process of representing database objects at more than one distinct site. Data replication configurations consist of a primary server and one or more secondary servers, such as an HDR secondary server, one or more SD secondary servers, and one or more RS secondary servers. In addition, IBM Informix supports IBM Informix Enterprise Replication (ER). You can combine data replication and Enterprise Replication on the same database server. For more information, see the *IBM Informix Enterprise Replication Guide*.

Data replication environments

A primary server coupled with an HDR secondary server supports synchronous replication of an entire database to the secondary database server, providing a hot standby in case of a catastrophic computer failure.

A primary server coupled with an SD secondary server provides read-only access to data on a disk array shared with the primary server. A primary server coupled with an RS secondary server supports asynchronous replication of the database on the primary server. A primary server coupled with any combination of HDR secondary, RS secondary, and SD secondary servers can exist in a data replication environment.

Related concepts:

Part 5, "High availability and scalability"

Chapter 21, "High-availability cluster configuration," on page 21-1

"Remote standalone secondary servers" on page 21-10

"Shared disk secondary servers" on page 21-18

Enterprise Replication

Enterprise Replication supports asynchronous data replication over geographically distributed database servers and you can use it to replicate both entire databases and subsets of databases and tables. Enterprise Replication offers limited support of user-defined data types. For detailed information about Enterprise Replication, see the *IBM Informix Enterprise Replication Guide*.

Auditing

If you intend to use database server auditing, you must specify where audit records are stored, how to handle error conditions, and so on. You also might want to change how users are audited if you suspect that they are abusing their access privileges. For information about tasks related to auditing, see the *IBM Informix Security Guide*.

Distributed queries

You can use the database server to query and update more than one database across multiple database servers of the same database server instance (cross-database distributed queries) and across multiple server instances (cross-server distributed queries). This type of query is called a *distributed query*. This term is not restricted to SELECT statements, and often refers more generally to any DML operation or execution of a routine that returns or references objects outside the local database. In cross-server distributed queries, the participating database servers can be located on a single host computer, on the same network, or on a gateway.

For more information about distributed queries, see the *IBM Informix Database Design and Implementation Guide* and *IBM Informix Guide to SQL: Tutorial*.

For information about using the Secure Sockets Layer (SSL) protocol for connections between servers involved in distributed queries, see the "Secure Sockets Layer Communication Protocol" topic of the *IBM Informix Security Guide*.

Global transactions

A *global transaction* is a transaction that involves more than one database server. IBM Informix database servers support two types of global transactions:
* TP/XA with a transaction manager
* Two-phase commit

IBM Informix uses a two-phase commit protocol to ensure that distributed queries are uniformly committed or rolled back across multiple database servers. For more information, see Chapter 25, "Multiphase commit protocols," on page 25-1.

Transaction manager

Transaction managers manage terminals and data recovery. The TP/XA library enables communication between a vendor-acquired transaction manager and IBM Informix databases in an X/Open environment. For more information, see"Transaction managers" on page 25-1.

Monitor database server activity

You can use a number of tools to monitor database server activity. Most of the tools are available for use on UNIX and Windows platforms. A few are available only on UNIX, as the table in this topic shows.

You can gather information about database server activity from the following sources.

Source of Information	UNIX	Windows
Event alarms	X	X
Message log	X	X

Source of Information	UNIX	Windows
ON-Monitor	X	
oncheck utility	X	X
onperf utility	X	
onstat utility	X	X
SMI tables	X	X
System console	X	X
Windows Event Viewer		X
Windows Performance Logs and Alerts		X

The following topics explain each of these sources.

Event alarms

To report situations that require your immediate attention, the database server uses the event-alarm feature. To use the event-alarm feature, set the ALARMPROGRAM configuration parameter to the full path name of an executable file that performs the necessary administrative actions.

For more information, see the appendix on event alarms and the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

IBM Informix Server Administrator (ISA)

ISA is a browser-based tool that provides system administration for the entire range of IBM Informix database servers. ISA provides access to almost all Informix database server command-line functions.

For more information about ISA, the ISA online help and screen instructions.

Message log

The database server *message log* is an operating-system file. The messages contained in the database server message log do not usually require immediate action.

To report situations that require your immediate attention, the database server uses event alarms. For more information, see "Event alarms."

Specify the destination for message-log messages

To specify the message-log path name, set the MSGPATH configuration parameter. The changes to MSGPATH take effect after you shut down and restart the database server.

Related reference:

▶ MSGPATH configuration parameter (Administrator's Reference)

Monitor the message log

You must monitor the message log once or twice a day to ensure that processing is proceeding normally and that events are being logged as expected. Use the **onstat -m** command to obtain the name of the message log and the 20 most recent entries. Use a text editor to read the complete message log. Use an operating-system command (such as the UNIX command **tail -f**) to see the messages as they occur.

Monitor the message-log size, because the database server appends new entries to this file. Edit the log as necessary, or back it up to tape and delete it.

If the database server experiences a failure, the message log serves as an audit trail for retracing the events that develop later into an unanticipated problem. Often the database server provides the exact nature of the problem and the suggested corrective action in the message log.

You can read the database server message log for a minute-by-minute account of database server processing in order to catch events before a problem develops. However, you are not required to perform this kind of monitoring.

For more information, see *IBM Informix Error Messages*.

Related reference:

 The finderr utility (Administrator's Reference)

 Message Log (Administrator's Reference)

ON-Monitor (UNIX)

ON-Monitor provides a simple way to monitor many aspects of the database server. Most of the monitoring functions are available under the **Status** menu.

Related concepts:

 The ON-Monitor Utility (Administrator's Reference)

The oncheck utility

The **oncheck** utility displays information about the database disk configuration and usage, such as the number of pages used for a table, the contents of the reserved pages, and the number of extents in a table.

Related concepts:

 The oncheck Utility (Administrator's Reference)

The onperf tool (UNIX)

The database server includes a graphical monitoring tool called **onperf**. This tool can monitor most of the metrics that **onstat** provides. It provides the following advantages over **onstat**:

- It displays the values of the metrics graphically in real time.
- It lets you choose which metrics to monitor.
- It saves recent-history metrics data to a buffer in memory. This data is available if you want to analyze a recent trend.
- It can save performance data to a file.

For more information about the **onperf** tool, see your *IBM Informix Performance Guide*.

The onstat utility

The **onstat** utility provides a way to monitor database server shared memory from the command line. The **onstat** utility reads data from shared memory and reports statistics that are accurate for the instant during which the command executes. That is, **onstat** provides information that changes dynamically during processing, including changes in buffers, locks, indexes, and users.

SMI tables

The system-monitoring interface (SMI) tables are special tables managed by the database server that contain dynamic information about the state of the database server. You can use SELECT statements on them to determine almost anything you might want to know about your database server. For a description of the SMI tables, see the topics on the **sysmaster** database in the *IBM Informix Administrator's Reference*.

Related concepts:

➡ The sysmaster Database (Administrator's Reference)

System console

The database server sends messages that are useful to the database server administrator by way of the *system console*. To specify the destination path name of console messages, set the CONSOLE configuration parameter.

The changes to CONSOLE take effect after you shut down and restart the database server.

Windows only: A database server system administrator can log-in to the console from any node to perform system management and monitoring tasks.

Related reference:

➡ CONSOLE configuration parameter (Administrator's Reference)

UNIX operating-system tools

The database server relies on the operating system of the host computer to provide access to system resources such as the CPU, memory, and various unbuffered disk I/O interfaces and files. Each operating system has its own set of utilities for reporting how system resources are used. Different operating-systems might have monitoring utilities with the same name but different options and informational displays.

The following table shows typical UNIX operating-system resource-monitor utilities. For information about how to monitor your operating-system resources, consult your system administration guide.

UNIX utility	Description
vmstat	Displays virtual-memory statistics
iostat	Displays I/O utilization statistics
sar	Displays various resource statistics
ps	Displays active process information
cron	Captures the status of system resources by a system scheduler that runs a command or program at regular intervals. You also can use other scheduling tools that are available with your operating system.

Windows Event Viewer

The Event Viewer shows informational, warning, and error messages for the operating system, other applications, and the database server.

To display database server messages on Windows:

1. Choose **Administrative Tools** > **Event Viewer**.

2. Choose **Security**.

3. Double-click any event for a more detailed message.

Windows Performance Logs and Alerts

The Windows Performance Logs and Alerts (**perfmon.exe**) monitors resources such as processor, memory, cache, threads, and processes. The Performance Logs and Alerts also provides charts, alerts, report capabilities, and the ability to save information to log files for later analysis.

To display the Performance Logs and Alerts on Windows, choose **Administrative Tools** > **Performance**.

Windows utilities

The following IBM Informix utilities simplify administration of the database server on Windows.

Windows utility	Description and usage
ixpasswd.exe	Changes the logon password for all services which log on as user **informix**. You can change the password interactively or on the command line using the **-y** option. Using this utility, you are not required to manually change the password for each service whenever you change the **informix** password.
	If you are logged on locally and run **ixpasswd**, it changes the password for services that log on as the local **informix** user. If you are logged on *domain* and run **ixpasswd**, it changes the password for services that log on as *domain***informix**
	Usage: ixpasswd [-y *new_password*]
ixsu.exe	Opens a command-line window that runs as the specified user. The user is a local user unless you specify a domain name. If you do not specify a user name, the default user is **informix**. You no longer are required to log off as the current user and log on as **informix** to perform DBA tasks that must be run as **informix**/
	The **ixsu** utility requires Advanced User Rights:
	• Act as part of the operating system
	• Increase quotas
	• Replace a process-level token
	To configure Advanced User Rights on Windows NT, select **User Manager** > **Policies** > **User Rights** and check the Advanced User Rights option. If you change the Advanced User Rights for the current user, you must log off and log back on for the new rights to take effect.
	Usage: ixsu [[*domain*\]*username*]
ntchname.exe	Changes the registry entries for IBM Informix from the old host name to the new host name. Run **ntchname** after you change the host name. This utility does not change user environment variables.
	After you run **ntchname**, edit the `%INFORMIXDIR%\%INFORMIXSERVER%.cmd` file and change the `INFORMIXSQLHOSTS` entry to the new host name.
	Usage: ntchname *old_name new_name*

The OpenAdmin Tool (OAT) for Informix

You can use a PHP-based web browser administration tool, the IBM OpenAdmin Tool (OAT) for Informix, to administer multiple database server instances from a single location. The graphical user interface eases many common administration tasks, and when possible provides recommendations for tuning your system. You can complete many of the administration tasks in the following list, using the OAT.

- System health monitoring through a graphical dashboard, alerts, and online message logs
  - "Controlling the size of the command_history table" on page 28-3
  - "Viewing SQL administration API history" on page 28-2
  - "Message log" on page 1-23
- Database server administration
  - "Set permissions, ownership, and group" on page 1-4
  - "Prepare your onconfig configuration file" on page 1-10
  - "Monitor database server activity" on page 1-22
  - Automate UPDATE STATISTICS
  - "Manage virtual processors" on page 1-21
  - Monitoring high availability solutions that include HDR, shared disk secondary servers, and remote standalone secondary servers: Part 5, "High availability and scalability"
- Space administration
  - Chapter 9, "Manage disk space," on page 9-1
  - "Manage dbspaces" on page 9-7
  - "Create storage spaces and chunks" on page 1-15
  - "Monitor checkpoint information" on page 16-3
  - "Force a checkpoint" on page 16-4
  - "Manage the logical log" on page 1-20
  - "Manage the physical log" on page 1-20
  - "Monitor physical and logical-logging activity" on page 16-2
  - "Configure disk space" on page 1-3
- Performance gathering and analysis such as query drill-down (Chapter 29, "Query drill-down," on page 29-1) ; session monitoring with information about SQL, locks, threads, and memory used by each session; and other methods as described in *IBM Informix Performance Guide*.
- Create, modify, and delete tasks for Chapter 27, "The Scheduler," on page 27-1
- "Display databases" on page 9-43, "Tables" on page 8-22, and "Extents" on page 8-8

You can easily plug your own extensions into the OAT to create the functionality you require.

Related concepts:

⯈ Installing the OpenAdmin Tool for Informix with the Client SDK (Client Products Installation Guide)

Chapter 2. Client/server communication

These topics explain the concepts and terms that you must understand in order to configure client/server communication.

Related tasks:

Client/server architecture

IBM Informix products conform to the *client/server* software-design model. Application or clients can be on the computer housing the database server or on a different computer. Client applications issue requests for services and data from the database server. The database server responds by providing the services and data that the client requested.

You use a *network protocol* together with a *network programming interface* to connect and transfer data between the client and the database server.

Network protocol

A *network protocol* is a set of rules that govern how data is transferred between applications and, in this context, between a client and a database server.

The rules of a protocol are implemented in a *network driver*. A network driver contains the code that formats the data when it is sent from client to database server and from database server to client.

Clients and database servers gain access to a network driver by way of a *network programming interface*. A network programming interface contains system calls or library routines that provide access to network-communications facilities. An example of a network programming interface for UNIX is TLI (Transport Layer Interface). An example of a network programming interface for Windows is WINSOCK (sockets programming interface).

The power of a network protocol lies in its ability to enable client/server communication even though the client and database server are on different computers with different architectures and operating systems.

You can configure the database server to support more than one protocol, but consider this option only if some clients use TCP/IP.

On UNIX, specify the `nettype` in the `sqlhosts` file to control the protocol the database server uses. On Windows, set the **PROTOCOL** field in the SQLHOSTS registry key.

Network programming interface

A *network programming interface* is an application programming interface (API) that contains a set of communications routines or system calls. An application can call these routines to communicate with another application that is on the same or on different computers. In the context of this explanation, the client and the database server are the applications that call the routines in the TLI or sockets API. Clients and database servers both use network programming interfaces to send and receive the data according to a communications protocol.

Both client and database server environments must be configured with the same protocol if client/server communication is to succeed. However, some network protocols can be accessed through more than one network programming interface. For example, TCP/IP can be accessed through either TLI or sockets, depending on which programming interface is available on the operating-system platform.

Windows network domain

Windows network technology enables you to create network *domains*. A domain is a group of connected Windows computers that share user account information and a security policy. A *domain controller* manages the user account information for all domain members.

The domain controller facilitates network administration. By managing one account list for all domain members, the domain controller relieves the network administrator of the requirement to synchronize the account lists on each of the domain computers. In other words, the network administrator who creates or changes a user account must update only the account list on the domain controller rather than the account lists on each of the computers in the domain.

To log in to a Windows database server, a user on another Windows computer must belong to either the same domain or a *trusted domain*. A trusted domain is one that establishes a *trust relationship* with another domain. In a trust relationship, user accounts are only in the trusted domain.

A user who attempts to log in to a Windows computer that is a member of a domain can do so either by using a local login and profile or a domain login and profile. However, if the user is listed as a trusted user or the computer from which the user attempts to log in is listed as a trusted host, the user can be granted login access without a profile.

Important: A client application can connect to the database server only if there is an account for the user ID in the Windows domain in which the database server runs. This rule also applies to trusted domains.

If you specify a user identifier but no domain name for a connection to a workstation that expects both a domain name and a user name (domain\user), the database server checks only the local workstation and the primary domain for the user account. If you explicitly specify a domain name, that domain is used to search for the user account. The attempted connection fails with error -951 if no matching domain\user account is found on the local workstation.

Use the CHECKALLDOMAINSFORUSER configuration parameter to configure how the database server searches for user names in a networked Windows environment.

Table 2-1. Locations Informix searches for user names specified either alone or with a domain name.

	Domain and user specified	User name only specified
CHECKALLDOMAINSFORUSER is unset	Searches in the specified domain only	Searches on the local host only
CHECKALLDOMAINSFORUSER=0	Searches in the specified domain only	Searches on the local host only
CHECKALLDOMAINSFORUSER=1	Searches in the specified domain only	Searches in all domains

Important: The database server's trusted client mechanism is unrelated to the trust relationship that you can establish between Windows domains. Therefore, even if a client connects from a trusted Windows domain, the user must have an account in the domain on which the database server is running.

Related reference:

➡ CHECKALLDOMAINSFORUSER configuration parameter (Administrator's Reference)

Database server connections

A client application establishes a connection to a database server with either the CONNECT or DATABASE SQL statement. For example, an application might contain the following CONNECT statement to connect to the database server named **my_server**:

```
CONNECT TO '@my_server'
```

Tip: The database server's internal communications facility is called Association Services Facility (ASF). If you see an error message that includes a reference to ASF, you have a problem with your connection.

Related reference:

"Network protocol" on page 2-1

➥ CONNECT statement (Guide to SQL: Syntax)

➥ DATABASE statement (Guide to SQL: Syntax)

Supporting multiplexed connections

Some applications connect multiple times to the same database server on behalf of one user. A *multiplexed connection* uses a single network connection between the database server and a client to handle multiple database connections from the client. Client applications can establish multiple connections to a database server to access more than one database on behalf of a single user. If the connections are not multiplexed, each database connection establishes a separate network connection to the database server. Each additional network connection uses additional computer memory and processor time, even for connections that are not active. Multiplexed connections enable the database server to create multiple database connections without using up the additional computer resources that are required for additional network connections.

To configure the database server to support multiplexed connections:
1. Define an alias using the DBSERVERALIASES configuration parameter. For example, specify:

   ```
   DBSERVERALIASES ifx_mux
   ```
2. Add an sqlhosts file entry for the alias using onsqlmux as the **nettype** entry. The **hostname** and **servicename**, must have entries, but the entries are ignored. Dashes (-) can be used as entries. For example:

   ```
   #dbservername   nettype     hostname    servicename   options
   ifx_mux         onsqlmux    -           -
   ```
3. Enable multiplexing for the selected connection types by specifying m=1 in the sqlhosts file or registry that the client uses for the database server connection. For example:

   ```
   #dbservername   nettype     hostname    servicename   options
   menlo           ontlitcp    valley      jfkl          m=1
   ```
4. On Windows platforms, you must also set the IFX_SESSION_MUX environment variable.

The following example shows both onconfig file and sqlhosts file entries.

onconfig file:

```
DBSERVERNAME web_tli
DBSERVERALIASES web_mux
```

sqlhosts file:

```
#dbservername   nettype     hostname    servicename   options
web_tli         ontlitcp    node5       svc5          m=1
web_mux         onsqlmux    -           -
```

You are not required to change the sqlhosts file or registry that the database server uses. The client program does not require any special SQL calls to enable connections multiplexing. Connection multiplexing is enabled automatically when the onconfig file and the sqlhosts file or SQLHOSTS registry key are configured appropriately and the database server starts.

Multiplexed connections do not support:

- Multithreaded client connections
- Shared-memory connections
- Connections to subordinate database servers (for distributed queries or data replication, for example)

If any of these conditions exist when an application attempts to establish a connection, the database server establishes a standard connection. The database server does not return an SQL error.

The Informix ESQL/C **sqlbreak()** function is not supported during a multiplexed connection.

Related concepts:

"The sqlhosts file and the SQLHOSTS registry key" on page 2-17

➦ Multiplexed connections and CPU utilization (Performance Guide)

Related reference:

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

➦ DBSERVERNAME configuration parameter (Administrator's Reference)

➦ DBSERVERALIASES configuration parameter (Administrator's Reference)

Connections that the database server supports

The database server supports the following types of connections with client application.

Connection type	Windows	UNIX	Local	Network
Sockets	X	X	X	X
TLI (TCP/IP)		X	X	X
Shared memory		X	X	
Secure Sockets Layer (SSL)	X	X		X
Stream pipe		X	X	
Named pipe	X		X	

Secure Sockets Layer (SSL) connections use encryption for data communication between two points over a network.

When configuring connectivity, consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTION configuration parameters. These parameters enable you to reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections.

UNIX only: On many UNIX platforms, the database server supports multiple network programming interfaces. The machine notes show the interface/protocol combinations that the database server supports for your operating system.

To set up a client connection:

1. Specify connectivity and connection configuration parameters in your `onconfig` file.
2. Set up appropriate entries in the connectivity files on your platform.

3. Specify connectivity environment variables in your UNIX start-up scripts or the local and domain-wide Windows registries.
4. Define a dbserver group for your database server in the `sqlhosts` file or registry.

Related concepts:

➞ Secure sockets layer protocol (Security Guide)

➞ Limiting denial-of-service flood attacks (Security Guide)

"The sqlhosts information" on page 2-20

Related reference:

➞ NETTYPE configuration parameter (Administrator's Reference)

"Network protocol" on page 2-1

➞ LISTEN_TIMEOUT configuration parameter (Administrator's Reference)

➞ MAX_INCOMPLETE_CONNECTIONS configuration parameter (Administrator's Reference)

"Connectivity files" on page 2-9

"ONCONFIG parameters related to connectivity" on page 2-40

"Environment variables for network connections" on page 2-45

"sqlhosts Connectivity information" on page 2-21

Local connections

A *local connection* is a connection between a client and the database server on the same computer. The following topics describe different types of local connections.

Shared-memory connections (UNIX)

A *shared-memory connection* uses an area of shared-memory as the channel through which the client and database server communicate with each other. A client cannot have more than one shared-memory connection to a database server.

The following figure illustrates a shared-memory connection.
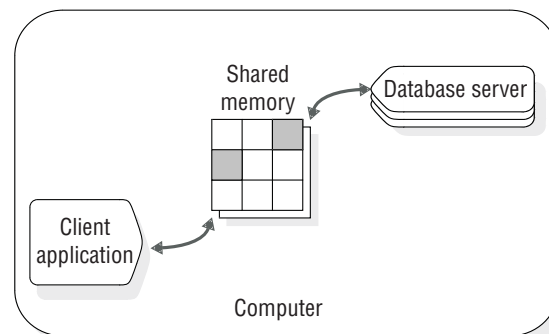
Figure 2-1. Client application and a database server communication through a shared-memory connection.

Shared memory provides fast access to a database server, but it poses some security risks. Errant or malicious applications might delete or view message buffers of their own or of other local users. Shared-memory communication is also vulnerable to programming errors if the client application performs explicit

memory addressing or over-indexes data arrays. Such errors do not affect the database server if you use network communication or stream pipes.

Example of a shared-memory connection

The following figure shows a shared-memory connection on the computer named **river**.
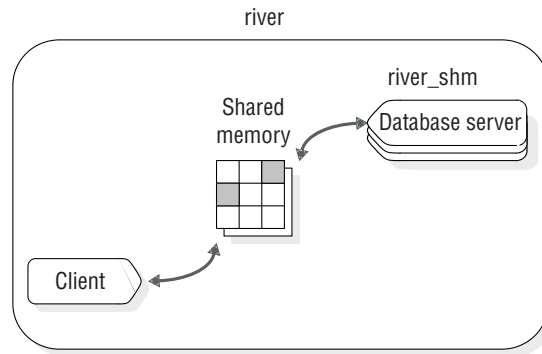
Figure 2-2. A shared-memory connection between a client application and a database server named river_shm.

The onconfig file for this installation includes the following line:

```
DBSERVERNAME river_shm
```

The sqlhosts file for this installation includes the following lines:

```
#dbservername   nettype   hostname   servicename options
 river_shm       onipcshm  river      rivershm
```

The client application connects to this database server using the following statement:

```
CONNECT TO '@river_shm'
```

For a shared-memory connection, no entries in network configuration files are required. Use arbitrary values for the hostname and servicename fields of the sqlhosts file or SQLHOSTS registry key.

Related concepts:

"Communications portion of shared memory (UNIX)" on page 6-19

"How a client attaches to the communications portion (UNIX)" on page 6-5

Related reference:

"Examples of client/server configurations" on page 2-51

Stream-pipe connections (UNIX and Linux)

A *stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other.

Stream-pipe connections have the following advantages:

* Unlike shared-memory connections, stream pipes do not pose the security risk of being overwritten or read by other programs that explicitly access the same portion of shared memory.
* Unlike shared-memory connections, stream-pipe connections allow distributed transactions between database servers that are on the same computer.

Stream-pipe connections have the following disadvantages:

- Stream-pipe connections might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all platforms.
- When you use shared memory or stream pipes for client/server communications, the **hostname** entry is ignored.

Related reference:

"sqlhosts Connectivity information" on page 2-21

Named-pipe connections (Windows)

Named pipes are application programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows. Named-pipe connections provide a high-level interface to network software by making transport-layer operations transparent. Named pipes store data in memory and retrieve it when requested, in a way that is similar to reading from and writing to a file system.

Local-loopback connections

A network connection between a client application and a database server on the same computer is called a *local-loopback* connection. The networking facilities used are the same as if the client application and the database server were on different computers. You can make a local-loopback connection provided your computer is equipped to process network transactions. Local-loopback connections are not as fast as shared-memory connections, but they do not pose the security risks of shared memory.

In a local-loopback connection, data seems to pass from the client application, out to the network, and then back in again to the database server. The internal connection processes send the information directly between the client and the database server and do not put the information out on the network.

An example of a local-loopback connection

The following figure shows a local-loopback connection that uses sockets and TCP/IP.
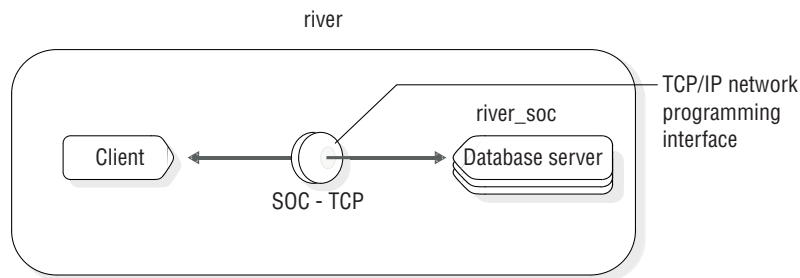
Figure 2-3. A local-loopback connection between a client and a database server named **river_soc** *on a computer named* **river***.*

The sqlhosts file for this installation includes the following lines:

```
#dbservername    nettype    hostname    servicename    options
 river_soc        onsoctcp   river       river01
```

If the network connection uses TLI instead of sockets, only the **nettype** entry in this example changes. In that case, the **nettype** entry is `ontlitcp` instead of `onsoctcp`.

The `onconfig` file for this installation includes the following lines:
```
DBSERVERNAME river_soc
```

This example assumes that an entry for **river** is in the `hosts` file and an entry for **riverol** is in the `services` file.

Related reference:

"Examples of client/server configurations" on page 2-51

Communication support services

Communication support services include connectivity-related services such as:

- **Authentication**, which is the process of verifying the identity of a user or an application. The most common form of authentication is to require the user to enter a name and password to obtain access to a computer or an application.
- **Message integrity**, which ensures that communication messages are intact and unaltered when they arrive at their destination.
- **Message confidentiality**, which protects messages from unauthorized viewing, usually through encryption and decryption, during transmission.

Communication support services can also include other processing such as data compression or traffic-based accounting.

The database server provides extra security-related communication support services through plug-in software modules called Communication Support Modules (CSM). The database server uses the default authentication policy when you do not specify a communications support module.

Related concepts:

"Network programming interface" on page 2-2

"Network security files" on page 2-12

➡ Communication support modules for data transmission encryption (Security Guide)

➡ Single sign-on authentication (Security Guide)

➡ Simple password encryption (Security Guide)

Connectivity files

The *connectivity files* contain the information that enables client/server communication and enable a database server to communicate with another database server. The connectivity configuration files can be divided into three groups:

- Network-configuration files
- Network security files
- The `sqlhosts` file or SQLHOSTS registry

Related tasks:

"Connections that the database server supports" on page 2-5

Network-configuration files

These topics identify and explain the use of network-configuration files on TCP/IP networks.

Related reference:

"Network protocol" on page 2-1

TCP/IP connectivity files

When you configure the database server to use the TCP/IP network protocol, you use information from the `hosts` and `services` files to prepare the `sqlhosts` information.

The `hosts` file requires a single entry for each network-controller card that connects a computer running an IBM Informix client/server product on the network. Each entry in the file contains the IP address (or ethernet card address) and host name. You can also include the host alias. Although the length of the host name is not limited in the `hosts` file, database server limits the host name to 256 bytes.

The following example has two entries.

```
#address        hostname  alias
98.555.43.21    odyssey
12.34.56.555    illiad    sales
```

The `services` file contains an entry for each service available through TCP/IP. Each entry is a single line that contains the following information:

- Service name

  IBM Informix products use this name to determine the port number and protocol for making client/server connections. The service name is limited to 128 bytes.

- Port number and connection protocol, separated by a forward slash

  The port number is the computer port, and the protocol for TCP/IP is `tcp`.

  The operating system imposes restrictions on the port number. User **informix** must use a port number equal to or greater than 1024. Only **root** users are allowed to use a port number lower than 1024.

- Host Aliases (optional)

The service name and port number are arbitrary. However, they must be unique within the context of the file and must be identical on all the computers running IBM Informix client/server products. The following example has one entry:

```
#servicename    port/protocol
server2         1526/tcp
```

This entry makes `server2` known as the service name for TCP port 1526. A database server can then use this port to service connection requests.

Important: For database servers that communicate with other database servers, you must define either a TCP/IP connection or an IPCSTR (interprocess communications stream pipe) connection for the DBSERVERNAME configuration parameter. You can also define at least one DBSERVERALIASES configuration parameter setting with the appropriate connection protocol for connectivity between the coordinator and the subordinate servers. For cross-server transactions,

each participating server to support a TCP/IP or an IPCSTR connection with the coordinator, even if both database server instances that are on the same workstation.

You typically include a separate NETTYPE parameter for each connection type that is associated with a dbserver name. You list dbserver name entries in the DBSERVERNAME and DBSERVERALIASES configuration parameters. You associate connection types with dbserver names through entries in the `sqlhosts` file or registry.

Related reference:

"sqlhosts Connectivity information" on page 2-21

TCP/IP connectivity files (UNIX):
On UNIX, the `hosts` and `services` files are in the `/etc` directory. The files must be present on each computer that runs an IBM Informix client/server product, or on the Network Information Service (NIS) server if your network uses NIS.

Warning: On systems that use NIS, the `hosts` and `services` files are maintained on the NIS server. The `hosts` and `services` files that are on your local computer might not be used and might not be up to date. To view the contents of the NIS files, enter the following commands on the command line:

```
ypcat hosts
ypcat services
```

TCP/IP connectivity files (Windows):
To prepare the SQLHOSTS registry key for the TCP/IP network protocol, use information from the `hosts` and `services` network-configuration files. These files are in the following locations:

- `%WINDIR%\system32\drivers\etc\hosts`
- `%WINDIR%\system32\drivers\etc\services`

As an alternative, configure TCP/IP to use the Domain Name Service (DNS) for host name resolutions.

The Dynamic Host Configuration Product (DHCP) dynamically assigns IP addresses from a pool of addresses instead of using IP addresses that are explicitly assigned to each workstation. If your system uses DHCP, Windows Internet Name Service (WINS) is required. DHCP is transparent to the database server.

Client and server actions when a TCP/IP connection is opened:
When a TCP/IP connection is opened, the following information is read on the client side:

- The **INFORMIXSERVER** environment variable.
- `hosts` file information (INFORMIXSQLHOSTS, $INFORMIXDIR/etc/sqlhosts, the registry entry on Windows NT) and `services` file information
- Other environment variables
- Resource files

The following information is read on the server side:

- The DBSERVERNAME configuration parameter
- The DBSERVERALIASES configuration parameter
- Server environment variables and configuration parameters, including any NETTYPE configuration parameter setting that manages TCP/IP connections.

Related reference:

➥ NETTYPE configuration parameter (Administrator's Reference)

Multiple TCP/IP ports

To take advantage of multiple ethernet cards:

- Make an entry in the `services` file for each port the database server uses, as in the following example:

```
#servicename    port/protocol    alias
soc1            21/tcp
soc2            22/tcp
```

  Each port of a single IP address must be unique. Separate ethernet cards can use unique or shared port numbers. You might want to use the same port number on ethernet cards connecting to the same database server. (In this scenario, the service name is the same.)

- Put one entry per ethernet card in the `hosts` file with a separate IP address, as in the following example:

```
#address           hostname    alias
192.147.104.19     svc8
192.147.104.20     svc81
```

- In the `onconfig` file, set DBSERVERNAME configuration parameter for one of the ethernet cards and the DBSERVERALIASES configuration parameter for the other ethernet card. The following lines show sample entries in the `onconfig` file:

```
DBSERVERNAME chicago1
DBSERVERALIASES chicago2
```

- In the `sqlhosts` file on UNIX or the SQLHOSTS registry key on Windows, make one entry for each ethernet card. That is, make an entry for the DBSERVERNAME and another entry for the DBSERVERALIASES.

```
#dbservername    nettype    hostname    servicename    options
chicago1         onsoctcp   svc8        soc1
chicago2         onsoctcp   svc81       soc2
```

After this configuration is in place, the application communicates through the ethernet card assigned to the dbserver name that the **INFORMIXSERVER** environment variable provides.

Related reference:

➥ INFORMIXSERVER environment variable (Guide to SQL: Reference)

Network security files

IBM Informix products follow standard security procedures that are governed by information contained in the network security files. For a client application to connect to a database server on a remote computer, the user of the client application must have a valid user ID on the remote computer.

Related concepts:

"Network programming interface" on page 2-2

"Communication support services" on page 2-9

Related reference:

➦ REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

➦ REMOTE_USERS_CFG configuration parameter (Administrator's Reference)

➦ S6_USE_REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

Trusted-host information

The `/etc/hosts.equiv` and `.rhosts` files provide the remote-authentication database for **rlogin**, **rsh**, **rcp**, and **rcmd**. The files specify remote hosts and users that are trusted. Trusted users are allowed to access the local system without supplying a password. The `hosts.equiv` file applies to the entire system. Individual users maintain their own `.rhosts` files in their home directories. On Windows, the `hosts.equiv` file is in the `\%WINDIR%\system32\drivers\etc` directory.

The database server can be configured to use the `/etc/hosts.equiv` or `.rhosts` files for remote authentication. To use trusted-host information for authentication, you should specify the `s=1` or `s=3` options in the `sqlhosts` file or the Windows SQLHOSTS registry key entry. If you do not specify an `s` option, `s=3` is the default.

You can configure the database server to use a different file for remote authentication. The REMOTE_SERVER_CFG configuration parameter can specify an alternative file to use instead of the `hosts.equiv` file. This alternative file is necessary in the following situations:

- You need different trusted hosts for the database server than those listed for the OS
- The security policies at your installation do not allow the use of `hosts.equiv`
- You are a user of a non-root server instance and need to control which hosts are trusted

Trusted-hosts file refers to either the `hosts.equiv` file or the file specified by the REMOTE_SERVER_CFG configuration parameter.

If a client application supplies an invalid account name and password, the database server rejects the connection even if the trusted-host information contains an entry for the client computer. Use trusted-hosts information only for client applications that do not supply a user account or password.

The `username` field is optional. Include the user name to limit the connection to the specified user.

To avoid an extra DNS lookup, specify the host name both with and without the domain name in the trusted-hosts file. For example, if the trusted host is named **argo** and it resides in the domain **example.com**, then the trusted-hosts file specify the following hosts:

```
#trustedhost     #username
argo             informix
argo.example.com informix
```

On some networks, the host name that a remote host uses to connect to a particular computer might not be the same as the host name that the computer

uses to refer to itself. For example, the network host with the fully qualified domain name (FQDN) **xyz.informix.com** might refer to itself with the local host name **viking**. If this situation occurs, the trusted-hosts information must specify both host-name formats, as in the following example:

```
#trustedhost
xyz.informix.com
viking
```

If you are using the system `hosts.equiv` file and you use the **rlogind** daemon, you can execute the following statement on the client computer to determine whether a client is trusted:

```
rlogin hostname
```

If you log-in successfully without receiving a password prompt, the client is a trusted computer. This method of determining if a client is trusted does not work when the REMOTE_SERVER_CFG configuration parameter specifies an alternative file name.

You can increase security for Enterprise Replication connection and High Availability connections by configuring a dedicated port in the `INFORMIXSQLHOSTS` file.

Related concepts:

Database Server Groups (Enterprise Replication Guide)

"Redirecting clients with the connectivity information" on page 24-8

Related tasks:

"Configuring secure connections for clusters" on page 21-5

Related reference:

"The sqlhosts file on UNIX" on page 1-8

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

INFORMIXSERVER environment variable (Guide to SQL: Reference)

DBPATH environment variable (Guide to SQL: Reference)

REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

S6_USE_REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

Trusted-user information

In their `.rhosts` file, a user can list hosts from which they can connect as a trusted user. The `.rhosts` file is located in the user's home directory on the computer housing the database server. To enable the trusted user authentication, specify `s=2` or `s=3` in the options in the `sqlhosts` file or the Windows SQLHOSTS registry key entry. If you do not specify an `s` option, `s=3` is the default.

There may be reasons why a user's `.rhosts` file cannot be used. For example, a non-root installation might not have read access to a specific user's `.rhosts` file. You can specify an alternate filename by setting the REMOTE_USERS_CFG configuration parameter. If you set this parameter, the database server only has a single trusted-user file for all users.

Each line of the `.rhosts` file is a host from which the user can connect. You must specify server names both with and without domain names to avoid performing an extra DNS lookup. For example:

```
#trustedusers
xxx.example.com
xxx

yyy.example.com
yyy
```

The file specified by the REMOTE_USERS_CFG configuration parameter must be a combination of individual `.rhosts` files. Each single-line entry of the file has the following format:

hostname username

For example, suppose the following two `.rhosts` files existed for users **John** and **Fred**:

```
~john/.rhosts
#trustedhosts
xxx.example.com
xxx

yyy.example.com
yyy

~fred/.rhosts
#trustedhosts
xxx.example.com
xxx

zzz.example.com
zzz
```

John does not trust **zzz.example.com** or **zzz**, and **Fred** does not trust **yyy.example.com** or **yyy**.

The `.rhosts` files could be combined into a single file with the following format:

```
#trustedhost      username
xxx.example.com   john
xxx               john

yyy.example.com   john
yyy               john

xxx.example.com   fred
xxx               fred

zzz.example.com   fred
zzz               fred
```

Note: On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the **User Manager** application.

Related concepts:

➥ Database Server Groups (Enterprise Replication Guide)

"Redirecting clients with the connectivity information" on page 24-8

Related reference:

"The sqlhosts file on UNIX" on page 1-8

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

➥ INFORMIXSERVER environment variable (Guide to SQL: Reference)

➥ DBPATH environment variable (Guide to SQL: Reference)

➥ REMOTE_USERS_CFG configuration parameter (Administrator's Reference)

The netrc information

The `netrc` information is optional information that specifies identity data. A user who does not have authorization to access the database server or is not on a computer that is trusted by the database server can use this file to supply a name and password that are trusted. A user who has a different user account and password on a remote computer can also provide this information.

On UNIX, the `netrc` information is in the `.netrc` file in the user's home directory. Use any standard text editor to prepare the `.netrc` file. The format of a `netrc` entry is:

```
machine machine_name login user_name password user_password
```

Windows maintains the `netrc` information in the registry keys. Use the **Host Information** tab of **setnet32** to edit the `netrc` information.

If you do not explicitly provide the user password in an application for a remote server (that is, through the USER clause of the CONNECT statement or the user name and password prompts in DB-Access), the client application looks for the user name and password in the `netrc` information. If the user explicitly specified the password in the application, or if the database server is not remote, the `netrc` information is not consulted.

The database server uses the `netrc` information regardless of whether it uses the default authentication policy or a communications support module.

For information about the specific content of this file, see your operating system documentation.

Windows only: On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the User Manager application

Related reference:

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

User impersonation:
For certain client queries or operations, the database server must impersonate the client to run a process or program on behalf of the client. In order to impersonate the client, the database server must receive a password for each client connection. Clients can provide a user ID and password through the CONNECT statement or `netrc` information.

The following examples show how you can provide a password to impersonate a client.

netrc

```
machine trngpc3 login bruce password im4golf
```

CONNECT statement

```
CONNECT TO ol_trngpc3 USER bruce USING "im4golf"
```

The sqlhosts file and the SQLHOSTS registry key

IBM Informix client/server connectivity information, the *sqlhosts information*, contains information that enables a client application to find and connect to any IBM Informix database server on the network.

The sqlhosts file (UNIX)

On UNIX, the `sqlhosts` file is located, by default, in the `$INFORMIXDIR/etc` directory.

Each entry in the `sqlhosts` file contains the `sqlhosts` information for a database server or a database server group. Additional syntax rules for each of the fields are provided in the following sections, which describe the entries in the `sqlhosts` file. Use any standard text editor to enter information in the `sqlhosts` file.

The SQLHOSTS registry key (Windows)

When you install the database server, the setup program creates a key in the Windows registry:

- For an Informix 64-bit product installed on a Windows 64-bit system or an Informix 32-bit product installed on a Windows 32-bit system, the **sqlhosts** information is in this key:

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
  ```

- For an Informix 32-bit product installed on a Windows 64-bit system, the **sqlhosts** information is in this key:

  ```
  HKEY_CLASSES_ROOT\VirtualStore\MACHINE\SOFTWARE\Wow6432Node\Informix\SQLHOSTS
  ```

Note: If you have 32-bit and 64-bit Informix software (client or server) operating together on Windows on the same machine, you must have duplicate entries in both registry hives.

This branch of the **HKEY_LOCAL_MACHINE** subtree stores the SQLHOSTS information. Each key on the SQLHOSTS branch is the name of a database server. When you click the database server name, the registry displays the values of the **HOST**, **OPTIONS**, **PROTOCOL**, and **SERVICE** fields for that particular database server.

Each computer that hosts a database server or a client must include the connectivity information either in the SQLHOSTS registry key or in a central registry. When the client application runs on the same computer as the database server, they share a single SQLHOSTS registry key.

When you install the database server, the installation program asks where you want to store the SQLHOSTS registry key. You can specify one of the following two options:

- The local computer where you are installing the database server
- Another computer in the network that serves as a central, shared repository of SQLHOSTS information for multiple database servers in the network

Using a shared SQLHOSTS registry key relieves you of the necessity to maintain the same `sqlhosts` information about multiple computers. However, the `hosts` and `services` files on each computer must contain information about all computers that have database servers.

The following tools can be used to manage the SQLHOSTS information:
- IBM Informix Server Administrator (ISA)
- **regedt32**
- **Setnet32**

Although you can use **Setnet32** to set up database servers, you cannot use it to set up database server groups.

Related concepts:

"Redirecting clients with the connectivity information" on page 24-8

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

"Modifying the sqlhosts file for Connection Manager" on page 23-6

"Supporting multiplexed connections" on page 2-4

Related reference:

"The sqlhosts file on UNIX" on page 1-8

"Network protocol" on page 2-1

Creating the sqlhosts file with a text editor (UNIX)

On UNIX, the `sqlhosts` file is located, by default, in the `$INFORMIXDIR/etc` directory. As an alternative, you can set the **INFORMIXSQLHOSTS** environment variable to the full path name and file name of a file that contains the `sqlhosts` file information. Each computer that hosts a database server or a client must have an `sqlhosts` file.

Open any standard text editor to create the `sqlhosts` file

Note:
- Use white space (spaces, tabs, or both) to separate the fields.
- Do not include any spaces or tabs within a field.
- To put comments in the `sqlhosts` file, start a line with the comment character (#). You can also leave lines blank for readability.

Sample sqlhosts file

The following code block shows a sample `sqlhosts` file.

```
#dbservername    nettype      hostname     servicename     options
menlo            onipcshm     valley       menlo
newyork          ontlitcp     hill         dynsrvr2        s=2,b=5120
payroll          onsoctcp     dewar        py1
asia             group        -            -               e=asia.3
asia.1           ontlitcp     node6        svc8            g=asia
asia.2           onsoctcp     node0        svc1            g=asia
portland         drsocssl     dewar        portland_serv
```

Setting up the SQLHOSTS registry key with Setnet32 (Windows)

A client application connects to an Informix database server that is running on a computer that can be reached through the network. To establish the connection, use **Setnet32** to specify the location of the Informix database server on the network and the network communications protocol to use. You must obtain this information from the administrator of the database server you want to use.

If you specify a shared SQLHOSTS registry key, you must set the INFORMIXSQLHOSTS environment variable on your local computer to the name of the Windows computer that stores the registry. The database server first looks for the SQLHOSTS registry key on the INFORMIXSQLHOSTS computer. If the database server does not find an SQLHOSTS registry key on the INFORMIXSQLHOSTS computer, or if INFORMIXSQLHOSTS is not set, the database server looks for an SQLHOSTS registry key on the local computer.

You must comply with Windows network-access conventions and file permissions to ensure that the local computer has access to the shared SQLHOSTS registry key. For information about network-access conventions and file permissions, see your Windows documentation.

1. Double-click **Setnet32** in the folder that contains the Client SDK products.

   The Informix **Setnet32** window opens.
2. Click the **Server Information** tab to display the **Server Information** page, which has the following elements:

   - **Informix Server**

     Select an existing Informix database server or type the name of a new database server.

   - **Host Name**

     Select the host computer with the database server that you want to use or type the name of a new host computer.

   - **Protocol Name**

     Select a network protocol from a list of protocols that the installation procedure provides.

   - **Service Name**

     Specify the service name that is associated with a specific database server. Type either the service name or the port number that is assigned to the database server on the host computer. You must obtain this information from the database server administrator.

     Requirement: If you enter a service name, it must be defined on the client computer in the `services` file in the Windows installation directory. This file is in `system32\drivers\etc\services`. The service definition must match the definition on the database server host computer.

   - **Options**

     Enter options specific to the database server. For more information, see the *IBM Informix Administrator's Guide*.

   - **Make Default Server**

     Sets the INFORMIXSERVER environment variable to the name of the current database server to make it the default database server.

   - **Delete Server**

     Deletes the definition of a database server from the Windows registry. It also deletes the host name, protocol name, and service name associated with that database server.

3. Click **OK** to save the values.

Sample SQLHOSTS registry key information

The following figure illustrates the location and contents of the SQLHOSTS registry key for the database server **payroll**.
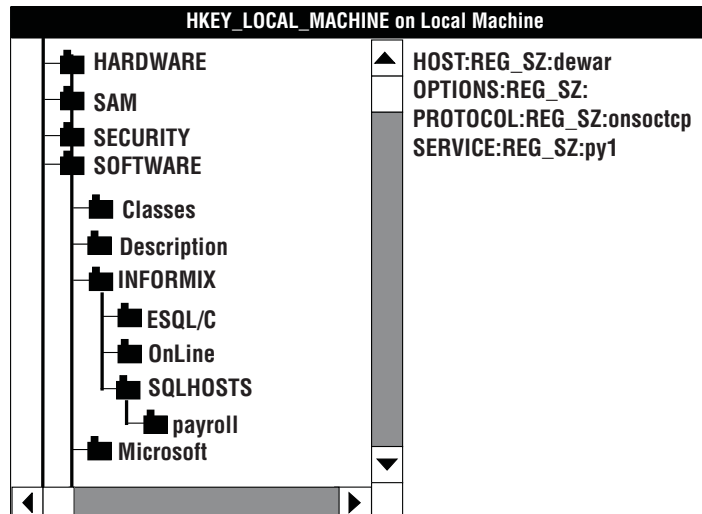
Figure 2-4. The sqlhosts information in the Windows registry

The sqlhosts information

The sqlhosts information in the sqlhosts file on UNIX or the SQLHOSTS registry key on Windows contains connectivity information for each database server. The sqlhosts information also contains definitions for groups. The database server looks up the connectivity information when you start the database server, when a client application connects to a database server, or when a database server connects to another database server.

The connectivity information for each database server includes four fields of required information and one optional field. The group information contains information in only three of its fields.

The five fields of connectivity information form one line in the UNIX sqlhosts file. On Windows, the database server name is assigned to a key in the SQLHOSTS registry key, and the other fields are values of that key. The following table summarizes the fields used for the SQLHOSTS information.

UNIX field name	Windows field name	Description of connectivity information	Description of group information
dbservername	Database server name key or database server group key	Database server name	Database server group name
nettype	PROTOCOL	Connection type	The word group
hostname	HOST	Host computer for the database server	No information. Use a dash as a placeholder in this field.
servicename	SERVICE	Alias for the port number	No information. Use a dash as a placeholder in this field.

UNIX field name	Windows field name	Description of connectivity information	Description of group information
options	OPTIONS	Options that describe or limit the connection	Group options

UNIX only: If you install IBM Informix Enterprise Gateway with DRDA® in the same directory as the database server, your `sqlhosts` file also contains entries for the Gateway and non-Informix database servers. However, this manual covers only the entries for the database server. For information about other entries in the `sqlhosts` file, see the *IBM Informix Enterprise Gateway with DRDA User Manual*.

Related tasks:

"Connections that the database server supports" on page 2-5

IANA standard service names and port numbers in the sqlhosts.std file

The Internet Assigned Numbers Authority (IANA) assigned the following service names and port numbers for IBM Informix database servers:

Port/service	IANA code	Description
sqlexec	9088/tcp	IBM Informix SQL Interface
sqlexec-ssl	9089/tcp	IBM Informix SQL Interface - Encrypted

These service names are created in the `sqlhosts.std` file of IBM Informix. You are not required to change installed Informix systems, because they continue to work with existing port numbers and service names. (Also, there is no guarantee that some other system is not already using the service names or port numbers assigned to Informix.)

Organizations that have policies for following standards can use these service names and port numbers if they want the database server to be in compliance with the IANA standard. If another application that is installed on the same workstation already uses one of the service names or port numbers, you can ask the publisher of the non-compliant application to register for an IANA port number assignment to avoid the conflict. When applications are noncompliant, you can run Informix using non-standard ports.

For more information, see the IANA organization website.

sqlhosts Connectivity information

This topic describes the connectivity information for each field of the `sqlhosts` file or SQLHOSTS registry key.

Syntax
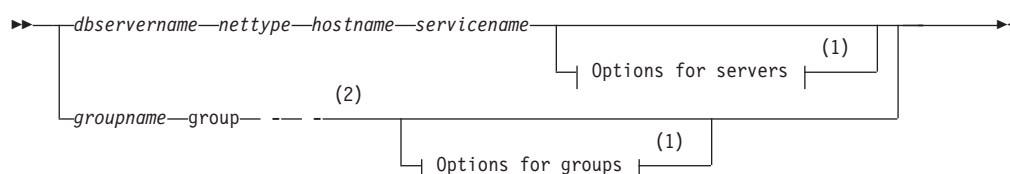
Notes:

1   See "sqlhosts file and SQLHOSTS registry key options" on page 2-25

2   UNIX only; fields are blank on Windows

Element	Purpose	Restrictions
dbservername	Names the database server for which the connectivity information is being specified.	Must match the name of a database server in the network, as specified by the DBSERVERNAME and DBSERVERALIASES configuration parameters in the `onconfig` file. Can include any printable character other than an uppercase character, a field delimiter, a new line character, or a comment character. The field length is limited to 128 bytes.
nettype	Describes the type of connection made between the database server and the client application or another database server.	
hostname	Specifies the computer where the database server is located.	The field length is limited to 256 bytes.
servicename	Specifies the alias for the port number. The interpretation of the service name field depends on the type of connection that the connection-type field (*nettype* or PROTOCOL) specifies.	The field length is limited to 128 bytes.
groupname group	You can use database server groups to treat multiple related database server entries as one logical entity to establish or change client/server connections. You can also use database server groups to simplify the redirection of connections to database servers.	The name must begin with a lowercase letter, and can contain lowercase letters, numbers, and underscore (_) symbols. The field length is limited to 128 bytes. Database server groups cannot be nested inside other database server groups, and database server group members cannot belong to more than one group.

dbservername field

Each database server across all of your associated networks must have a unique database server name.

UNIX only: If the `sqlhosts` file has multiple entries with the same dbservername, only the first one is used.

Connection-type field

The connection-type field is called *nettype* on the UNIX operating system and PROTOCOL on the Windows operating system.

The following table summarizes the possible connection-type values for database server connections.

Table 2-2. Summary of nettype and PROTOCOL values

nettype value (UNIX or Linux)	PROTOCOL value (Windows)	Description	Connection type
drsocssl	drsocssl	Secured Sockets Layer (SSL) protocol for DRDA. You must configure a new server alias in the `sqlhosts` file or SQLHOSTS registry that uses `drsoctcp` connection protocol.	Network
drsoctcp	drsoctcp	Distributed Relational Database Architecture™ (DRDA) - connection for IBM data server client. You must configure a new server alias in the `sqlhosts` file or SQLHOSTS registry that uses `drsoctcp` connection protocol.	Network
drtlitcp	drtlitcp	Distributed Relational Database Architecture (DRDA) - connection for IBM data server client. You must configure a new server alias in the `sqlhosts` file or SQLHOSTS registry that uses `drtlitcp` connection protocol.	Network
onipcshm		Shared-memory communication. Requires the `cfd` option in the `sqlhosts` file if used for a non-root installation where the server and client are in different locations.	IPC
onipcstr		Stream-pipe communication. Requires the `cfd` option in the `sqlhosts` file if used for a non-root installation where the server and client are in different locations.	IPC
	onipcnmp	Named-pipe communication	IPC
ontlitcp		TLI with TCP/IP protocol	Network
onsocssl	onsocssl	Secured Sockets Layer (SSL) protocol	Network
onsoctcp	onsoctcp	Sockets with TCP/IP protocol	Network
onsocimc		Sockets with TCP/IP protocol for communication with Informix MaxConnect	Network
ontliimc		TLI with TCP/IP protocol for communication with Informix MaxConnect	Network
onsqlmux	onsqlmux	Multiplexed connection	Network

Note: The listed nettype and PROTOCOL values that begin with "on" can use "ol" in the place of "on". For example, either `onipcshm` or `olipcshm` specify shared-memory connections if used in the `sqlhosts` file.

Host name

The host name is entered in the `hostname` field on the UNIX operating system and in the **HOST** registry key on the Windows operating system.

If the connection type is `onsqlmux`, the `hostname` field must not be empty, but any specific value entered in it is ignored.

Following is an explanation of how client applications derive the values used in the host name field.

Network communication with TCP/IP

When you use the TCP/IP connection protocol, the `host name` field is a key to the `hosts` file, which provides the network address of the computer. The name that you use in the `hostname` field must correspond to the name in the `hosts` file. In most cases, the host name in the `hosts` file is the same as the name of the computer.

In some situations, you might want to use the actual Internet IP address in the host name field.

Shared-memory and stream-pipe communication (UNIX)

When you use shared memory or stream pipes for client/server communications, the `hostname` field must contain the actual host name of the computer on which the database server is located.

Multiplexed connections

When you use `onsqlmux` as the connection type, the `hostname` field must have an entry, but the entry is ignored. Dashes (`-`) can be used as entries.

Service name field

Network communication with TCP/IP

When you use the TCP/IP connection protocol, the service name entry must correspond with the name in the `services` file. The port number in the `services` file tells the network software how to find the database server on the specified host.

The service name field is called `servicename` on the UNIX operating system and **SERVICE** on the Windows operating system.

The following figure shows the relationship between the `sqlhosts` file or registry and the `hosts` file, and the relationship of `sqlhosts` to the `services` file.

sqlhosts entry to connect by TCP/IP

dbservername	nettype	hostname	servicename	options
sales	onsoctcp	knight	sales_ol	

hosts file

IP address	hostname	alias
37.1.183.92	knight	

services file

service name	port#/protocol
sales_ol	1543/tcp
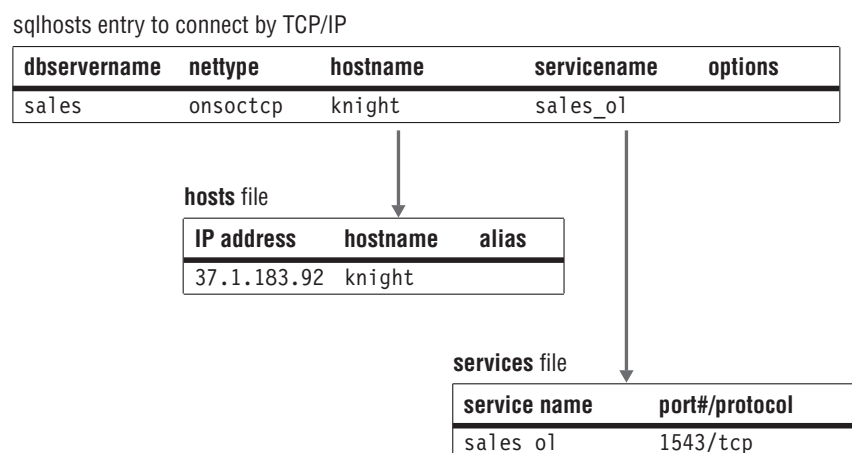
Figure 2-5. Relationship of `sqlhosts` file or registry to `hosts` and `services` files

In some cases, you might use the actual TCP listen-port number in the service name field.

Named-pipe communication (Windows)

When the PROTOCOL ENTRY specifies a named-pipe connection (`onipcnmp`), the SERVICE entry can be any short group of letters that is unique in the environment of the host computer where the database server is located.

Shared-memory and stream-pipe communication (UNIX)

When the *nettype* field specifies a shared-memory connection (`onipcshm`) or a stream-pipe connection (`onipcstr`), the database server uses the value in the *servicename* entry internally to create a file that supports the connection. For both `onipcshm` and `onipcstr` connections, the *servicename* can be any short group of letters that is unique in the environment of the host computer where the database server is located.

Tip: Use the *dbservername* as the *servicename* for stream-pipe connections.

Multiplexed connections

When you use `onsqlmux` as the connection type, the `hostname` field must have an entry, but the entry is ignored. Dashes (-) can be used as entries.

Related tasks:

"Configuring secure connections for clusters" on page 21-5

"Changing client connectivity information" on page 24-9

"Connections that the database server supports" on page 2-5

Related reference:

"ONCONFIG parameters related to connectivity" on page 2-40

"Group information" on page 2-34

"Stream-pipe connections (UNIX and Linux)" on page 2-7

"Specifying Network Connections" on page 4-21

"IBM Informix MaxConnect" on page 2-54

"TCP/IP connectivity files" on page 2-10

"Alternatives for TCP/IP connections" on page 2-36

"Connection Manager configuration file" on page 23-6

sqlhosts file and SQLHOSTS registry key options

This topic explains the various options that are available to include in the options field of the `sqlhosts` file or SQLHOSTS registry key

The following syntax fragment shows the server options in the **sqlhosts** file.

Server options:
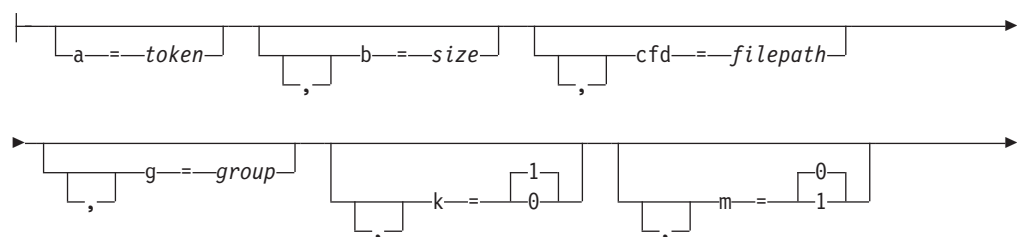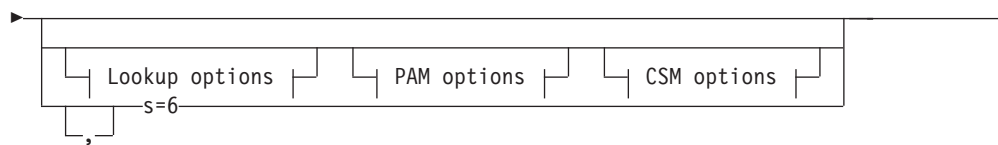
```
        ┌─────────────────────────────────────────────────────────────┐
►───────┼──────────────────────────┬──────────────────┬──────────────────┼───────┤
        │  ┌─ Lookup options ─┤  ┌──┤ PAM options ├──  ┌──┤ CSM options ├──│
        │  └──────── s=6 ──────────┘                                      │
        └─ , ─┘
```
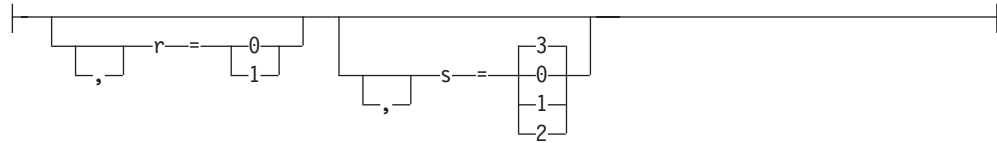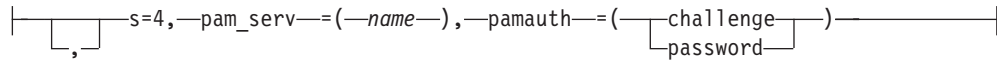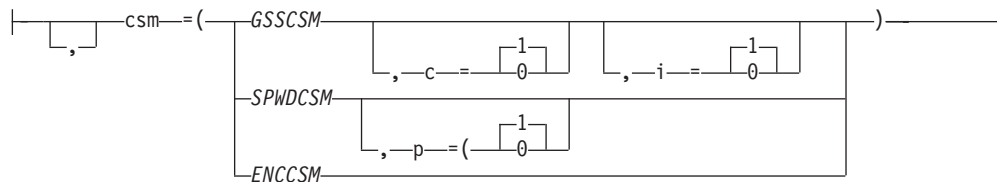
Lookup options:

```
├────┬────────────────────┬──┬──────────────────────┬──────────────────────────┤
     │  ┌────┐             │  │  ┌────┐      ┌─ 3 ─┐ │
     └──┤  , ├── r ── = ──┬┤  └──┤  , ├── s ─=─┼─ 0 ─┤ │
          └────┘      ├─ 0 ─┤      └────┘      ├─ 1 ─┤
                      └─ 1 ─┘                  └─ 2 ─┘
```

PAM options:

```
├──┬────┬── s=4, ── pam_serv ── = ( ─ name ─ ) , ── pamauth ── = ( ─┬─ challenge ─┬─ ) ──┤
   │ , │                                                            └─ password ──┘
   └────┘
```

CSM options:

```
├──┬────┬── csm ── = ( ─┬─ GSSCSM ────────────────────────────────────────────┬─ ) ──┤
   │ , │                │        ┌────┐  ┌─ 1 ─┐   ┌────┐  ┌─ 1 ─┐            │
   └────┘                │        └─ , ── c ─ = ─┼─ 0 ─┤   └─ , ── i ─ = ─┼─ 0 ─┤      │
                        ├─ SPWDCSM ──────────────────────────────────────────┤
                        │         ┌────┐      ┌─ 1 ─┐                        │
                        │         └─ , ── p ── = ( ─┼─ 0 ─┤ )                │
                        └─ ENCCSM ───────────────────────────────────────────┘
```

Important: Options must be separated by commas, but the first option listed in each `sqlhosts` entry must not have a comma before it.

The following syntax fragment shows the group options in the `sqlhosts` file.

Group options:
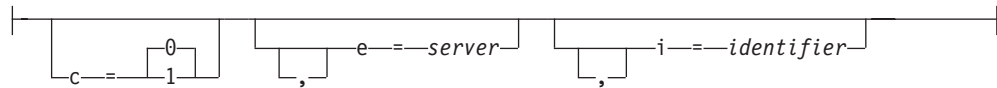
```
├──┬──────────────────┬──┬───────────────────┬──┬──────────────────────────┬──┤
   │        ┌─ 0 ─┐   │  │  ┌────┐            │  │  ┌────┐                  │
   └─ c ── = ─┼─ 1 ─┤   │  └─ , ── e ── = ── server ─┘  └─ , ── i ── = ── identifier ─┘
             └─────┘
```

Important: Options must be separated by commas, but the first option listed in each `sqlhosts` entry must not have a comma before it.

Table 2-3. Server options in the `sqlhosts` file and SQLHOSTS registry key.

Element	Purpose	Restrictions
a	Stores the authentication token required for connecting to the Informix Warehouse Accelerator (IWA). This entry is not created by a DBA, but by Informix during the IWA connection setup.	**Important:** Do not manually change this option.

Table 2-3. Server options in the `sqlhosts` file and SQLHOSTS registry key. (continued)

Element	Purpose	Restrictions
b	Specifies, in bytes, the size of the communications buffer space for TCP/IP connections.	On many operating systems, the maximum buffer size supported is 16 KB.
cfd	Indicates the storage location for communication files used in shared-memory and stream-pipe connections.	The length of the `cfd` path is restricted to 70 bytes. Relative-path byte lengths include $INFORMIXDIR.
csm	Describes the communication support module (CSM) for each database server that uses a CSM.	CSM entries must be specified in the `concsm.cfg` file.
ENCCSM	The name of the encryption communication support module.	The ENCCSM must be specified in the `concsm.cfg` file. You cannot use an ENCCSM with • Enterprise Replication and high-availability clusters • A multiplexed connection • A simple password CSM (SPWDCSM)
g	Specifies the name of the group to which the database server belongs.	The group must be defined.
GSSCSM	The name of the generic security services communications support module for single sign-on (SSO) authentication.	The GSSCSM must be specified in the `concsm.cfg` file. Cannot be used for Enterprise Replication and high-availability clusters.
k	Enables the network service to check periodically whether the connection between the client and server is still active. If the connection is found to be broken the network service frees resources.	Only available for TCP/IP connections.
m	Enables the database server to create multiple database connections without using up the additional computer resources that are required for additional network connections.	• Multithreaded client connections, shared-memory connections, and connections to subordinate database servers are not supported. • The Informix ESQL/C sqlbreak() function is not supported. • Cannot be used with a CSM.
r	Enables the control of operating-system security-file lookups to control the way that a client (user) gains access to a database server. The s option identifies database server-side settings, and the r option identifies client-side settings.	The database server ignores r settings.

Table 2-3. Server options in the `sqlhosts` *file and SQLHOSTS registry key. (continued)*

Element	Purpose	Restrictions
s	Enables the control of operating-system security-file lookups to control the way that a client (user) gains access to a database server. The s option identifies database server-side settings, and the r option identifies client-side settings.	A client ignores s settings.
pam_serv	Gives the name of a PAM service that a database is using.	Must be used with s=4 option.
pamauth	Describes the authorization method used by the PAM service.	Must be used with s=4 option.
SPWDCSM	The name of the simple password communication support module	The SPWDCSM must be specified in the `concsm.cfg` file. You cannot use an SPWDCSM with • Enterprise Replication and high-availability clusters • A multiplexed connection • An encryption CSM (ENCCSM)

Table 2-4. Group options in the `sqlhosts` *file and SQLHOSTS registry key.*

Element	Purpose	Restrictions
c	Indicates the order in which client applications choose database servers or aliases within a database server group that is aligned with the `sqlhosts` file.	
e	Specifies a database server name that marks the end of a database server group.	
i	Assigns an identifying number to a database server group.	The identifier must be a positive integer from 1 through 32767 and must be unique within your network environment. The i option is required for Enterprise Replication.

Usage

When you change option values in an `sqlhosts` entry, those changes affect the next connection that a client application makes, and the server automatically recognizes any changes made to the `sqlhosts` file.

The database server evaluates the options entries as a series of columns. A comma or white space in an options entry represents an end of a column. Client and database server applications check each column to determine whether the option is supported.

You can combine multiple options in each entry, and you can include them in any order. The maximum length for an options entry is 256 bytes.

CAUTION:
Unsupported or incorrect options do not trigger a notification.

Buffer option (b)

The b option applies only to connections that use the TCP/IP connection protocol. Other types of connections ignore the b option.

You can adjust the buffer size to use system and network resources more efficiently; however, if the buffer size is set too high, the user receives a connection-reject error because no memory can be allocated. For example, if you set b=64000 on a system that has 1000 users, the system might require 64 megabytes of memory for the communications buffers. This setting might exhaust the memory resources of the computer. The default buffer size for the database server for TCP/IP is 4096 bytes.

If your network includes several different types of computers, be careful when you change the size of the communications buffer.

Tip: Use the default size for the communications buffer. If you choose to set the buffer size to a different value, set the client-side communications buffer and the database server-side communications buffer to the same size.

Connection redirection option (c)

The c option is valid only for servers assigned to a server group.

Use the c option to:
- Balance the load across multiple database server instances.
- Set High-Availability Data Replication (HDR) to transfer over to a backup database server in the event of a failure.

Table 2-5. Settings for the connection-redirection option.

Setting	Result
c=0	This is the default setting. Client applications connect to the first database server instance listed in the server group in the sqlhosts file. If the client cannot connect to the first instance, it attempts to connect to the second instance, and so on.
c=1	Client applications choose a random starting point from which to connect to a database server instance in a server group.

Communication files directory option (cfd)

You can use the communication files directory option to store shared-memory or stream-pipe connection communication files in a new location. Specifying the communication files directory option for non-root installations of Informix is necessary if the server and client are in different locations, and increases system performance if the server and client are in the same location.

The cfd option can define an absolute path or a path relative to $INFORMIXDIR for storing communication files:
- cfd=/*location* defines an absolute path
- cfd=*location* defines a path relative to $INFORMIXDIR

The length of the cfd path is restricted to 70 bytes. Relative-path byte lengths include $INFORMIXDIR.

Non-root installations of Informix do not have permission to write to the /INFORMIXTMP directory, so shared-memory and stream-pipe connection communication files are written to the $INFORMIXDIR/etc directory if no communication files directory is specified as an option in the sqlhosts file.

Important: This option must be defined for non-root installations of Informix, where the server and client are in different locations, or the connection fails.

Communication support module option (csm)

The format of the CSM option is csm=(*name*,*options*)

The value of *name* must match a name entry in the concsm.cfg file.

CSM options defined in the sqlhosts file override options specified in the concsm.cfg file. CSM encryption options cannot be specified in the sqlhosts file.

If you do not specify the csm option, the database server uses the default authentication policy for that database server.

Note: The s=7 option is deprecated and not required for the Single Sign-On (SSO) CSM.

Table 2-6. CSM options for the sqlhosts file

Option	Description	Settings
p	Password The option is available as part of the simple password CSM, which provides password encryption.	• p=0 password is not required (default) • p=1 password is required
c	Confidentiality service Data transmitted to and from the SSO-authenticated user is encrypted and can be viewed only by the user logged in with the authorized credentials. The option is available as part of the Generic Security Services CSM, which supports single sign-on (SSO).	• c=1 enables the service (default) • c=0 disables the service

Table 2-6. CSM options for the sqlhosts file (continued)

Option	Description	Settings
i	Integrity service Ensures that data sent between user and the DBMS is not altered during transmission. The option is available as part of the Generic Security Services CSM, which supports single sign-on (SSO).	• i=1 enables the service (default) • i=0 disables the service

End of group option (e)

Members of a server group are not required to be in order in the sqlhosts file. You can use the e option to specify the end of a server group, and improve system performance. The network layer scans the sqlhosts file until the entry specified by the e option is read. If no e option is specified for a group, but all sqlhosts entries specify either groups or group members, the network must scan the entire file.

If no end-of-group option is specified for a group, the group members are assumed to be contiguous. The end of group is determined when an entry is reached that does not belong to the group, or at the end of file, whichever comes first.

In the following example, the e option specifies entry lx3, so entry lx4 is not scanned by the network layer.

```
#dbservername   nettype   hostname   servicename   options
g_x1            group     -          -             i=10,e=lx3
lx1             onsoctcp  apollo11   9810          g=g_x1
lx2             onsoctcp  apollo12   9820          g=g_x1
lx3             onsoctcp  apollo13   9830          g=g_x1

lx4             onsoctcp  apollo14   9840
```

Keep-alive option (k)

This option enables the network service to check periodically whether the connection between the client and server is still active. If the receiving end of the connection does not respond within the time specified by the parameters of your operating system, the network service immediately detects the broken connection and frees resources.

Table 2-7. Settings for the keep-alive option

Setting	Result
k=0	Disables this service
k=1	Enables this service (default)

Multiplex option (m)

This option enables the database server to create multiple database connections to client applications without using up the additional computer resources that are required for additional network connections. You must restart the server after you enable this service.

Table 2-8. Settings for the multiplex option

Setting	Result
m=0	Disables this service (default)
m=1	Enables this service

PAM options (pam_serv and pam_auth)

The database server provides an interface to use PAM for session authentication. To configure this interface, supply the PAM service name and authentication method. Authentication can be the connection password or a user challenge that requires the user to answer a question. Informix PAM authentication calls the pam_authenticate() and pam_acct_mgmt() functions.

Table 2-9. Settings for PAM services

Option	Description	Settings
pam_serv	The name of the PAM service that the database server is using.	PAM services typically are in the /usr/lib/security directory and parameters are listed in the /etc/pam.conf file. In Linux, the /etc/pam.conf file can be replaced with a directory called /etc/pam.d, where there is a file for each PAM service. If /etc/pam.d exists, /etc/pam.conf is ignored by Linux.
pamauth	The method of authentication used by the PAM service uses. With this authentication mode, an application must be designed to respond to the challenge prompt correctly before connecting to the database server.	pamauth=password uses the connection request password for authentication pamauth=challenge authentication requires a correct user reply to a question or prompt

Trusted-host and trusted-user lookup options (s)

With these security options, you can specifically enable or disable the use of either or both files.

Table 2-10. Settings for trusted-host and trusted-user lookup.

Setting	Result
s=0	Disables trusted-hosts lookup in `hosts.equiv` or the file specified by the REMOTE_SERVER_CFG configuration parameter.
	Disables trusted-user lookup in `rhosts` files or the file specified by the REMOTE_USERS_CFG configuration parameter.
	Only incoming connections with passwords are accepted. Cannot be used for distributed database operations.
s=1	Enables trusted-hosts lookup in `hosts.equiv` or the file specified by the REMOTE_SERVER_CFG configuration parameter.
	Disables trusted-user lookup in `rhosts` files or the file specified by the REMOTE_USERS_CFG configuration parameter.
s=2	Disables trusted-hosts lookup in `hosts.equiv` or the file specified by the REMOTE_SERVER_CFG configuration parameter.
	Enables trusted-user lookup in `rhosts` files or the file specified by the REMOTE_USERS_CFG configuration parameter.
	Cannot be used for distributed database operations.
s=3	Enables trusted-hosts lookup in `hosts.equiv` or the file specified by the REMOTE_SERVER_CFG configuration parameter.
	Enables trusted-user lookup in `rhosts` files or the file specified by the REMOTE_USERS_CFG configuration parameter.(default)

Secure connections for clusters option (s=6)

The `s=6` option in the `sqlhosts` file ensures that the connections between cluster servers are trusted. Secure ports listed in the sqlhosts file can be used only for cluster communication. Client applications cannot connect to secure ports.

Table 2-11. The secure connection option for clusters.

Setting	Result
s=6	Configures Enterprise Replication and High Availability Connection Security. Cannot be used with any other s option.

netrc lookup options (r)

With r options, you can enable or disable netrc lookup.

Table 2-12. Settings for netrc lookup options.

Setting	Result
r=0	`netrc` lookup is disabled.
r=1	`netrc` lookup is enabled (default)

Related concepts:

"Trusted-host information" on page 2-13

"Trusted-user information" on page 2-14

↪ Pluggable authentication modules for systems running on UNIX or Linux (Security Guide)

↪ Communication support modules for data transmission encryption (Security Guide)

↪ Simple password encryption (Security Guide)

Related tasks:

"Supporting multiplexed connections" on page 2-4

"Configuring secure connections for clusters" on page 21-5

↪ Configuring an IBM Informix instance for SSO (Security Guide)

↪ Configuring secure connections for replication servers (Enterprise Replication Guide)

Related reference:

"Group information"

"The netrc information" on page 2-16

Group information

You define server groups in the `sqlhosts` file or SQLHOSTS registry key. When you create a server group, you can treat multiple related database server entries as one logical entity for establishing or changing client/server connections. You can also use groups to simplify the redirection of connections to database servers.

To use server groups for Enterprise Replication, use the `i` and `g` options. All database servers that participate in replication must be a member of a database server group. Each database server in the enterprise must have a unique identifier, which is the server group. Make sure that the `sqlhosts` file is set up appropriately on each database server that participates in replication.

To use database server groups for High-Availability Data Replication (HDR), use the `c`, `e`, and `g` options in HDR. HDR requires two identical systems.

You can use the name of a database server group instead of the database server name in the following environment variables, or in the SQL CONNECT command:

- The value of the **INFORMIXSERVER** environment variable for a client application can be the name of a database server group. However, you cannot use a database server group name as the value of the **INFORMIXSERVER** environment variable for a database server or database server utility.
- The value of the **DBPATH** environment variable can contain the names of database server groups as database server names.

Related reference:

"sqlhosts Connectivity information" on page 2-21

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

Creating a database server group in the sqlhosts file (UNIX)

You can define a database server group and the members of the group by adding entries to the `sqlhosts` file.

To create a database server group in the `sqlhosts` file:

1. Add an entry to define the database server group:

   dbservername
   > The name of the group. The name must begin with a lowercase letter, and can contain lowercase letters, numbers, and underscore (_) symbols.

   nettype
   > The word `group`.

   hostname
   > A dash (`-`) character (ASCII 45), to indicate that the field value is null.

   servicename
   > A dash (`-`) character (ASCII 45), to indicate that the field value is null.

   options
   > The `c`, `e`, or `i` options, as appropriate.

2. Add one or more entries for database servers that are members of the group. Include the `g=`*groupname* option.

Example

The following example shows a database server group definition named **asia** and the **asia** group members in the `sqlhosts` file:

```
#dbservername   nettype   hostname   servicename   options
asia            group     –          –             e=asia.4
asia.1          ontlitcp  node6      svc8          g=asia
asia.2          onsoctcp  node0      svc1          g=asia
asia.3          onsoctcp  node10     svc10         g=asia
asia.4          onsoctcp  node1      svc9          g=asia
```

Setting up the database server group registry key Windows

To set up the database server group registry key on Windows:

1. Run the Windows program, **regedit**.
2. In the **Registry Editor** window, select the window for the **HKEY_LOCAL_MACHINE** subtree.
3. Click the folder icons to select the **\SOFTWARE\INFORMIX\SQLHOSTS** branch.
4. With the **SQLHOSTS** key selected, add a key.
5. Give the new key the name of the database server group.

   This value must correspond to the **OPTIONS** value in the database server name key.

6. Select the key with the database server group name that you created and add a string value for it.
7. Give the value the name of one of the fields of the SQLHOSTS information (**HOST**, **OPTIONS**, **PROTOCOL**, **SERVICE**).
8. Add a value for the field.

   For a database server group, the SQLHOSTS registry key fields must have the following values:

```
HOST       -
OPTIONS    i=unique_integer
PROTOCOL   group
SERVICE    -
```

Each database server group must have an associated identifier (i) value that is unique among all database servers in your environment. Use dash (-) characters (ASCII 45) as null-field indicators for **HOST** and **SERVICE** to indicate that you are not assigning specific values to those fields.

9. Repeat steps 7 on page 2-35 and 8 on page 2-35 for the remaining fields of the SQLHOSTS registry hey.

10. Select the database server group key and choose to add a key.

11. Give the new key the name of the database server.

This value must correspond to the database server key, whose **OPTIONS** value was set to the database server group key.

12. If you are combining Enterprise Replication with HDR, create keys for primary and secondary HDR servers under the same database server group.

Alternatives for TCP/IP connections

The following topic describes some ways to bypass port and IP address lookups for TCP/IP connections.

IP addresses for TCP/IP connections

For TCP/IP connections (both TLI and sockets), you can use the actual IP address in the **hostname** field instead of the host name or alias found in the `hosts` file. The following example shows sample IP addresses and hosts from a `hosts` file.

```
#address        hostname  alias
555.12.12.12    smoke
98.555.43.21    odyssey
12.34.56.555    knight    sales
```

Using the IP address for **knight** from the table, the following two `sqlhosts` entries are equivalent:

```
#dbservername   nettype   hostname      servicename   options
sales           ontlitcp  12.34.56.789  sales_ol

#dbservername   nettype   hostname      servicename   options
sales           ontlitcp  knight        sales_ol
```

Using an IP address might speed up connection time in some circumstances. However, because computers are usually known by their host name, using IP addresses in the host name field makes it less convenient to identify the computer with which an entry is associated.

UNIX only: You can find the IP address in the net address field of the `hosts` file, or you can use the UNIX **arp** or **ypmatch** command.

Windows only: You can configure Windows to use either of the following mechanisms to resolve a domain to an IP address:
• Windows Internet Name Service
• Domain Name Server

Wildcard addressing for TCP/IP connections

You can use wildcard addressing in the **hostname** field of the `hosts` file when both of the following conditions are met:
• You are using TCP/IP connections.
• The computer where the database server is located uses multiple network-interface cards (NICs).

If the preceding conditions are met, you can use an asterisk (*) as a *wildcard* in the **hostname** field that the database server uses. When you enter a wildcard in the **hostname** field, the database server can accept connections at any valid IP address on its host computer.

Each IP address is associated with a unique host name. When a computer uses multiple NICs, as in the following table, the `hosts` file must have an entry for each interface card. For example, the `hosts` file for the **texas** computer with two NICs might include these entries.

```
#address        hostname    alias
123.45.67.81    texas1
123.45.67.82    texas2
```

If the client application and database server share the `sqlhosts` file or the SQLHOSTS registry key, you can specify both the wildcard and a host name or IP address in the **hostname** field (for example, `*texas1` or `*123.45.67.81`). The client application ignores the wildcard and uses the host name (or IP address) to make the connection, and the database server uses the wildcard to accept a connection from any IP address.

The wildcard format allows the listen thread of the database server to wait for a client connection using the same service port number on each of the valid network-interface cards. However, waiting for connections at multiple IP addresses might require more processor time than waiting for connections with a specific host name or IP address.

The following figure shows a database server on a computer named **texas** that has two network-interface cards. The two client sites use different network cards to communicate with the database server.
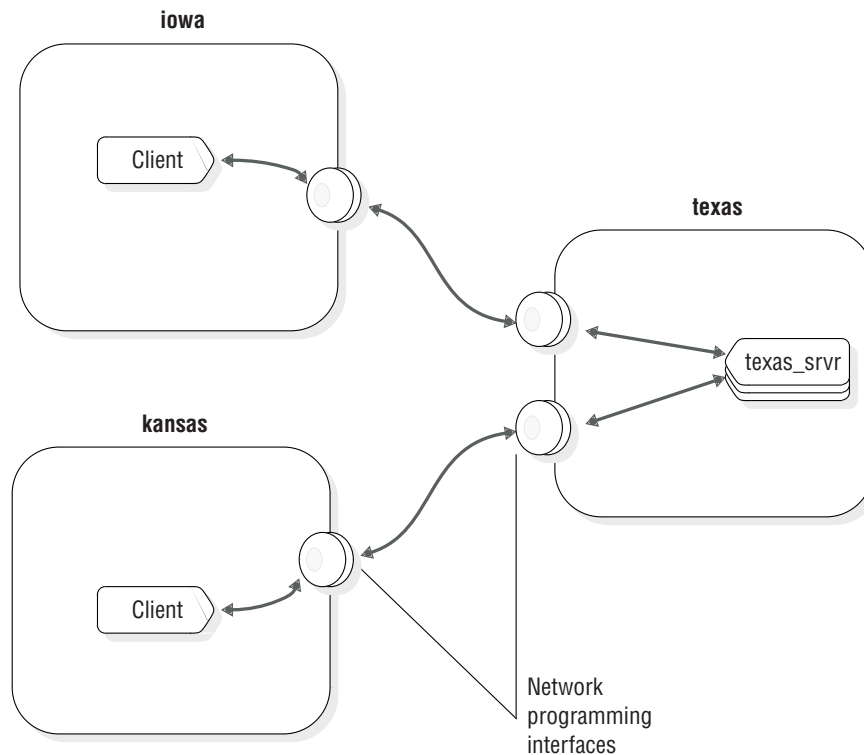
Figure 2-6. Using multiple network-interface cards

The following examples show potential sqlhosts connectivity information for the **texas_srvr** database server.

```
#dbservername   nettype    hostname        servicename   options
texas_srvr      ontlitcp   *texas1         pd1_on

#dbservername   nettype    hostname        servicename   options
texas_srvr      ontlitcp   *123.45.67.81   pd1_on

#dbservername   nettype    hostname        servicename   options
texas_srvr      ontlitcp   *texas2         pd1_on

#dbservername   nettype    hostname        servicename   options
texas_srvr      ontlitcp   *123.45.67.82   pd1_on

#dbservername   nettype    hostname        servicename   options
texas_srvr      ontlitcp   *               pd1_on
```

If the connectivity information corresponds to any of the preceding lines, the **texas_srvr** database server can accept client connections from either of the network cards. The database server finds the wildcard in the **hostname** field and ignores the explicit host name.

Tip: For clarity and ease of maintenance, include a host name when you use the wildcard in the host name field (that is, use *host instead of *).

The connectivity information used by a client application must contain an explicit host name or IP address. The client applications on **iowa** can use any of the following host names: texas1, *texas1, 123.45.67.81, or *123.45.67.81. If there is a wildcard (*) in the hostname field, the client application ignores it.

The client application on **kansas** can use any of the following host names: texas2, *texas2, 123.45.67.82, or *123.45.67.82.

Port numbers for TCP/IP connections

For the TCP/IP network protocol, you can use the actual TCP listen port number in the service name field.

For example, if the port number for the **sales** database server in the `services` file is 1543, you can write an entry in the `sqlhosts` file as follows:

```
#dbservername   nettype     hostname    servicename    options
sales           ontlitcp    knight      1543
```

Using the actual port number might save time when you make a connection in some circumstances. However, as with the IP address in the `hostname` field, using the actual port number might make administration of the connectivity information less convenient.

Related reference:

Informix support for IPv6 addresses

On all platforms, IBM Informix recognizes Internet Protocol Version 6 (IPv6) addresses, which are 128 bits long, and Internet Protocol Version 4 (IPv4) addresses, which are 32 bits long.

Beginning with Informix 10.00.xC4 and Client SDK 2.90.xC4, the database server checks, on startup, whether IPv6 is supported in the underlying operating system. If IPv6 is supported it is used. If the underlying operating system does not support IPv6, the IPv4 address is used. Informix and Client SDK retrieve the IP address from the name service.

You can treat Informix that runs on a host with both IPv4 and IPv6 addresses the same way you treat a server running on a multi-homed host. You can configure Informix on a host with both IPv4 and IPv6 addresses in either of the following ways:

- Create aliases (using the DBSERVERALIASES configuration parameter) and assign an IPv6 address to one of them and an IPv4 address to the other.
- Instruct the Informix to listen on all the IP addresses configured on the host by using a wild-carded hostname in the `sqlhosts` file.

  For example:

```
#dbservername   nettype     hostname    servicename    options
olserver1       onsoctcp    *myhost     onservice1
```

Starting with Informix Version 10.0, the host name entry in the SQLHOSTS file maps to an IPv6 address if the host has a configured IPv6 address. If the host does not have a configured IPv6 address, the hostname entry maps to an IPv4 address.

Disabling IPv6 Support

Informix also provides a way to disable IPv6 support when working in IPv4 environments.

To disable IPv6 support for all database instances and client applications:

- Create an empty file `$INFORMIXDIR/etc/IFX_DISABLE_IPV6`.

The file must have read permission for user **informix**. The file is not read from or written to, and is not required to contain any data.

To disable IPv6 support for a single database instance or for a single client application:

• On the database server instance, or on the workstation on which applications are run, create an environment variable named `IFX_DISABLE_IPV6` and set its value to **yes**, as in:

```
IFX_DISABLE_IPV6=yes
```

ONCONFIG parameters related to connectivity

Some of the configuration parameters in the `onconfig` file specify information related to connectivity.

When you restart the database server, the restart procedure uses the values that you set in these configuration parameters.

The following configuration parameters are related to connectivity:
• DBSERVERNAME
• DBSERVERALIASES
• LIMITNUMSESSIONS
• NETTYPE
• NS_CACHE
• NUMFDSERVER
• HA_ALIAS

Related tasks:

"Connections that the database server supports" on page 2-5

Related reference:

"sqlhosts Connectivity information" on page 2-21

Connection information set in the DBSERVERNAME configuration parameter

When a client application connects to a database server, it must specify the name for the database server. The `sqlhosts` information that is associated with the specified database server name describes the type of connection between the application and the database server.

For example, to assign the name **nyc_research** to a database server, set the DBSERVERNAME value in `onconfig` or Windows registry key:

```
DBSERVERNAME nyc_research
```

Client applications specify the name of the database server in one of the following places:

• In the `INFORMIXSERVER` environment variable
• In SQL statements such as CONNECT, DATABASE, CREATE TABLE, and ALTER TABLE, which specify a database environment
• In the `DBPATH` environment variable

The DBSERVERNAME must specify either the database server name or one of the database server aliases. The name must begin with a lowercase letter and can

contain other lowercase letters, digits, and underscores. The name must not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

For `onimcsoc` or `onsoctcp` protocols, you can update the DBSERVERNAME configuration parameter to include the number of multiple listen threads for the database server aliases in your `sqlhosts` information, as follows:

```
DBSERVERNAME name-number_of_multiple_listen_threads
```

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

Related reference:

➡ DBSERVERNAME configuration parameter (Administrator's Reference)

Connection information set in the DBSERVERALIASES configuration parameter

The DBSERVERALIASES configuration parameter lets you assign additional dbserver names to the same database server.

The maximum number of aliases is 32. The following example shows entries in an `onconfig` configuration file that assign three dbserver names to the same database server instance.

```
DBSERVERNAME          sockets_srvr
DBSERVERALIASES       ipx_srvr,shm_srvr
```

The `sqlhosts` entries associated with the dbserver names from previous example can include those entries shown in the following example. Because each dbserver name has a corresponding entry in the `sqlhosts` file or SQLHOSTS registry key, you can associate multiple connection types with one database server.

```
shm_srvr         onipcshm    my_host           my_shm
sockets_srvr     onsoctcp    my_host           port1
ipx_srvr         ontlispx    nw_file_server    ipx_srvr
```

Using the `sqlhosts` file shown in the previous example, a client application uses the following statement to connect to the database server using shared-memory communication:

```
CONNECT TO '@shm_srvr'
```

A client application can initiate a TCP/IP sockets connection to the *same* database server using the following statement:

```
CONNECT TO '@sockets_srvr'
```

DBSERVERALIASES must begin with a lowercase letter and can contain other lowercase letters, digits, and underscores. DBSERVERALIASES must not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

In the previous examples, the `@shm_srvr` statement connects to an unidentified database at that server; alternatively, you can connect to `dbase1@shm_srvr`.

For **onimcsoc** or **onsoctcp** protocols, you can update the DBSERVERALIASES configuration parameter to include the number of multiple listen threads for the database server aliases in your **sqlhosts** information, as follows:

`DBSERVERALIASES,name[-number],name[-number]]`

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

Related reference:

▐➡ DBSERVERALIASES configuration parameter (Administrator's Reference)

Connection information set in the LIMITNUMSESSIONS configuration parameter

The LIMITNUMSESSIONS configuration parameter is an optional parameter that specifies the maximum number of sessions that you want connected to IBM Informix. If you specify a maximum number, you can also specify whether you want Informix to print messages to the `online.log` file when the number of sessions approaches the maximum number.

Distributed queries against a server are counted against the limit.

You might be required to dynamically increase or temporarily turn off the LIMITNUMSESSIONS configuration parameter to allow administrative utilities to run if the database server is reaching the limit. Use **onmode -wf** or **onmode -wm** to dynamically increase or turn off LIMITNUMSESSIONS.

If the LIMITNUMSESSIONS configuration parameter is enabled and sessions are restricted because of this limit, both regular user threads and DBSA user threads connecting to any database count against the limit. However, a DBSA user is allowed to connect to the server even after the limit is reached.

The LIMITNUMSESSIONS configuration parameter is not intended to be used as a means to adhere to license agreements.

Example

The following example specifies that you want a maximum of 100 sessions to connect to the database server and you want to print a warning message when the number of connected sessions approaches 100: `LIMITNUMSESSIONS 100,1`

Connection information set in the NETTYPE configuration parameter

The NETTYPE configuration parameter lets you adjust the number and type of virtual processors the database server uses for communication. Each type of network connection (for example, ipcshm or soctcp) can have a separate NETTYPE entry in the configuration file.

Recommendation: Although the NETTYPE parameter is not a required parameter, you must set NETTYPE if you use two or more connection types. After the database server is running for some time, you can use the NETTYPE configuration parameter to tune the database server for better performance.

For more information about NETTYPE, see "Network virtual processors" on page 4-21. For information about the NETTYPE configuration parameter, see the *IBM Informix Administrator's Reference*.

Name service maximum retention time set in the NS_CACHE configuration parameter

The NS_CACHE configuration parameter defines the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache. If you specify maximum retention times, the database server gets host, service, user, and group database server information from the cache.

Each cache entry expires either after the time configured for the specific cache or when the time is reconfigured.

Usually the network name service provider (for example, DNS) is on a remote computer. To avoid spending the time required to return information from the network name service provider, you can use the NS_CACHE configuration parameter to specify the maximum retention times for obtaining information from one of the internal caches. Then Informix looks for information in the cache. If the information is not there, the database server queries the operating system for the information.

You can avoid many of these operating system lookups by using the Informix name service caching mechanism, which can keep and reuse each retrieved piece of information for a configurable amount of time.

The server can get information from the cache faster than it does when querying the operating system. However, if you disable one or more of these caches by setting the retention time to 0, the database server queries the operating system for the host, service, user, or group information.

As a DBA, you might want to modify the NS_CACHE configuration parameter settings if the network name service provider runs on a remote computer or the MSC VP is running with a large amount of processor usage.

For example, you can run the **onstat -g glo** command to check the `msc` VP usage in the `Individual virtual processors` portion of the output. In the following ouput sample, the `msc` processor usage, shown in the `usercpu` and `syscpu` columns is high. If you suspect the usage is high because the DNS call takes too much time, you can confirm the high usage with an operating system command and then modify the NS_CACHE configuration parameter settings.

```
Individual virtual processors:
vp    pid      class     usercpu   syscpu    total     Thread    Eff
1     2036     cpu       76.95     7.14      84.09     99.08     84%
2     2149     adm       0.00      0.00      0.00      0.00      0%
3     2151     LIC       0.00      0.00      0.00      0.00      0%
4     2260     lio       0.00      0.00      0.00      0.03      0%
5     2442     pio       0.00      0.00      0.00      0.00      0%
6     2443     aio       0.00      0.01      0.01      0.11      8%
7     2444     msc       14.18     14.64     28.82     199.91    14%
8     2446     fifo      0.00      0.00      0.00      0.00      0%
```

You might also want to specify NS_CACHE information, if your operating system does not have a name service (NS) cache or if you disabled the operating system NS cache.

Example

To define the maximum retention time for your host and service connections as 600 seconds, and to disable the maximum retention limit for your user and group database server connections, specify:

```
NS_CACHE host=600,service=600,user=0,group=0
```

Connection information set in the NUMFDSERVERS configuration parameter

For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections migrating between Informix virtual processors (VPs).

Specifying NUMFDSERVERS information is useful if Informix has a high rate of new connect and disconnect requests or if you find a high amount of contention between network shared file (NSF) locks.

Related reference:

➡ NUMFDSERVERS configuration parameter (Administrator's Reference)

Connection information set in the HA_ALIAS configuration parameter

The HA_ALIAS configuration parameter is an optional parameter that defines a network alias for a secondary server. The network alias that is specified by the HA_ALIAS configuration parameter is used when you specify a secondary server in **onmode -d** commands.

When a database server's HA_ALIAS configuration parameter is set, all server-to-server communication with other high-availability cluster nodes occurs through the specified network alias.

If the primary server in a high-availability cluster fails, the Connection Manager identifies a secondary server to promote to a primary server. If the secondary server's HA_ALIAS configuration parameter is set, then the HA_ALIAS network alias is used to identify the new primary. The HA_ALIAS configuration parameter only affects RS Secondary and SD Secondary server types.

The value of the HA_ALIAS configuration parameter must be one of the name values specified in either the DBSERVERNAME or the DBSERVERALIASES configuration parameter. The network alias must have a connection type that is a TCP network protocol.

Related concepts:

Chapter 23, "Connection Management," on page 23-1

Related reference:

➡ HA_ALIAS configuration parameter (Administrator's Reference)

➡ onmode -d: Set data-replication types (Administrator's Reference)

➡ DBSERVERALIASES configuration parameter (Administrator's Reference)

➡ DBSERVERNAME configuration parameter (Administrator's Reference)

Environment variables for network connections

The **INFORMIXCONTIME** (connect time) and **INFORMIXCONRETRY** (connect retry) environment variables are client environment variables that affect the behavior of the client when it is trying to connect to a database server. Use these environment variables to minimize connection errors caused by busy network traffic.

If the client application explicitly attaches to shared-memory segments, you might be required to set **INFORMIXSHMBASE** (shared-memory base).

You can use the **INFORMIXSERVER** environment variable to specify a default dbserver name to which your clients connect.

Related concepts:

"How a client attaches to the communications portion (UNIX)" on page 6-5

Related tasks:

"Connections that the database server supports" on page 2-5

Related reference:

➡ INFORMIXCONTIME environment variable (Guide to SQL: Reference)

➡ INFORMIXCONRETRY environment variable (Guide to SQL: Reference)

➡ INFORMIXSHMBASE environment variable (UNIX) (Guide to SQL: Reference)

➡ INFORMIXSERVER environment variable (Guide to SQL: Reference)

Distributed Relational Database Architecture (DRDA) communications

This section contains information about how to configure IBM Informix to use the Distributed Relational Database Architecture (DRDA), which is a set of protocols that enables multiple database systems and application programs to work together.

Overview of DRDA

Distributed Relational Database Architecture (DRDA) is a set of protocols that enable communication between applications and database systems on disparate platforms and enables relational data to be distributed among multiple platforms.

Any combination of relational database management products that use DRDA can be connected to form a distributed relational database management system. DRDA coordinates communication between systems by defining what must be exchanged and how it must be exchanged.

You can configure the database server to use DRDA to respond to requests from a common API, such as the IBM Data Server JDBC Driver or the IBM Data Server .NET Provider.

You can also enable DRDA connections between a client API and the Informix Connection Manager (the **oncmsm** utility), which manages and redirects client connection requests between a primary server and one or more secondary servers. The Connection Manager provides automatic failover and load balancing capabilities. When a client makes a request to connect to a server, the Connection Manager routes the connection request to the appropriate server based on a configurable service level agreement. You can configure the Connection Manager to automatically fail over to specific servers or to specific types of servers.

Enterprise Replication, data replication, and Informix utilities, such as DB-Access, require SQLI connections. Enterprise Replication utilities do not operate over DRDA connections. However, Enterprise Replication connections can coexist with DRDA connections.

Communication between a common API and Informix can use encrypted password security or an encrypted user ID and encrypted password security.

You can use the Secure Sockets Layer (SSL) protocol to encrypt data in end-to-end, secure TCP/IP and DRDA connections between Informix and a client common API.

Related concepts:

↪ Secure sockets layer protocol (Security Guide)

Chapter 23, "Connection Management," on page 23-1

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients"

"Allocating poll threads for an interface/protocol combination with the NETTYPE configuration parameter" on page 2-47

Related reference:

"Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter" on page 2-48

"The DRDAEXEC thread and queries from clients" on page 2-48

"SQL and supported and unsupported data types" on page 2-49

"Display DRDA connection information" on page 2-50

"Display DRDA session information" on page 2-50

Configuring Informix for connections to IBM Data Server Clients

You must have installed an IBM Data Server Client and the applicable driver.

This topic explains how to configure IBM Informix for connections to IBM data server clients.

To configure Informix to connect to an IBM Data Server Client:

1. Configure a new server alias in the `sqlhosts` file or Windows registry, using either the **drtlitcp** or **drsoctcp** connection protocol.

   For example, for a new `sqlhosts` file entry for a DRDA connection, specify:

   ```
   #dbservername    nettype    hostname    servicename    options
   valley01_dr      drtlitcp   pdwest      9502
   ```

   If you are defining `sqlhosts` file entries for the Connection Manager, include entries for the **drtlitcp** or **drsoctcp** connection protocol, as shown in the following list:

```
#dbservername   nettype    hostname   servicename   options
ifx_primary     onsoctcp   ifx_host   ifx_port
ifx_dr          drsoctcp   ifx_host   ifx_drda_port

ifx_hdr         onsoctcp   hdr_host   hdr_port
ifx_hdr_dr      drsoctcp   hdr_host   hdr_drda_port

ifx_sds         onsoctcp   sds_host   sds_port
ifx_sds_dr      drsoctcp   sds_host   sds_drda_port
```

2. Verify that the onconfig file lists the DRDA connection as one of the server aliases.
3. Use the DRDA to connect to the server.

If you receive error -23104 when accessing the server through the DRDA protocol, the client application might be trying to bind a value that has an encoding different from the code page or code set of the database locale. Set the environment variable **GL_USEGLU** to 1 before you start the Informix instance. This enables the server to initialize the appropriate Unicode converters that are required to handle the code set conversions.

Related concepts:

"The sqlhosts file and the SQLHOSTS registry key" on page 2-17

"Connection Manager setup example" on page 23-23

"Overview of DRDA" on page 2-45

Related tasks:

"Allocating poll threads for an interface/protocol combination with the NETTYPE configuration parameter"

Related reference:

➡ The GL_USEGLU environment variable (GLS User's Guide)

➡ PDQPRIORITY environment variable (Guide to SQL: Reference)

"Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter" on page 2-48

"The DRDAEXEC thread and queries from clients" on page 2-48

"SQL and supported and unsupported data types" on page 2-49

"Display DRDA connection information" on page 2-50

"Display DRDA session information" on page 2-50

Allocating poll threads for an interface/protocol combination with the NETTYPE configuration parameter

The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. You can use this configuration parameter to allocate more than one poll thread for an interface/protocol combination.

Set the NETTYPE configuration parameter as follows:

1. Specify **SQLI**, **drtlitcp**, or **drsoctcp** as the connection protocol.
2. Add information about the number of poll threads, the number of connections, and the virtual processor class.

For example, specify:
```
NETTYPE drtlitcp,3,2,CPU
```

A NETTYPE entry can handle multiple database server aliases on the same protocol type. Thus, when DRDA is in use, the network listener thread (NETTYPE **drtlitcp** or **drsoctcp**) typically has at least two sockets open and listening for connections. One socket is open for SQLI connections and another is open for DRDA connections. Additional sockets might be open if many separate server aliases are configured.

For more information about the NETTYPE configuration parameter, see the *IBM Informix Administrator's Reference*.

Related concepts:

"Overview of DRDA" on page 2-45

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter

Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer. The minimum size is 4 KB, the maximum size is 2 megabytes, and the default value is 32 KB.

You can specify a one megabyte buffer as 1M, 1m, 1024K, 1024k, or 1024. IBM Informix automatically resets values that are less than 4 KB as 32 KB.

When a DRDA session is established, the session allocates a communication buffer of the current buffer size.

You can use the **isgetdrdacommbuffsize()** function to return the current value of DRDA_COMMBUFFSIZE.

You cannot use the **onmode -wm** command to change the setting while the database server is running.

Related concepts:

"Overview of DRDA" on page 2-45

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

The DRDAEXEC thread and queries from clients

For every DRDA client, IBM Informix creates a session and a DRDAEXEC thread, which is the equivalent of an SQLEXEC thread, to process and run the queries. This thread also formats the results of the queries in the DRDA protocol format and sends the results back to the client computer.

Queries issued from a DRDA client run in parallel if PDQPRIORITY is set and the query can run in parallel. Queries run from DRDAEXEC threads can also run in parallel.

Related concepts:
"Overview of DRDA" on page 2-45
Related tasks:
"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

SQL and supported and unsupported data types

When using DRDA, IBM Informix syntax is supported over the common API.

The following data types are supported over the common API:
- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BYTE
- CHAR(32k)
- CLOB
- DATE
- DATETIME
- DECIMAL
- FLOAT
- INT
- INT8
- INTERVAL
- LVARCHAR(32k)
- MONEY
- NCHAR(32k)
- NVARCHAR(255)
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT
- VARCHAR(255)

When using DRDA connections, Informix rounds decimal and money values to 32-digit precision for all data retrieval operations on decimal or money data types.

Informix DATETIME values are mapped to DATE, TIME, or TIMESTAMP values.

The following data types are supported for use with database server host variables:
- CHAR
- DATE
- INT
- SMALLINT
- VCHAR

Related concepts:

"Overview of DRDA" on page 2-45

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

Display DRDA connection information

Use the following **onstat** and **onmode** commands to display information that includes the DRDA thread name and an indicator that distinguishes SQLI and DRDA sessions:

- **onstat -g ses**
- **onstat -g sql**
- **onstat -g ath**
- **onstat -g stk**
- **onstat -u**
- **onstat -x**
- **onstat -G**
- **onstat -g ddr**
- **onstat -g env**
- **onstat -g stm**
- **onstat -g ssc**
- **onmode -D**
- **onmode -Z**

For example, the **onstat** output might show "drdaexec" as the threadname.

Related concepts:

"Overview of DRDA" on page 2-45

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

Display DRDA session information

Use the **syssesappinfo** table in the **sysmaster** database to view DRDA client session information. The table shows the client session ID, session application name, and a session value in the **sesapp_sid**, **sesapp_name**, and **sesapp_value** columns.

For example, the table might show the following information:

- **sesapp_sid**: 6
- **sesapp_name**: Accting
- **sesapp_value**: db2jcc_application

You can also display client session information using the **onstat -g ses** command.

Related concepts:

"Overview of DRDA" on page 2-45

➭ The sysmaster Database (Administrator's Reference)

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

Related reference:

➭ syssesappinfo (Administrator's Reference)

➭ onstat -g ses command: Print session-related information (Administrator's Reference)

Examples of client/server configurations

The next several sections of this chapter show the correct entries in the `sqlhosts` file or SQLHOSTS registry key for several client/server connections. You can assume that the network-configuration files `hosts` and `services` have been correctly prepared even if they are not explicitly mentioned. The following examples are included:

- Using a network connection
- Using multiple connection types
- Accessing multiple database servers

Examples of shared-memory and local-loopback connections can be found with the explanation of shared memory and local-loopback connections.

Related reference:

"Shared-memory connections (UNIX)" on page 2-6

"Local-loopback connections" on page 2-8

A network connection

The following figure shows a configuration in which the client application is on host **river** and the database server is on host **valley**.

sqlhosts entry on **river**

dbservername	nettype	hostname	servicename	options
valley_ds	onsoctcp	valley	valleyol	

SOC - TCP

TLI - TCP

sqlhosts entry on **valley**

dbservername	nettype	hostname	servicename	options
valley_ds	ontlitcp	valley	valleyol	
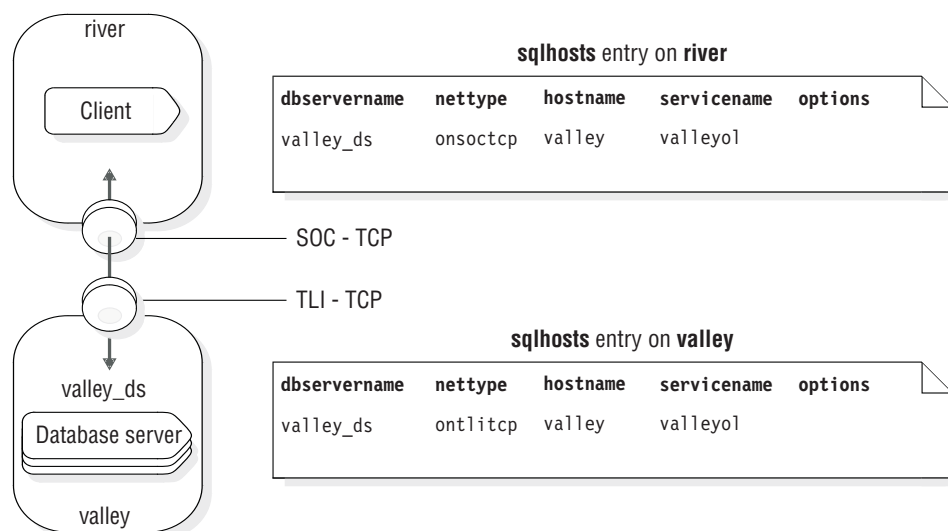
Figure 2-7. An example of a network client/server configuration

An entry for the **valley_ds** database server is in the `sqlhosts` files or SQLHOSTS registries on both computers. Each entry in the `sqlhosts` file or SQLHOSTS registry key on the computer where the database server is located has a corresponding entry on the computer where the client application is located.

UNIX only: Both computers are on the same TCP/IP network, but the host **river** uses sockets for its network programming interface, while the host **valley** uses TLI for its network programming interface. The **nettype** field must reflect the type of network programming interface used by the computer on which `sqlhosts` is located. In this example, the **nettype** field for the **valley_ds** database server on host **river** is `onsoctcp`, and the **nettype** field for the **valley_ds** database server on host **valley** is `ontlitcp`.

Related concepts:

"Network programming interface" on page 2-2

Multiple connection types

A single instance of the database server can provide more than one type of connection. The following figure illustrates such a configuration. The database server is on host **river**. Client A connects to the database server with a shared-memory connection because shared memory is fast. Client B must use a network connection because the client and server are on different computers.

When you want the database server to accept more than one type of connection, you must take the following actions:

- Add DBSERVERNAME and DBSERVERALIASES entries in the `onconfig` file.
- Add an entry to the `sqlhosts` file or SQLHOSTS registry key for each database server/connection type pair.

For the configuration in the following figure, the database server has two dbserver names: **river_net** and **river_shm**. The `onconfig` file includes the following entries:

```
DBSERVERNAME            river_net
DBSERVERALIASES         river_shm
```
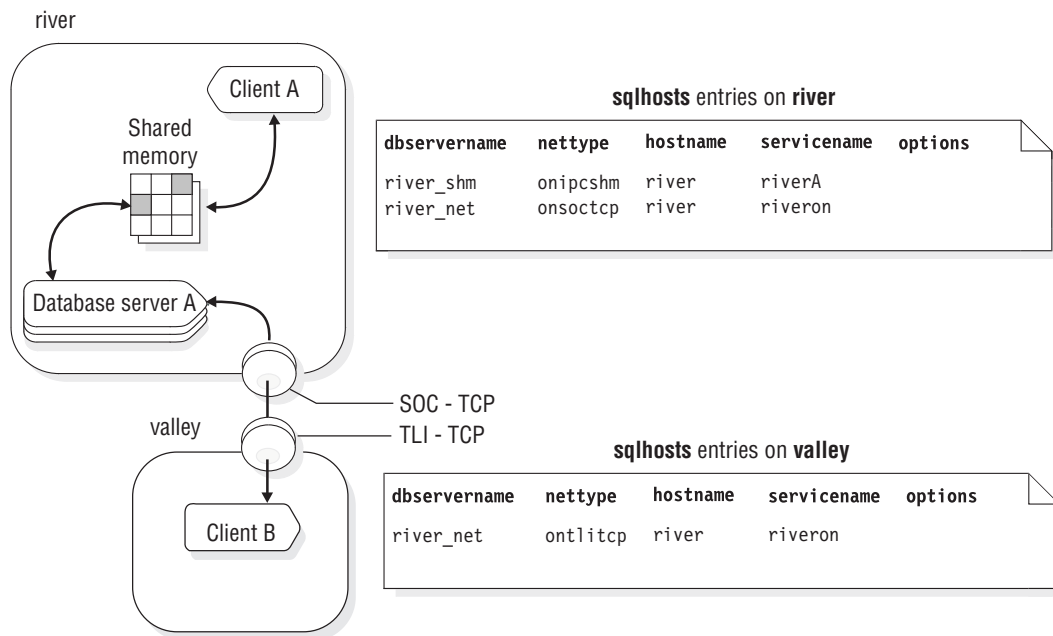
Figure 2-8. An example of a UNIX client/server configuration that uses multiple connection types

The dbserver name used by a client application determines the type of connection that is used. Client A uses the following statement to connect to the database server:

```
CONNECT TO '@river_shm'
```

In the `sqlhosts` file or SQLHOSTS registry key, the **nettype** associated with the name **river_shm** specifies a shared-memory connection, so this connection is a shared-memory connection.

Client B uses the following statement to connect to the database server:

```
CONNECT TO '@river_net'
```

In the `sqlhosts` file or registry, the **nettype** value associated with **river_net** specifies a network (TCP/IP) connection, so Client B uses a network connection.

Accessing multiple database servers

The following figure shows a configuration with two database servers on host **river**. When more than one database server is active on a single computer, it is known as *multiple residency*.

river

Shared memory   riverA_shm

Database server A

Client

SOC - TCP   riverB_soc

Database server B

sqlhosts entries on **river**

```
dbservername   nettype    hostname   servicename   options

riverA_shm     onipcshm   river      riverA
riverB_soc     onsoctcp   river      riveron
```
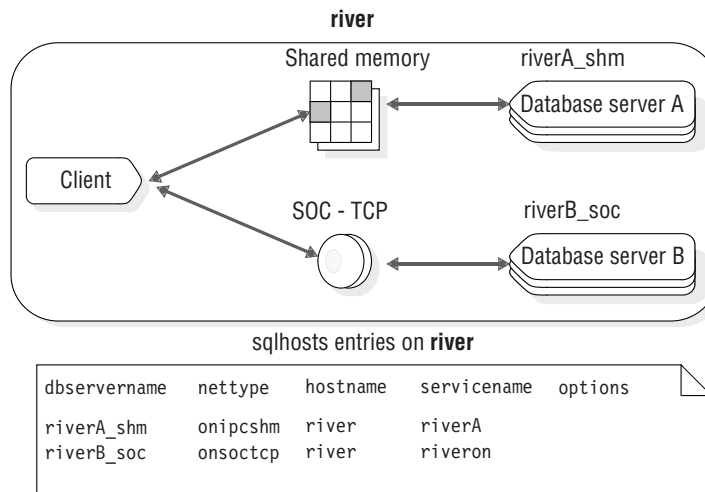
Figure 2-9. Multiple database servers on UNIX

For the configuration in previous example, you must prepare a onconfig file for server **A** and another one for database server **B**. The sqlhosts file or SQLHOSTS registry key includes the connectivity information for both database servers.

The onconfig file for database server **A** includes the following line:

```
DBSERVERNAME            riverA_shm
```

The onconfig file for database server **B** includes the following line:

```
DBSERVERNAME riverB_soc
```

Related tasks:

↪ Multiple residency (Installation Guide for Windows)

↪ Setting up multiple residency (Installation Guide for UNIX, Linux, and Mac OS X)

IBM Informix MaxConnect

IBM Informix MaxConnect is a networking product for IBM Informix database server environments on UNIX. Informix MaxConnect manages large numbers (from several hundred to tens of thousands) of client/server connections. Informix MaxConnect multiplexes connections so that the ratio of client connections to database connections can be 200:1 or higher. Informix MaxConnect increases system scalability to many thousands of connections and saves system resources, reducing response times and processor requirements. Informix MaxConnect is best for OLTP data transfers and should not be used for large multimedia data transfers.

Install Informix MaxConnect separately from your IBM Informix database server and client applications. For maximum performance benefit, install Informix MaxConnect either on a separate computer to which IBM Informix clients connect or on the client application server. You can install Informix MaxConnect in the following configurations:

- On a dedicated server to which IBM Informix clients connect
- On the client application server
- On the database server computer

Two protocols for multiplexing connections, **ontliimc** and **onsocimc**, are available for Informix MaxConnect users. You can use the **ontliimc** and **onsocimc** protocols in the following two configurations:

- To connect Informix MaxConnect to the database server.

  In this configuration, the client connections are multiplexed and use packet aggregation.

- To connect the client applications directly to the database server without going through Informix MaxConnect.

  In this configuration, the client does not get the benefits of connection multiplexing or packet aggregation. Choose this configuration when the client application is transferring simple- or smart-large-object data, because a direct connection to the database server is best.

For more information about how to configure Informix MaxConnect and monitor it with the **onstat -g imc** and **imcadmin** commands, see the *IBM Informix MaxConnect User's Guide*.

Important: Informix MaxConnect and the *IBM Informix MaxConnect User's Guide* ship separately from the IBM Informix database server.

Related reference:

"sqlhosts Connectivity information" on page 2-21

Chapter 3. Database server initialization

The database server requires both disk-space initialization and shared-memory initialization.

Types of initialization

Initialization of the database server is composed of two related activities: shared-memory initialization and disk-space initialization.

Shared-memory initialization, or starting the server, establishes the contents of database server shared memory as follows: internal tables, buffers, and the shared-memory communication area. Shared memory is initialized every time the database server starts. You use the **oninit** utility from the command line to initialize database server shared memory and bring the database server online.

Shared-memory initialization also occurs when you restart the database server.

One key difference distinguishes shared-memory initialization from disk-space initialization:

Shared-memory initialization has no effect on disk-space allocation or layout. No data is deleted.

Disk-space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk. When you initialize disk space, the database server automatically initializes shared memory as part of the process. Disk space is initialized the first time the database server starts. It is only initialized thereafter during a cold restore or at the request of the database server administrator.

Warning: When you initialize disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all the data in the earlier database server becomes inaccessible and, in effect, is deleted.

Initializing disk space

You initialize disk space for the root dbspace when you are starting a database server for the first time or you want to remove all dbspaces and their associated data. When you install the database server and choose to initialize a new instance of the database server, the database server is initialized automatically.

Warning: When you initialize the database server, all existing data in the database server disk space is deleted.

Prerequisites:
- **UNIX, Linux, or Mac OS X:** You must be logged in as user **root** or **informix**.
- **Windows:** You must be a member of the **Administrators** or **Power Users** group.

Before you reinitialize a root dbspace that is already being used by the database server:
- Back up existing data by performing a level-0 backup.
- Stop the database server by running the **onmode -k** command.

- Set the FULL_DISK_INIT configuration parameter to 1.

To initialize the database server:

UNIX, Linux, or Mac OS X: Run the **oninit -iy** command.

Windows: Use one of the following methods:
- In the **Services** control application, choose the database server service and type **-iy** in the Startup parameters field. Then click **Start**.
- Use the **starts** command from the command line with the database server name and the **-iy** options: `starts dbservername -iy`

After initialization is complete, you can perform a level-0 restore.

Related tasks:

Related reference:

⮕ The oninit utility (Administrator's Reference)

⮕ onmode -k, -m, -s, -u, -j: Change database server mode (Administrator's Reference)

⮕ FULL_DISK_INIT configuration parameter (Administrator's Reference)

Initialization process

When you start the database server or initialize disk space, the database server performs a set of steps. You can see the results of each step in the message log.

Disk-space initialization always includes the initialization of shared memory. However, some activities that normally take place during shared-memory initialization, such as recording configuration changes, are not required during disk initialization because those activities are not relevant with a newly initialized disk.

The following table shows the main tasks completed during the two types of initialization. The following sections explain each step.

Table 3-1. Initialization steps

Shared-memory initialization	Disk initialization
Process configuration file.	Process configuration file.
Create shared-memory segments.	Create shared-memory segments.
Initialize shared-memory structures.	Initialize shared-memory structures.
	Initialize disk space.
Start all required virtual processors.	Start all required virtual processors.
Make necessary conversions.	
Initiate fast recovery.	
Initiate a checkpoint.	Initiate a checkpoint.
Document configuration changes.	
Update `oncfg_servername.servernum` file.	Update `oncfg_servername.servernum` file.
Change to quiescent mode.	Change to quiescent mode.
Drop temporary tblspaces (optional).	

Table 3-1. Initialization steps (continued)

Shared-memory initialization	Disk initialization
Set forced residency, if requested.	Set forced residency, if specified.
Change to online mode and return control to user.	Change to online mode and return control to user.
If the SMI tables are not current, update the tables.	Create **sysmaster** database that contains the SMI tables.
	Create the **sysutils** database.
	Create the **sysuser** database
	Create the **sysadmin** database
Monitor maximum number of user connections at each checkpoint.	Monitor maximum number of user connections at each checkpoint.

Configuration file used during initialization

The database server uses configuration parameters to allocate shared-memory segments during initialization and restart. If you modify a shared-memory configuration parameter, you must shut down and restart the database server for the change to take effect.

The **ONCONFIG** environment variable, the onconfig file, and the onconfig.std template are stored in $INFORMIXDIR/etc on UNIX and %INFORMIXDIR%\etc on Windows. The default configuration files are intended as templates and not as functional configurations.

Always set the **ONCONFIG** environment variable, which specifies the onconfig file that contains your configuration parameters, before you initialize or restart the database server. Be sure that you also have the onconfig.std file. The server does not initialize if the onconfig.std file is missing.

During initialization, the database server looks for configuration values in the following files:

- If the **ONCONFIG** environment variable is set, the database server reads values from the onconfig file.

  If the **ONCONFIG** environment variable is set, but the database server cannot access the specified onconfig file, the server returns an error message.

- If the **ONCONFIG** environment variable is not set, the database server reads the values from the onconfig file.

If you omit any configuration parameters in your onconfig file, the database server reads the configuration values from the $INFORMIXDIR/etc/onconfig.std file.

The restart process compares the values in the current configuration file with the previous values, if any, that are stored in the root dbspace reserved page, PAGE_CONFIG. When differences exist, the database server uses the values from the current onconfig configuration file when the database server is restarted.

Create shared-memory portions

The database server uses the configuration values to calculate the required size of the database server resident shared memory. In addition, the database server computes additional configuration requirements from internal values. Space requirements for overhead are calculated and stored.

To create shared memory, the database server acquires the shared-memory space from the operating system for three different types of memory:

- Resident portion, used for data buffers and internal tables
- Virtual portion, used for most system and user-session memory requirements
- IPC communication portion, used for IPC communication

  The database server allocates this portion of shared memory only if you configure an IPC shared-memory connection.

Next, the database server attaches shared-memory segments to its virtual address space and initializes shared-memory structures. For more information about shared-memory structures, see "Virtual portion of shared memory" on page 6-13.

After initialization is complete and the database server is running, it can create additional shared-memory segments as necessary. The database server creates segments in increments of the page size.

Initialize or restart shared-memory

After the database server attaches to shared memory, it clears the shared-memory space of uninitialized data. Next the database server lays out the shared-memory header information and initializes data in the shared-memory structures. The database server lays out the space required for the logical-log buffer, initializes the structures, and links together the three individual buffers that form the logical-log buffer. For more information about these structures, see the **onstat** utility section in the *IBM Informix Administrator's Reference*.

After the database server remaps the shared-memory space, it registers the new starting addresses and sizes of each structure in the new shared-memory header.

During shared-memory initialization, disk structures and disk layout are not affected. The database server reads essential address information, such as the locations of the logical and physical logs, from disk and uses this information to update pointers in shared memory.

Initialize disk space

This procedure is performed only during disk-space initialization, not when the database server is restarted. After shared-memory structures are initialized, the database server begins initializing the disk. The database server initializes all the reserved pages that it maintains in the root dbspace on disk and writes page zero control information to the disk.

The FULL_DISK_INIT configuration parameter specifies whether **oninit -i** can run on your instance when a page zero exists at the root path location (at the first page of the first chunk location). Use this configuration parameter to prevent an accidental disk reinitialization of an existing server instance.

The default setting of the FULL_DISK_INIT configuration parameter is 0. When the configuration parameter is set to 0, the **oninit -i** command runs only if there is not a page zero at the root path location.

If a page zero exists at the root path location, initialization occurs only if the FULL_DISK_INIT configuration parameter is set to 1. The database server automatically resets the FULL_DISK_INIT configuration parameter to 0 after the initialization.

Related reference:

➟ The oninit utility (Administrator's Reference)

Start all required virtual processors

The database server starts all the virtual processors that it requires. The parameters in the onconfig file influence what processors are started. For example, the NETTYPE parameter can influence the number and type of processors started for making connections. For more information about virtual processors, see "Virtual processors" on page 4-1.

Make necessary conversions

The database server checks its internal files. If the files are from an earlier version, it updates these files to the current format. For information about database conversion, see the *IBM Informix Migration Guide*.

Start fast recovery

The database server checks if fast recovery is required and, if so, starts it. For more information about fast recovery, see "Fast recovery" on page 15-6.

Fast recovery is not performed during disk-space initialization because there is not yet anything to recover.

Start a checkpoint

After fast recovery completes, the database server executes a checkpoint to verify that all recovered transactions are flushed to disk so the transactions are not repeated.

As part of the checkpoint procedure, the database server writes a checkpoint-complete message in the message log. For more information about checkpoints, see "Checkpoints" on page 15-4.

The database server now moves to quiescent mode or online mode, depending on how you started the initialization or database-server restart process.

Document configuration changes

The database server compares the current values stored in the configuration file with the values previously stored in the root dbspace reserved page PAGE_CONFIG. When differences exist, the database server notes both values (old and new) in a message to the message log.

This task is not performed during disk-space initialization or restart.

Create the oncfg_servername.servernum file

The database server creates the `oncfg_servername.servernum` file and updates it every time that you add or delete a dbspace, blobspace, logical-log file, or chunk. You are not required to manipulate this file in any way, but you can see it listed in your $INFORMIXDIR/etc directory on UNIX or in your %INFORMIXDIR%\etc directory on Windows. The database server uses the `oncfg_servername.servernum` file during a full-system restore for salvaging the logical log.

For more information about the `oncfg_servername.servernum` file, see the section on files that the database server uses in the *IBM Informix Administrator's Reference*.

Drop Temporary Tblspaces

The database server searches through all dbspaces for temporary tblspaces. (If you use the **-p** option of **oninit** to initialize the database server, the database server skips this step.) These temporary tblspaces, if any, are tblspaces left by user processes that died prematurely and were unable to perform appropriate cleanup. The database server deletes any temporary tblspaces and reclaims the disk space. For more information about temporary tblspaces, see "Temporary tables" on page 8-27.

This task is performed when the database server is restarted; it is not performed during disk-space initialization.

Set forced residency if specified

If the value of the RESIDENT configuration parameter is `-1` or a number greater than 0, the database server tries to enforce residency of shared memory. If the host computer system does not support forced residency, the initialization procedure continues. Residency is not enforced, and the database server sends an error message to the message log. For more information about the RESIDENT configuration parameter, see the *IBM Informix Administrator's Reference*.

Return control to user

The database server writes the `IBM Informix Dynamic Server initialized - complete disk initialized` message in the message log only if initialization, not database-server restart, occurred. The database server also dynamically allocates a virtual shared-memory segment.

At this point, control returns to the user. Any error messages generated by the initialization procedure are displayed in the following locations:

- The command line
- The database server message log file, specified by the MSGPATH configuration parameter

  For more information about the MSGPATH parameter, see the *IBM Informix Administrator's Reference*.

- The **Summary** section of IBM Informix Server Administrator (ISA)

You can use the **oninit -w** utility to force the server to return to a command prompt within a configurable timeout. The **oninit -w** utility is useful for troubleshooting initialization failures. For syntax and information about **oninit**, see the *IBM Informix Administrator's Reference*.

Create sysmaster database and prepare SMI tables

Even though the database server has returned control to the user, it has not finished its work. The database server now checks the system-monitoring interface (SMI) tables. If the SMI tables are not current, the database server updates the tables. If the SMI tables are not present, as is the case when the disk is initialized, the database server creates the tables. After the database server builds the SMI tables, it puts the message `sysmaster database built successfully` in the message log. The database server also recreates the **sysmaster** database during conversion and reversion. For more information about SMI tables, see the chapter on the **sysmaster** database in the *IBM Informix Administrator's Reference*.

If you shut down the database server before it finishes building the SMI tables, the process of building the tables stops. This condition does not damage the database server. The database server builds the SMI tables the next time that you bring the database server online. However, if you do not allow the SMI tables to finish building, you cannot run any queries against those tables, and you cannot use ON-Bar for backups.

After the SMI tables have been created, the database server is ready for use. The database server runs until you stop it or, possibly, until a malfunction.

Recommendation: Do not try to stop the database server by stopping a virtual processor or ending another database server process. For more information, see "Start and stop virtual processors" on page 5-2.

Create the sysutils database

The database server drops and recreates the **sysutils** database during disk initialization, conversion, or reversion. ON-Bar stores backup and restore information in the **sysutils** database. Wait until the message `sysutils database built successfully` displays in the message log. For more information, see the *IBM Informix Backup and Restore Guide*.

Create the sysuser database

The **sysuser** database is used for Pluggable Authentication Module (PAM) authentication in IBM Informix server to server communication.

Create the sysadmin database

The **sysadmin** database provides remote administration and scheduler API features in IBM Informix.

Monitor maximum number of user connections

At each checkpoint, the database server prints the maximum number of user connections in the message log: `maximum server connections` *number*. You can monitor the number of users who have connected to the database server since the last restart or disk initialization.

The number displayed is reset when the customer reinitializes the database server.

Database server operating modes

You can determine the current database server mode by running the **onstat** utility from the command line. The **onstat** header displays the mode.

The table shows the principal modes of operation of the database server.

Table 3-2. Operating modes

Operating mode	Description	Users allowed access
Offline mode	The database server is not running. Shared memory is not allocated.	Only the administrator (user **informix**) can change from this mode to another mode.
Quiescent mode	Database-server processes are running and shared-memory resources are allocated. Administrators use this mode to perform maintenance functions that do not require the execution of SQL and DDL statements.	Only the administrator (user **informix**) can access the database server. Other users can view database-server status information, but they cannot access the database server.

Administration mode	This mode is an intermediary mode between Quiescent mode and Online mode.
	
	Administrators use this mode to perform any maintenance task, including tasks requiring the execution of SQL and DDL statements. Administrators can also perform all other functions available in Online mode.
		The following users can connect to the database server in administration mode:
		• User **informix**
		• Users who have the DBSA role
		  Set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to 1 if you want users who are members of the DBSA group (in addition to user **informix**) to connect to the database server in administration mode.
		• One or more users who have administration mode access
		  User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode through the **onmode -j** command, the **oninit -U** command, or the ADMIN_MODE_USERS configuration parameter.
		
		Other users can view database-server status information, but they cannot access the database server.

Online mode	This is the normal operating mode of the database server.	Any authorized user can connect with the database server and perform all database activities. User **informix** or user **root** can use the command-line utilities to change many database server ONCONFIG parameter values.

In addition, the database server can also be in one of the following modes:
• *Read-only mode* is used by the secondary database server in a data replication environment. An application can query a secondary database server that is in read-only mode, but the application cannot write to a read-only database.

- *Recovery mode* is transitory. It occurs when the database server performs fast recovery or recovers from a system archive or system restore. Recovery occurs during the change from offline to quiescent mode.
- *Shutdown mode* is transitory. It occurs when the database server is moving from online to quiescent mode or from online (or quiescent) to offline mode. For the current users access the system, but no new users are allowed access.

  After shutdown mode is initiated, it cannot be canceled.

Related tasks:

"Starting the database server" on page 1-12

Change database server operating modes

This section describes how to change from one database server operating mode to another with the **oninit** and **onmode** utilities. It also contains information about using the ADMIN_MODE_USERS configuration parameter to specify which users can connect to the server in administration mode.

This section describes how to change from one database server operating mode to another with the **oninit** and **onmode** utilities and with ISA. It also contains information about using the ADMIN_MODE_USERS configuration parameter to specify which users can connect to the server in administration mode.

Windows only: In Windows, the database server runs as a service. Windows provides a service control application (also called the Services tool) to start, stop, and pause a service. The service control application is located in the Control Panel program group. The service name for the database server includes the database server name (the value of DBSERVERNAME in the `onconfig` file). For example, the IBM Informix service for the `newyork` database server is:

```
IBM Informix Database Server - newyork
```

To change mode with the Services tool, start the tool and select the database server service. Then choose the appropriate option in the Services window. The tables shown later in these topics explain which option you select for each mode.

To start and stop the database server, you can use other Windows tools, such as the NET command and the Server Manager tool. For more information about these methods, consult your Windows operating-system documentation.

Tip: After you change the mode of your database server, run the **onstat** command to verify the current server status.

Users permitted to change modes
UNIX only

Users who are logged in as **root** or **informix** and members of the DBSA group can change the operating mode of the database server.

If you want users with the DBSA group to connect to the database server in administration mode, set the `ADMIN_USER_MODE_WITH_DBSA` configuration parameter to 1. If this parameter is set to zero, then access is restricted to user **informix** only. If the parameter is missing from $ONCONFIG, it is treated as 0.

User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode, using the **onmode** utility, the **oninit** utility, or the `ADMIN_MODE_USERS` configuration parameter.

Note: For a member of the DBSA group, the permissions on `$INFORMIXDIR/bin/oninit` must be changed to allow **public execute permission - root:informix:6755** in a standard IBM Informix installation.

Windows only

Table 3-2 on page 3-8 shows which users can change the operating mode of the database server in Windows. Apache as user **informix**. The Apache server runs as a member of the **Informix-Admin** group.

Table 3-2 on page 3-8 shows which users can change the operating mode of the database server in Windows. If you use ISA, you can log-in to the operating system as any user but you must log-in to Apache as user **informix**. The Apache server runs as a member of the **Informix-Admin** group.

Table 3-3. Changing operating modes in windows

Changing operating mode	Administrators group	Informix-Admin group
command-line utilities such as **starts**		X
ISA		X
services control panel	X	

ISA options for changing modes

You can use IBM Informix Server Administrator (ISA) to change the database server mode. For more information, see the ISA online help.

Action	ISA option
Initialize the database server.	**Mode > Online (Initialize Disks)** or **Quiescent (Initialize Disks)**
Change from offline or administration to quiescent.	**Mode > Quiescent**
Change from any mode to administration.	**Mode > Single-User**
Change from offline, quiescent, or administration to online.	**Mode > Online**
Change gracefully from online to quiescent.	**Mode > Quiescent (Graceful)**
Change immediately from online to quiescent.	**Mode > Quiescent (Immediate)**
Shut down the database server.	**Mode > Offline**

ON-Monitor options for changing modes (UNIX)

You can use ON-Monitor to change the database server mode on UNIX. For more information, see the topics about ON-Monitor in the *IBM Informix Administrator's Reference*.

Action	Menu option
Initialize the database server.	**Parameters > Initialize**
Change from offline to quiescent.	**Mode > Startup**
Change from offline or quiescent to online.	**Mode > Online**

Action	Menu option
Change gracefully from online to quiescent.	**Mode** > **Graceful Shutdown**
Change immediately from online to quiescent.	**Mode** > **Immediate Shutdown**
Change to administration mode.	**Mode** > **Single User**
Shut down the database server.	**Mode** > **Take Offline**

Command-line options for changing modes

Table 3-2 on page 3-8 contains descriptions of each mode and shows which users can access the database server when the server is in each mode. These topics contain information about commands for changing modes and information about how mode changes effect user sessions.

Also see "Specify administration mode users with the ADMIN_MODE_USERS configuration parameter" on page 3-14.

Change from offline to quiescent mode

When the database server changes from offline mode to quiescent mode, the database server initializes shared memory. Only administrators can access the database server to perform maintenance functions that do not involve the execution of SQL and DDL statements.

Operating system	Action
UNIX	Run **oninit -s**.
Windows	On the command line, use the **starts dbservername -s** command.

Change from offline to online mode

When you move the database server from offline to online mode, the database server initializes shared memory and is available for all user sessions.

Operating system	Action
UNIX	Run **oninit**.
Windows	With the Services tool, select the database server service and click **Start**. On the command line, use the **starts** *dbservername* command.

Change from offline to administration mode

When you move the database server from offline to administration mode, you move the server into a mode that only administrators can use to perform database server functions and maintenance functions, including those involving the execution of SQL and DDL statements.

Operating system	Action
UNIX or Windows	Run **oninit -j**.

User **informix** or a DBSA can use the **oninit -U** command to specify a list of administration mode users, as shown in this example:

```
oninit -U mark,ajay,carol
```

Users specified in the **oninit -U** list can connect for the period of time in which the server instance is active or until you run the **onmode -j -U** command to change the list of users who can connect to the server. Run the **onmode -j -U** command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Also see "Specify administration mode users with the ADMIN_MODE_USERS configuration parameter" on page 3-14.

Change from quiescent to online mode

When you take the database server from quiescent mode to online mode, all sessions gain access.

If you have already taken the database server from online mode to quiescent mode and you are now returning the database server to online mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.

Operating system	Action
UNIX or Windows	Run **onmode -m**.
Windows only	With the Services tool, choose the database server service and click **Continue**.

Change gracefully from online to quiescent mode

Take the database server gracefully from online mode to quiescent mode to restrict access to the database server without interrupting current processing.

After you perform this task, the database server sets a flag that prevents new sessions from gaining access to the database server. The current sessions are allowed to finish processing.

After you initiate the mode change, it cannot be canceled. During the mode change from online to quiescent, the database server is considered to be in Shutdown mode.

Operating system	Action
UNIX or Windows	Run **onmode -s** or **onmode -sy**.
Windows only	With the Services tool, choose the database server service and click **Pause**.

Change immediately from online to quiescent mode

Take the database server immediately from online mode to quiescent mode to restrict access to the database server as soon as possible. Work in progress can be lost.

A prompt asks for confirmation of the immediate shutdown. If you confirm, the database server sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates the session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

The database server cleans up all sessions that the database server terminated. Active transactions are rolled back.

Operating system	Action
UNIX or Windows	Run **onmode -u** or **onmode -uy**

The **-y** option eliminates the requirement to confirm the prompt. |

Change from quiescent or online to administration mode

When you move the database server from quiescent or online to administration mode, you move the server into a mode that only administrators can use.

If you begin in online mode, the database server automatically disconnects any users who are connected with any user ID that is not user **informix** and the users receive an error message. If a connection is terminated during a transaction, the database server rolls back the transaction.

Change to administration mode when you want to run SQL and DLL commands when no other users are connected.

Operating system	Action
UNIX or Windows	Run **onmode -j**.

User **informix** or a DBSA can use the **onmode -j -U** option to grant individual users access to the database server in administration mode.

For example, run the following command to enable three individual users to connect to the database server and have database server access until the database server mode changes to offline, quiescent or online mode:

```
onmode -j -U mark,ajay,carol
```

After connecting, these individual users can run any SQL or DDL commands. When the server is changed to administration mode, all sessions for users not identified in the **onmode -j -U** command lose their database server connection.

After initially running the **onmode -j -U** command, you can remove individuals by running **onmode -j -U** and removing individual user names from the new list of names in the command, for example, by running:

```
onmode -j -U mark,carol
```

Run the **onmode -j -U** command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Also see "Specify administration mode users with the ADMIN_MODE_USERS configuration parameter" on page 3-14.

Change from administration to online mode

When you move the database server from administration to online mode, all users can access the database server.

Operating system	Action
UNIX or Windows	Run **onmode -m**.

Change from administration to quiescent mode

When you move the database server from administration to quiescent mode, you move the server into a mode that only administrators can use to perform maintenance functions that do not involve the execution of SQL and DDL statements.

Operating system	Action
UNIX or Windows	Run **onmode -s**.

Change from any mode immediately to offline mode

You can take the database server immediately from any mode to offline mode.

A prompt asks for confirmation to go offline. If you confirm, the database server initiates a checkpoint request and sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates this session.

The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated.

After you take the database server to offline mode, restart the database server in quiescent, administration, or online mode. When you restart the database server, it performs a fast recovery to ensure that the data is logically consistent.

The database server cleans up all sessions that were terminated by the database server. Active transactions are rolled back.

Operating system	Action
UNIX or Windows	Run **onmode -k** or **onmode -ky**. The **-y** option eliminates the automatic prompt that confirms an immediate shutdown.
Windows only	With the Services tool, choose the database server service and click **Stop**.

If the **onmode** command fails to shut down the database server, you can use the **onclean** utility to force an immediate shutdown. For more information about the **onclean** utility, see the *IBM Informix Administrator's Reference*.

Specify administration mode users with the ADMIN_MODE_USERS configuration parameter

The ADMIN_MODE_USERS configuration parameter enables you to specify which users can connect to the database server in administration mode. Unlike the **oninit** and **onmode** commands that enable you to specify administration mode users until the server changes to offline, quiescent, or online mode, the ADMIN_MODE_USERS configuration parameter preserves a list of administration mode users indefinitely.

To create a list of administration mode users that is preserved in the `onconfig` file, specify a comma-separated list of users as ADMIN_MODE_USERS configuration parameter values, for example, `mark,ajay,carol`.

To override ADMIN_MODE_USERS during a session, use the **onmode -wf** command, as shown in this example:

```
onmode -wf ADMIN_MODE_USERS=sharon,kalpana
```

The effect of the ADMIN_MODE_USERS configuration parameter is to add to the list of people permitted to access the server in administration mode. Those people listed in the **onmode** command line override those listed in the `onconfig` file.

Part 2. Disk, memory, and process management

Chapter 4. Virtual processors and threads

These topics describe virtual processors, explain how threads run within the virtual processors, and explain how the database server uses virtual processors and threads to improve performance.

Virtual processors

Database server processes are called *virtual processors* because the way they function is similar to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a database server virtual processor runs multiple threads to service multiple SQL client applications.

A virtual processor is a process that the operating system schedules for processing.

The following figure illustrates the relationship of client applications to virtual processors. A small number of virtual processors serve a much larger number of client applications or queries.
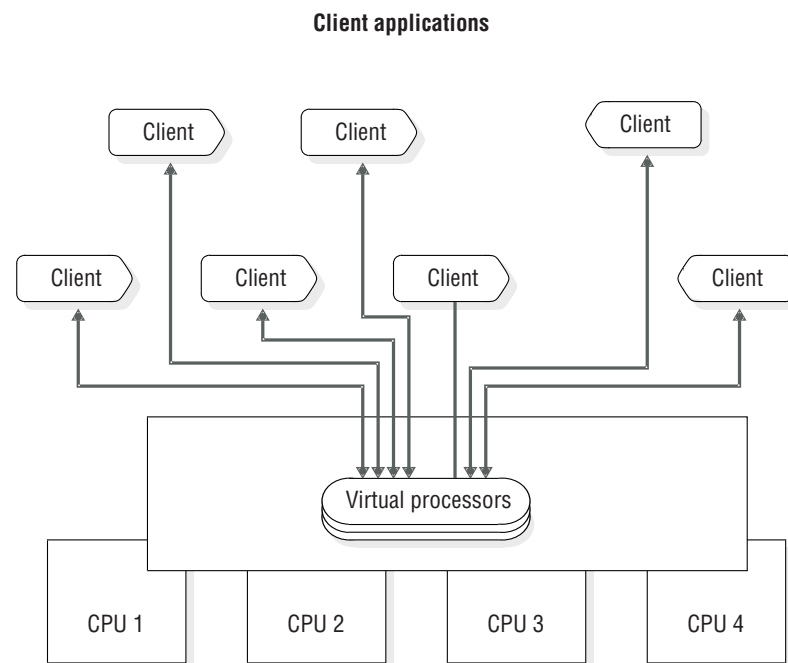
Figure 4-1. Virtual processors

Threads

A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU. The virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that the virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes, but they make fewer demands on the operating system.

Database server virtual processors are multithreaded because they run multiple concurrent threads.

The nature of threads is as follows.

Operating system	Action
UNIX	A thread is a task that the virtual processor schedules internally for processing.
Windows	A thread is a task that the virtual processor schedules internally for processing. Because the virtual processor is implemented as a Windows thread, database server threads run within Windows threads.

Important: Throughout these topics, all references to *thread* refer to the threads created, scheduled, and deleted by the database server. All references to "Windows threads" refer to the threads created, scheduled, and deleted by Windows.

A virtual processor runs threads on behalf of SQL client applications (*session* threads) and also to satisfy internal requirements (*internal* threads). In most cases, for each connection by a client application, the database server runs one session thread. The database server runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks. For cases in which the database server runs multiple session threads for a single client, see "Parallel processing" on page 4-5.

A *user thread* is a database server thread that services requests from client applications. User threads include session threads, called **sqlexec** threads, which are the primary threads that the database server runs to service client applications.

User threads also include a thread to service requests from the **onmode** utility, threads for recovery, B-tree scanner threads, and page-cleaner threads.

To display active user threads, use **onstat -u**. For more information about monitoring sessions and threads, see *IBM Informix Performance Guide*.

Types of virtual processors

Each class of virtual processor is dedicated to processing certain types of threads.

The following table shows the classes of virtual processors and the types of processing that they do.

The number of virtual processors of each class that you configure depends on the availability of physical processors (CPUs), hardware memory, and the database applications in use.

Table 4-1. Virtual-processor classes

Virtual-processor class	Category	Purpose
ADM	Administrative	Performs administrative functions.
ADT	Auditing	Performs auditing functions.

Table 4-1. Virtual-processor classes (continued)

Virtual-processor class	Category	Purpose
AIO	Disk I/O	Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
BTS	Basic text searching	Runs basic text search index operations and queries.
CPU	Central processing	Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O (KAIO) where available. Can run a single poll thread, depending on configuration.
CSM	Communications Support Module	Performs communications support service operations.
dwavp	Data warehousing	Runs the administrative functions and procedures for Informix Warehouse Accelerator on a database server that is connected to Informix Warehouse Accelerator.
Encrypt	Encryption	Used by the database server when encryption or decryption functions are called.
IDSXMLVP	XML publishing	Runs XML publishing functions.
Java VP (JVP)	Java UDR	Runs Java UDRs. Contains the Java Virtual Machine (JVM).
LIO	Disk I/O	Writes to the logical-log files (internal class) if they are in cooked disk space.
MQ	MQ messaging	Performs MQ messaging transactions.
MSC	Miscellaneous	Services requests for system calls that require a very large stack.
OPT (UNIX)	Optical	Performs I/O to optical disk.
PIO	Disk I/O	Writes to the physical-log file (internal class) if it is in cooked disk space.
SHM	Network	Performs shared memory communication.
SOC	Network	Uses sockets to perform network communication.
TLI	Network	Uses the Transport Layer Interface (TLI) to perform network communication.
WFSVP	Web feature service	Runs web feature service routines.
classname	User defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected. Specified with the VPCLASS configuration parameter. You must specify *classname*.

The following figure illustrates the major components and the extensibility of the database server.
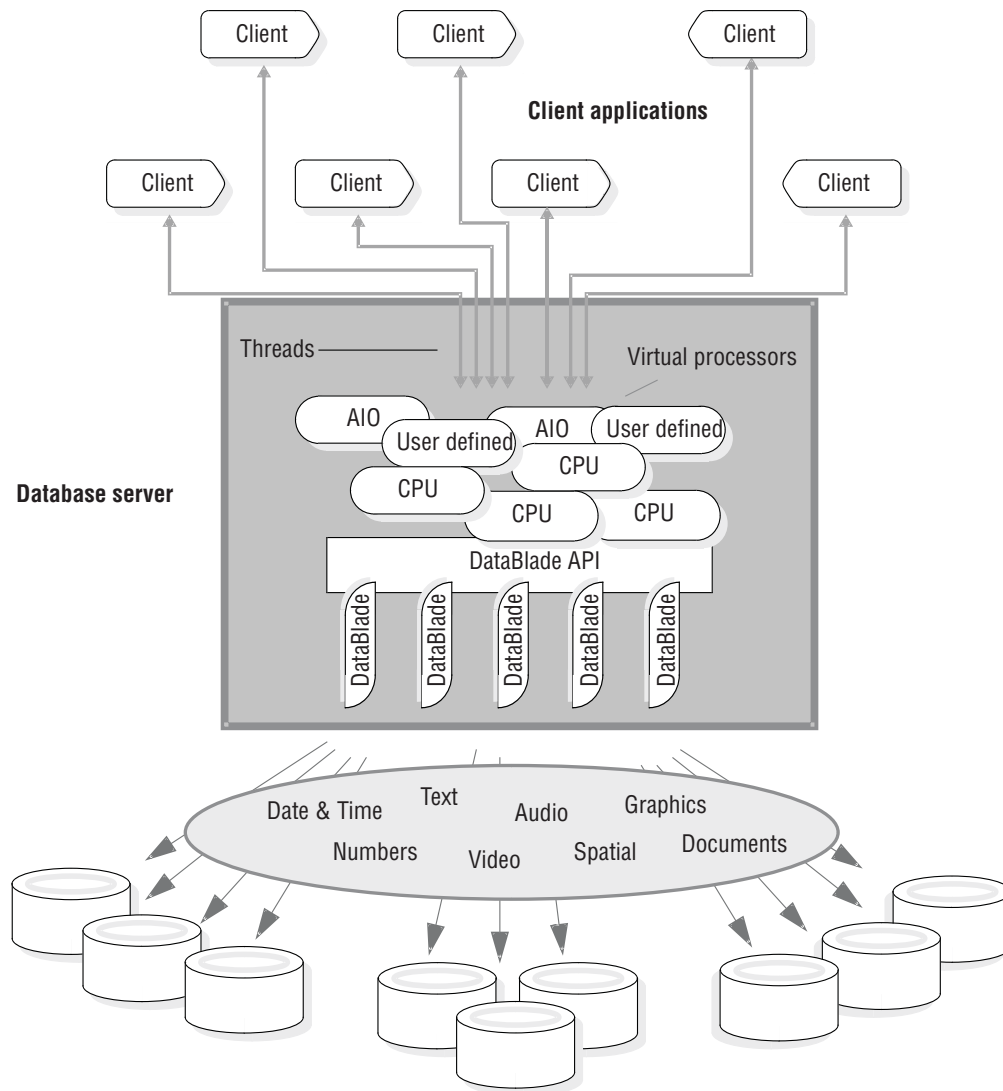
Figure 4-2. Database server

Advantages of virtual processors

Compared to a database server process that services a single client application, the dynamic, multithreaded nature of a database server virtual processor provides the following advantages:

- Virtual processors can share processing.
- Virtual processors save memory and resources.
- Virtual processors can perform parallel processing.
- You can start additional virtual processors and terminate active CPU virtual processors while the database server is running.
- You can bind virtual processors to CPUs.

The following topics describe these advantages.

Shared processing

Virtual processors in the same class have identical code and share access to both data and processing queues in memory. Any virtual processor in a class can run any thread that belongs to that class.

Generally, the database server tries to keep a thread running on the same virtual processor because moving it to a different virtual processor can require some data from the memory of the processor to be transferred on the bus. When a thread is waiting to run, however, the database server can migrate the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the data.

Shared processing within a class of virtual processors occurs automatically and is transparent to the database user.

Save memory and resources

The database server is able to service many clients with a small number of server processes compared to architectures that have one client process to one server process. It does so by running a thread, rather than a process, for each client.

Multithreading permits more efficient use of the operating-system resources because threads share the resources allocated to the virtual processor. All threads that a virtual processor runs have the same access to the virtual-processor memory, communication ports, and files. The virtual processor coordinates access to resources by the threads. Individual processes, though, each have a distinct set of resources, and when multiple processes require access to the same resources, the operating system must coordinate the access.

Generally, a virtual processor can switch from one thread to another faster than the operating system can switch from one process to another. When the operating system switches between processes, it must stop one process from running on the processor, save its current processing state (or context), and start another process. Both processes must enter and exit the operating-system kernel, and the contents of portions of physical memory might require replacement. Threads, though, share the same virtual memory and file descriptors. When a virtual processor switches from one thread to another, the switch is from one path of execution to another. The virtual processor, which is a process, continues to run on the CPU without interruption. For a description of how a virtual processor switches from one thread to another, see "Context switching" on page 4-7.

Parallel processing

In the following cases, virtual processors of the CPU class can run multiple session threads, working in parallel, for a single client:
• Index building
• Sorting
• Recovery
• Scanning
• Joining
• Aggregation
• Grouping
• User-defined-routine (UDR) execution

For more information about parallel UDR execution, see *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

The following figure illustrates parallel processing. When a client initiates index building, sorting, or logical recovery, the database server creates multiple threads to work on the task in parallel, using as much of the computer resources as possible. While one thread is waiting for I/O, another can be working.
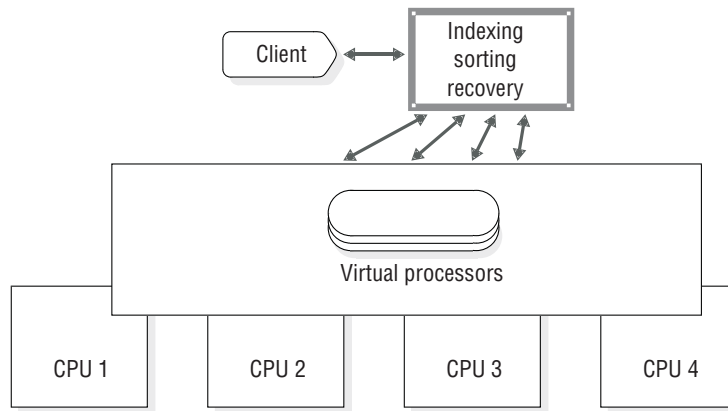
Figure 4-3. Parallel processing

Add and drop virtual processors in online mode

You can add virtual processors to meet increasing demands for service while the database server is running. For example, if the virtual processors of a class become compute bound or I/O bound (meaning that CPU work or I/O requests are accumulating faster than the current number of virtual processors can process them), you can start additional virtual processors for that class to distribute the processing load further.

You can add virtual processors for any of the classes while the database server is running. For more information, see "Add virtual processors in online mode" on page 5-3.

Windows only: In Windows, you cannot drop a virtual processor of any class.

While the database server is running, you can drop virtual processors of the CPU or a user-defined class. For more information, see "Set virtual-processor configuration parameters" on page 5-1.

Bind virtual processors to CPUs

You can use some multiprocessor systems to bind a process to a particular CPU. This feature is called *processor affinity*.

On multiprocessor computers for which the database server supports processor affinity, you can bind CPU virtual processors to specific CPUs in the computer. When you bind a CPU virtual processor to a CPU, the virtual processor runs exclusively on that CPU. This operation improves the performance of the virtual processor because it reduces the amount of switching between processes that the operating system must do. Binding CPU virtual processors to specific CPUs also enables you to isolate database work on specific processors on the computer, leaving the remaining processors free for other work. Only CPU virtual processors can be bound to CPUs.

For information about how to assign CPU virtual processors to hardware processors, see "Processor affinity" on page 4-13.

How virtual processors service threads

At a given time, a virtual processor can run only one thread. A virtual processor services multiple threads concurrently by switching between them. A virtual processor runs a thread until it yields. When a thread yields, the virtual processor switches to the next thread that is ready to run. The virtual processor continues this process, eventually returning to the original thread when that thread is ready to continue. Some threads complete their work, and the virtual processor starts new threads to process new work. Because a virtual processor continually switches between threads, it can keep the CPU processing continually. The speed at which processing occurs produces the appearance that the virtual processor processes multiple tasks simultaneously and, in effect, it does.

Running multiple concurrent threads requires scheduling and synchronization to prevent one thread from interfering with the work of another. Virtual processors use the following structures and methods to coordinate concurrent processing by multiple threads:

- Control structures
- Context switching
- Stacks
- Queues
- Mutexes

These topics describe how virtual processors use these structures and methods.

Control structures

When a client connects to the database server, the database server creates a *session* structure, which is called a *session control block*, to hold information about the connection and the user. A session begins when a client connects to the database server, and it ends when the connection terminates.

Next, the database server creates a thread structure, which is called a *thread-control block* (TCB) for the session, and initiates a primary thread (**sqlexec**) to process the client request. When a thread *yields*—that is, when it pauses and allows another thread to run—the virtual processor saves information about the state of the thread in the thread-control block. This information includes the content of the process system registers, the program counter (address of the next instruction to execute), and the stack pointer. This information constitutes the context of the thread.

In most cases, the database server runs one primary thread per session. In cases where it performs parallel processing, however, it creates multiple session threads for a single client, and, likewise, multiple corresponding thread-control blocks.

Context switching

A virtual processor switches from running one thread to running another one by *context switching*. The database server does not preempt a running thread, as the operating system does to a process, when a fixed amount of time (time-slice) expires. Instead, a thread yields at one of the following points:

- A predetermined point in the code
- When the thread can no longer execute until some condition is met

When the amount of processing required to complete a task would cause other threads to wait for an undue length of time, a thread yields at a predetermined

point. The code for such long-running tasks includes calls to the yield function at strategic points in the processing. When a thread performs one of these tasks, it yields when it encounters a yield function call. Other threads in the ready queue then get a chance to run. When the original thread next gets a turn, it resumes executing code at the point immediately after the call to the yield function. Predetermined calls to the yield function allow the database server to interrupt threads at points that are most advantageous for performance.

A thread also yields when it can no longer continue its task until some condition occurs. For example, a thread yields when it is waiting for disk I/O to complete, when it is waiting for data from the client, or when it is waiting for a lock or other resource.

When a thread yields, the virtual processor saves its context in the thread-control block. Then the virtual processor selects a new thread to run from a queue of ready threads, loads the context of the new thread from its thread-control block, and begins executing at the new address in the program counter. The following figure illustrates how a virtual processor accomplishes a context switch.

Thread-control blocks

t0	prgm ctr
registers	
stack ptr	
etc.	

t1	prgm ctr
registers	
stack ptr	
etc.	

Save   Restore

Time

Virtual processor

Thread t0   Context switch   Thread t1
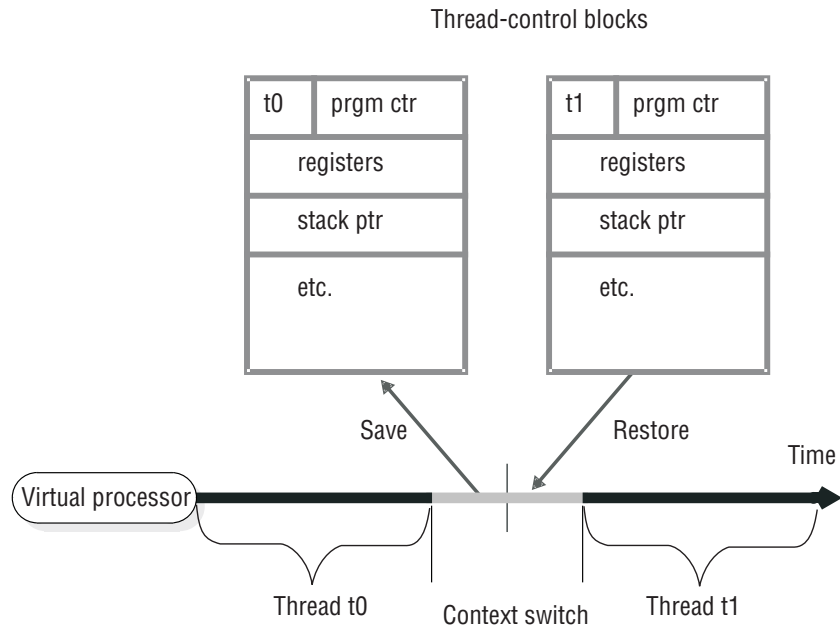
Figure 4-4. Context switch: how a virtual processor switches from one thread to another

Stacks

The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*. For information about how to set the size of the stack, see "Stacks" on page 6-17.

The stack enables a virtual processor to protect the nonshared data of a thread from being overwritten by other threads that concurrently execute the same code. For example, if several client applications concurrently perform SELECT statements, the session threads for each client execute many of the same functions in the code. If a thread did not have a private stack, one thread might overwrite local data that belongs to another thread within a function.

When a virtual processor switches to a new thread, it loads a stack pointer for that thread from a field in the thread-control block. The stack pointer stores the beginning address of the stack. The virtual processor can then specify offsets to the beginning address to access data within the stack. The figure illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.
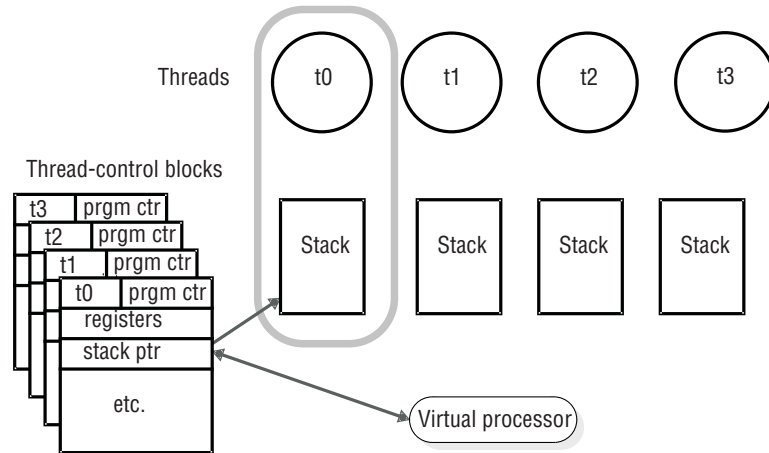
Figure 4-5. Virtual processors segregate nonshared data for each user

Queues

The database server uses three types of queues to schedule the processing of multiple, concurrently running threads.

Virtual processors of the same class share queues. This fact, in part, enables a thread to migrate from one virtual processor in a class to another when necessary.

Ready queues

Ready queues hold threads that are ready to run when the current (running) thread yields. When a thread yields, the virtual processor picks the next thread with the appropriate priority from the ready queue. Within the queue, the virtual processor processes threads that have the same priority on a first-in-first-out (FIFO) basis.

On a multiprocessor computer, if you notice that threads are accumulating in the ready queue for a class of virtual processors (indicating that work is accumulating faster than the virtual processor can process it), you can start additional virtual processors of that class to distribute the processing load. For information about how to monitor the ready queues, see "Monitor virtual processors" on page 5-4. For information about how to add virtual processors while the database server is in online mode, see "Add virtual processors in online mode" on page 5-3.

Sleep queues

Sleep queues hold the contexts of threads that have no work to do at a particular time. A thread is put to sleep either for a specified period of time or forever.

The administration class (ADM) of virtual processors runs the system timer and special utility threads. Virtual processors in this class are created and run automatically. No configuration parameters affect this class of virtual processors.

The ADM virtual processor wakes up threads that have slept for the specified time. A thread that runs in the ADM virtual processor checks on sleeping threads at

one-second intervals. If a sleeping thread has slept for its specified time, the ADM virtual processor moves it into the appropriate ready queue. A thread that is sleeping for a specified time can also be explicitly awakened by another thread.

A thread that is sleeping forever is awakened when it has more work to do. For example, when a thread that is running on a CPU virtual processor must access a disk, it issues an I/O request, places itself in a sleep queue for the CPU virtual processor, and yields. When the I/O thread notifies the CPU virtual processor that the I/O is complete, the CPU virtual processor schedules the original thread to continue processing by moving it from the sleep queue to a ready queue. The following figure illustrates how the database server threads are queued to perform database I/O.
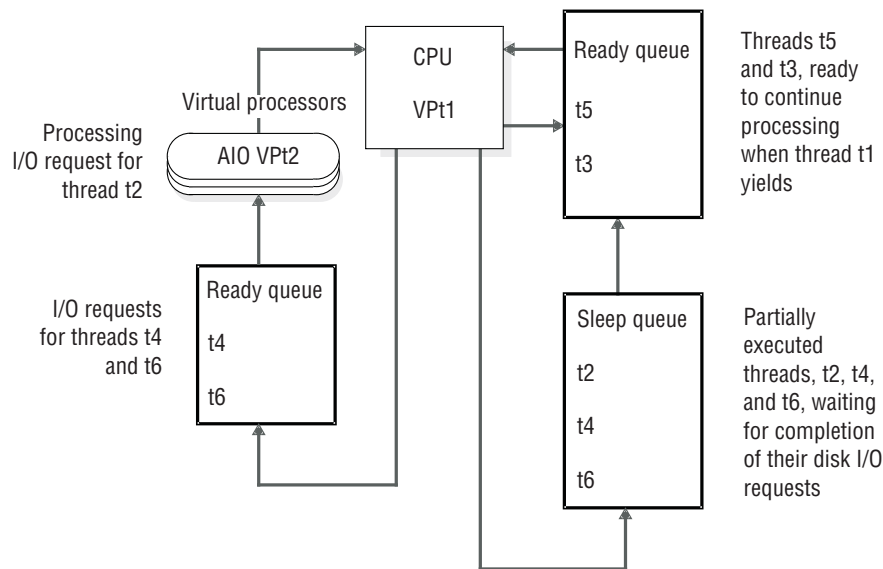
Figure 4-6. How database server threads are queued to perform database I/O

Wait queues
Wait queues hold threads that must wait for a particular event before they can continue to run. For example, wait queues coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch but finds that the latch is held by another user, the thread that was denied access puts itself in the logical-log wait queue. When the thread that owns the lock is ready to release the latch, it checks for waiting threads, and, if threads are waiting, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mut*ually *ex*clusive), also called a *latch*, is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources. Mutexes are similar to semaphores, which some operating systems use to regulate access to shared data by multiple processes. However, mutexes permit a greater degree of parallelism than semaphores.

A mutex is a variable that is associated with a shared resource such as a buffer. A thread must acquire the mutex for a resource before it can access the resource. Other threads are excluded from accessing the resource until the owner releases it. A thread acquires a mutex, after a mutex becomes available, by setting it to an

in-use state. The synchronization that mutexes provide ensures that only one thread at a time writes to an area of shared memory.

For information about monitoring mutexes, see "Monitor the shared-memory profile and latches" on page 7-9.

Virtual-processor classes

A virtual processor of a given class can run only threads of that class. These topics describe the types of threads, or the types of processing, that each class of virtual processor performs. They also explain how to determine the number of virtual processors that you must run for each class.

CPU virtual processors

The CPU virtual processor runs all session threads (the threads that process requests from SQL client applications) and some internal threads. Internal threads perform services that are internal to the database server. For example, a thread that listens for connection requests from client applications is an internal thread.

Each CPU virtual processor can have a private memory cache associated with it. Each private memory cache block consists of 1 to 32 memory pages, where each memory page is 4096 bytes. The database server uses the private memory cache to improve access time to memory blocks. Use the VP_MEMORY_CACHE_KB configuration parameter to enable a private memory cache and specify information about the memory cache. For more information, see the *IBM Informix Administrator's Reference* and the *IBM Informix Performance Guide*.

Determine the number of CPU virtual processors needed

The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You must not allocate more CPU virtual processors than the number of hardware processors in the computer.

When the database server starts, the number of CPU virtual processors is automatically increased to half the number of CPU processors on the database server computer, unless the SINGLE_CPU_VP configuration parameter is enabled. However, you can adjust the number of CPU VPs based on your system.

To evaluate the performance of the CPU virtual processors while the database server is running, repeat the following command at regular intervals over a set period of time:

```
onstat -g glo
```

If the accumulated *usercpu* and *syscpu* times, taken together, approach 100 percent of the actual elapsed time for the period of the test, add another CPU virtual processor if you have a CPU available to run it.

You can use the VPCLASS configuration parameter to specify all of the following information:
* The number of virtual processors to start initially for a class
* The maximum number of virtual processors to run for the class
* Processor affinity for CPU class virtual processors
* Disabling of priority aging, if applicable

Use the VPCLASS configuration parameter to specify this information. (If you upgraded to the current version of Informix from a much older version, note that you must use the VPCLASS configuration parameter instead of the discontinued AFF_SPROC, AFFNPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS configuration parameters. Additionally, you cannot use the VPCLASS parameter cannot in combination with the discontinued parameters. If the `onconfig` file contains a NUMCPUVPS parameter, for example, you receive an error message if the file also contains a VPCLASS **cpu** parameter.)

In addition to considering the number of CPUs in the computer and the number of users who connect to the database server, also consider the fact that user-defined routines and DataBlade modules, which are collections of user-defined routines, run on either CPU virtual processors or user-defined virtual processors.

Related reference:

➥ VPCLASS configuration parameter (Administrator's Reference)

"Run poll threads on CPU or network virtual processors" on page 4-21

"Assign a UDR to a user-defined virtual-processor class" on page 4-15

➥ onstat -g glo command: Print global multithreading information (Administrator's Reference)

Run on a multiprocessor computer

If you are running multiple CPU virtual processors on a multiprocessor computer, set the MULTIPROCESSOR parameter in the `onconfig` file to 1. When you set MULTIPROCESSOR to 1, the database server performs locking in a manner that is appropriate for a multiprocessor computer. For information about setting multiprocessor mode, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

Run on a single-processor computer

If you are running the database server on a single-processor computer, set the MULTIPROCESSOR configuration parameter to 0. To run the database server with only one CPU virtual processor, set the SINGLE_CPU_VP parameter to 1.

Setting MULTIPROCESSOR to 0 enables the database server to bypass the locking that is required for multiple processes on a multiprocessor computer. For information about the MULTIPROCESSOR configuration parameter, see the *IBM Informix Administrator's Reference*.

Setting SINGLE_CPU_VP to 1 allows the database server to bypass some of the mutex calls that it normally makes when it runs multiple CPU virtual processors. For information about setting the SINGLE_CPU_VP parameter, see the *IBM Informix Administrator's Reference*.

Important: Setting *VPCLASS num* to 1 and SINGLE_CPU_VP to 0 does not reduce the number of mutex calls, even though the database server starts only one CPU virtual processor. You must set SINGLE_CPU_VP to 1 to reduce the amount of latching that is performed when you run a single CPU virtual processor.

Setting the SINGLE_CPU_VP parameter to 1 imposes two important restrictions on the database server, as follows:
- Only one CPU virtual processor is allowed.

  You cannot add CPU virtual processors while the database server is in online mode.

- No user-defined classes are allowed. (However, users can still define routines that run directly on the CPU VP.)

For more information, see "Add virtual processors in online mode" on page 5-3.

Add and drop CPU virtual processors in online mode
You can add or drop CPU class virtual processors while the database server is online. For instructions on how to do this, see "Add virtual processors in online mode" on page 5-3 and "Drop CPU and user-defined virtual processors" on page 5-4.

Prevent priority aging
Some operating systems lower the priority of long-running processes as they accumulate processing time. This feature of the operating system is called *priority aging*. Priority aging can cause the performance of database server processes to decline over time. In some cases, however, you can use the operating system to disable this feature and keep long-running processes running at a high priority.

To determine if priority aging is available on your computer, check the machine notes file that comes with your installation and is described in the Introduction to this guide.

If you can disable priority aging through the operating system, you can disable it by specifying `noage` for the priority entry in the VPCLASS configuration parameter. For more information, see the *IBM Informix Administrator's Reference*.

Processor affinity
The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*. Your database server distribution includes a machine notes file that contains information about whether your database server version supports this feature. When you assign a CPU virtual processor to a specific CPU, the virtual processor runs only on that CPU, but other processes also can run on that CPU.

Use the VPCLASS configuration parameter with the *aff* option to implement processor affinity on multiprocessor computers that support it.

The following figure illustrates the concept of processor affinity.

CPU virtual processors

Virtual processor

Virtual processor

Virtual processor

Starting CPU = 1

CPU 0   CPU 1   CPU 2   CPU 3

Number of virtual processors = 3
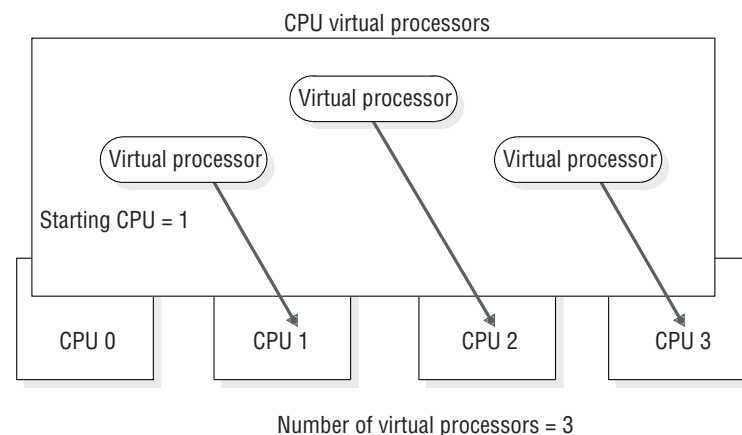
Figure 4-7. Processor affinity

UNIX only: To see if processor affinity is supported on your UNIX platform, see the machine notes file.

Set processor affinity with the VPCLASS configuration parameter:
To set processor affinity with the VPCLASS configuration parameter, you can specify individual processors or ranges of processors that you want to assign the virtual processors. When specifying a range of processors, you can also specify an incremental value with the range that indicates which CPUs in the range are assigned to the virtual processors. For example, you can specify that the virtual processors are assigned to every other CPU in the range 0-6, starting with CPU 0.

```
VPCLASS CPU,num=4,aff=(0-6/2)
```

The virtual processors are assigned to CPUs 0, 2, 4, 6.

If you specify `VPCLASS CPU,num=4,aff=(1-10/3)`, the virtual processors are assigned to every third CPU in the range 1-10, starting with CPU 1. The virtual processors are assigned to CPUs 1, 4, 7, 10.

When you specify more than one value or range, the values and ranges are not required to be incremental or in any particular order. For example you can specify `aff=(8,12,7-9,0-6/2)`.

The database server assigns CPU virtual processors to CPUs in a circular pattern, starting with the first processor number that you specify in the `aff` option. If you specify a larger number of CPU virtual processors than physical CPUs, the database server continues to assign CPU virtual processors starting with the first CPU. For example, suppose you specify the following VPCLASS settings:

```
VPCLASS cpu,num=8,aff=(4-7)
```

The database server makes the following assignments:
- CPU virtual processor number 0 to CPU 4
- CPU virtual processor number 1 to CPU 5
- CPU virtual processor number 2 to CPU 6
- CPU virtual processor number 3 to CPU 7
- CPU virtual processor number 4 to CPU 4
- CPU virtual processor number 5 to CPU 5
- CPU virtual processor number 6 to CPU 6
- CPU virtual processor number 7 to CPU 7

For more information, see the VPCLASS configuration parameter in the *IBM Informix Administrator's Reference*.

User-defined classes of virtual processors

You can define special classes of virtual processors to run user-defined routines or to run a DataBlade module . User-defined routines are typically written to support user-defined data types. If you do not want a user-defined routine to run in the CPU class, which is the default, you can assign it to a user-defined class of virtual processors (VPs). User-defined classes of virtual processors are also called *extension virtual processors*.

These topics provide the following information about user-defined virtual processors:
- When to run a C-language UDR in a user-defined VP instead of in the CPU VP

- How to assign a C-language UDR to a particular user-defined VP class
- How to add and drop user-defined VPs when the database server is in online mode

Related reference:

"Specify a virtual processor class" on page 5-2

Determine the number of user-defined virtual processors needed

You can specify as many user-defined virtual processors as your operating system allows. If you run many UDRs or parallel PDQ queries with UDRs, you must configure more user-defined virtual processors.

User-defined virtual processors

User-defined classes of virtual processors protect the database server from *ill-behaved* user-defined routines. An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

A well-behaved C-language UDR has none of these characteristics. Run only well-behaved C-language UDRs in a CPU VP.

Warning: Execution of an ill-behaved routine in a CPU VP can cause serious interference with the operation of the database server, possibly causing it to fail or behave erratically. In addition, the routine itself might not produce correct results.

To ensure safe execution, assign any ill-behaved user-defined routines to a user-defined class of virtual processors. User-defined VPs remove the following programming restrictions on the CPU VP class:

- The requirement to yield the processor regularly
- The requirement to eliminate blocking I/O calls

Functions that run in a user-defined virtual-processor class are not required to yield the processor, and they might issue direct file-system calls that block further processing by the virtual processor until the I/O is complete.

The normal processing of user queries is not affected by ill-behaved traits of a C-language UDR because these UDRs do not execute in CPU virtual processors. For a more detailed explanation of ill-behaved routines, see the *IBM Informix DataBlade API Programmer's Guide*.

Specify user-defined virtual processors

The VPCLASS parameter with the *vpclass* option defines a user-defined VP class. You also can specify a nonyielding user-defined virtual processor. For more information, see "Specify a virtual processor class" on page 5-2 and the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

Assign a UDR to a user-defined virtual-processor class

The SQL CREATE FUNCTION statement registers a user-defined routine. For example, the following CREATE FUNCTION statement registers the user-defined routine, **GreaterThanEqual()**, and specifies that calls to this routine are executed by the user-defined VP class named UDR:

```
CREATE FUNCTION GreaterThanEqual(ScottishName, ScottishName)
   RETURNS boolean
   WITH (CLASS = UDR  )
   EXTERNAL NAME '/usr/lib/objects/udrs.so'
   LANGUAGE C
```

To execute this function, the onconfig file must include a VPCLASS parameter that defines the UDR class. If not, calls to the **GreaterThanEqual** function fail.

Tip: The CLASS routine modifier can specify any name for the VP class. This class name is not required to exist when you register the UDR. However, when you try to run a UDR that specifies a user-defined VP class for its execution, this class must exist and have virtual processors assigned to it.

To configure the UDR class, include a line similar to the following one in the onconfig file. This line configures the UDR class with two virtual processors and with no priority aging.

```
VPCLASS       UDR  ,num=2,noage
```

The preceding line defines the UDR VP class as a yielding VP class; that is, this VP class allows the C-language UDR to yield to other threads that must access to the UDR VP class. For more information about how to use the VPCLASS configuration parameter, see the *IBM Informix Administrator's Reference*.

For more information about the CREATE FUNCTION statement, see the *IBM Informix Guide to SQL: Syntax*.

Related reference:

Add and drop user-defined virtual processors in online mode

You can add or drop virtual processors in a user-defined class while the database server is online. For instructions on how to do this, see

Java virtual processors

Java UDRs and Java applications run on specialized virtual processors, called *Java virtual processors* (JVPs). A JVP embeds a Java virtual machine (JVM) in its code. A JVP has the same capabilities as a CPU VP in that it can process complete SQL queries.

You can specify as many JVPs as your operating system allows. If you run many Java UDRs or parallel PDQ queries with Java UDRs, you must configure more JVPs. For more information about UDRs written in Java, see *J/Foundation Developer's Guide*.

Use the VPCLASS configuration parameter with the jvp keyword to configure JVPs. For more information, see the configuration parameters chapter in the *IBM Informix Administrator's Reference*.

Disk I/O virtual processors

The following classes of virtual processors perform disk I/O:
* PIO (physical-log I/O)
* LIO (logical-log I/O)

- AIO (asynchronous I/O)
- CPU (kernel-asynchronous I/O)

The PIO class performs all I/O to the physical-log file, and the LIO class performs all I/O to the logical-log files, *unless* those files are in raw disk space and the database server has implemented KAIO.

On operating systems that do not support KAIO, the database server uses the AIO class of virtual processors to perform database I/O that is not related to physical or logical logging.

The database server uses the CPU class to perform KAIO when it is available on a platform. If the database server implements KAIO, a KAIO thread performs all I/O to raw disk space, including I/O to the physical and logical logs.

UNIX only: To find out if your UNIX platform supports KAIO, see the machine notes file.

Windows only: Windows supports KAIO.

For more information about nonlogging I/O, see "Asynchronous I/O" on page 4-18.

Related reference:

"Specify a virtual processor class" on page 5-2

I/O priorities

In general, the database server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the nonlogging I/O queues. Prioritizing ensures that a high-priority log I/O, for example, is never queued behind a write to a temporary file, which has a low priority. The database server prioritizes the different types of disk I/O that it performs, as the table shows.

Table 4-2. How database server prioritizes disk I/O

Priority	Type of I/O	VP class
1st	Logical-log I/O	CPU or LIO
2nd	Physical-log I/O	CPU or PIO
3rd	Database I/O	CPU or AIO
3rd	Page-cleaning I/O	CPU or AIO
3rd	Read-ahead I/O	CPU or AIO

Logical-log I/O

The LIO class of virtual processors performs I/O to the logical-log files in the following cases:
- KAIO is not implemented.
- The logical-log files are in cooked disk space.

Only when KAIO is implemented and the logical-log files are in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the logical log.

The logical-log files store the data that enables the database server to roll back transactions and recover from system failures. I/O to the logical-log files is the highest priority disk I/O that the database server performs.

If the logical-log files are in a dbspace that is **not** mirrored, the database server runs only one LIO virtual processor. If the logical-log files are in a dbspace that is mirrored, the database server runs two LIO virtual processors. This class of virtual processors has no parameters associated with it.

Physical-log I/O

The PIO class of virtual processors performs I/O to the physical-log file in the following cases:

- KAIO is not implemented.
- The physical-log file is stored in buffered-file chunks.

Only when KAIO is implemented and the physical-log file is in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the physical log. The physical-log file stores *before-image*s of dbspace pages that have changed since the last *checkpoint.* (For more information about checkpoints, see "Checkpoints" on page 15-4.) At the start of recovery, before processing transactions from the logical log, the database server uses the physical-log file to restore before-images to dbspace pages that have changed since the last checkpoint. I/O to the physical-log file is the second-highest priority I/O after I/O to the logical-log files.

If the physical-log file is in a dbspace that is **not** mirrored, the database server runs only one PIO virtual processor. If the physical-log file is in a dbspace that is mirrored, the database server runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

The database server performs database I/O asynchronously, meaning that I/O is queued and performed independently of the thread that requests the I/O. Performing I/O asynchronously allows the thread that makes the request to continue working while the I/O is being performed.

The database server performs all database I/O asynchronously, using one of the following facilities:

- AIO virtual processors
- KAIO on platforms that support it

Database I/O includes I/O for SQL statements, read-ahead, page cleaning, and checkpoints.

Kernel-asynchronous I/O: The database server uses KAIO when the following conditions exist:

- The computer and operating system support it.
- A performance gain is realized.
- The I/O is to raw disk space.

The database server implements KAIO by running a KAIO thread on the CPU virtual processor. The KAIO thread performs I/O by making system calls to the operating system, which performs the I/O independently of the virtual processor.

The KAIO thread can produce better performance for disk I/O than the AIO virtual processor can, because it does not require a switch between the CPU and AIO virtual processors.

UNIX only: IBM Informix implements KAIO when Informix ports to a platform that supports this feature. The database server administrator does not configure KAIO. To see if KAIO is supported on your platform, see the machine notes file.

Linux only: Kernel asynchronous I/O (KAIO) is enabled by default. You can disable this by specifying that KAIOOFF=1 in the environment of the process that starts the server.

On Linux, there is a system-wide limit of the maximum number of parallel KAIO requests. The /proc/sys/fs/aio-max-nr file contains this value. The Linux system administrator can increase the value, for example, by using this command:

```
# echo new_value > /proc/sys/fs/aio-max-nr
```

The current number of allocated requests of all operating system processes is visible in the /proc/sys/fs/aio-nr file.

By default, Dynamic Version allocates half of the maximum number of requests and assigns them equally to the number of configured CPU virtual processors. You can use the environment variable KAIOON to control the number of requests allocated per CPU virtual processor. Do this by setting KAIOON to the required value before starting Informix.

The minimum value for KAIOON is 100. If Linux is about to run out of KAIO resources, for example when dynamically adding many CPU virtual processors, warnings are printed in the online.log file. If this happens, the Linux system administrator must add KAIO resources as described previously.

AIO virtual processors: If the platform does not support KAIO or if the I/O is to buffered-file chunks, the database server performs database I/O through the AIO class of virtual processors. All AIO virtual processors service all I/O requests equally within their class.

The database server assigns each disk chunk a queue, sometimes known as a *gfd queue*, based on the file name of the chunk. The database server orders I/O requests within a queue according to an algorithm that minimizes disk-head movement. The AIO virtual processors service queues that have work pending in round-robin fashion.

All other non-chunk I/O is queued in the AIO queue.

Use the VPCLASS parameter with the *aio* keyword to specify the number of AIO virtual processors that the database server starts initially. For information about VPCLASS, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

You can start additional AIO virtual processors while the database server is in online mode. For more information, see "Add virtual processors in online mode" on page 5-3.

You cannot drop AIO virtual processors while the database server is in online mode.

Automatic increasing and decreasing of AIO virtual processors:
The AUTO_AIOVPS configuration parameter enables or disables the database server to automatically increase the number of AIO VPS and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload. Disable this parameter if you want to manually adjust the number. For details on setting this parameter, see the *IBM Informix Administrator's Reference*.

Number of AIO virtual processors needed:
The goal in allocating AIO virtual processors is to allocate enough of them so that the lengths of the I/O request queues are kept short; that is, the queues have as few I/O requests in them as possible. When the *gfd* queues are consistently short, it indicates that I/O to the disk devices is being processed as fast as the requests occur.

The **onstat-g ioq** command displays the length and other statistics about I/O queues. You can use this command to monitor the length of the *gfd* queues for the AIO virtual processors. For more information, see "Monitor virtual processors" on page 5-4 and information about monitoring virtual processors in the *IBM Informix Performance Guide*.

One AIO virtual processor might be sufficient:
- If the database server implements kernel asynchronous I/O (KAIO) on your platform and all of your dbspaces are composed of raw disk space
- If your file system supports direct I/O for the page size used for the dbspace chunk and you use direct I/O

Allocate two AIO virtual processors per active dbspace that is composed of buffered file chunks.
- If the database server implements KAIO, but you are using some buffered files for chunks
- IF KAIO is not supports by the system for chunks.

If KAIO is *not* implemented on your platform, allocate two AIO virtual processors for each disk that the database server accesses frequently.

If you use cooked files and if you enable direct I/O using the DIRECT_IO configuration parameter, you might be able to reduce the number of AIO virtual processors.

If the database server implements KAIO and you enabled direct I/O using the DIRECT_IO configuration parameter, IBM Informix attempts to use KAIO, so you probably do not require more than one AIO virtual processor. However, even when direct I/O is enabled, if the file system does not support either direct I/O or KAIO, you still must allocate two AIO virtual processors for every active dbspace that is composed of buffered file chunks or does not use KAIO.

Temporary dbspaces do not use direct I/O. If you have temporary dbspaces, you probably require more than one AIO virtual processors.

Allocate enough AIO virtual processors to accommodate the peak number of I/O requests. Generally, it is not detrimental to allocate too many AIO virtual processors.

You can use the AUTO_AIOVPS configuration parameter to enable the database server to automatically increase the number of AIO virtual processors and page-cleaner threads when the server detects that AIO virtual processors are not keeping up with the I/O workload.

Network virtual processors

As explained in Chapter 2, "Client/server communication," on page 2-1, a client can connect to the database server in the following ways:

- Through a network connection
- Through a pipe
- Through shared memory

The network connection can be made by a client on a remote computer or by a client on the local computer mimicking a connection from a remote computer (called a *local-loopback connection*).

Specifying Network Connections

In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the **sqlhosts** file or registry. Each dbservername parameter in **sqlhosts** has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more *poll threads* for each unique **nettype** entry.

The NETTYPE configuration parameter provides optional configuration information for an interface/protocol combination. You can use it to allocate more than one poll thread for an interface/protocol combination and also designate the virtual-processor class (CPU or NET) on which the poll threads run.

For a complete description of the NETTYPE configuration parameter, see the *IBM Informix Administrator's Reference*.

Related reference:

"sqlhosts Connectivity information" on page 2-21

Run poll threads on CPU or network virtual processors

Poll threads can run either on CPU virtual processors or on network virtual processors. In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This might not be true, however, on a multiprocessor computer with many remote clients.

The NETTYPE parameter has an optional entry, called *vp class*, which you can use to specify either CPU or NET, for CPU or network virtual-processor classes, respectively.

If you do not specify a virtual processor class for the interface/protocol combination (poll threads) associated with the **DBSERVERNAME** variable, the class defaults to CPU. The database server assumes that the interface/protocol combination associated with **DBSERVERNAME** is the primary interface/protocol combination and that it is the most efficient.

For other interface/protocol combinations, if no vp class is specified, the default is NET.

While the database server is in online mode, you cannot drop a CPU virtual processor that is running a poll or a listen thread.

Important: You must carefully distinguish between poll threads for network connections and poll threads for shared memory connections, which run one per CPU virtual processor. TCP connections must only be in network virtual processors, and you must only have the minimum required to maintain responsiveness. Shared memory connections must only be in CPU virtual processors and run in every CPU virtual processor.

Related reference:

"Determine the number of CPU virtual processors needed" on page 4-11

Specify the number of networking virtual processors

Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by the NET class. If you specify CPU for the vp class, you must allocate a sufficient number of CPU virtual processors to run the poll threads. If the database server does not have a CPU virtual processor to run a CPU poll thread, it starts a network virtual processor of the specified class to run it.

For most systems, one poll thread and consequently one virtual processor per network interface/protocol combination is sufficient. For systems with 200 or more network users, running additional network virtual processors might improve throughput. In this case, you must experiment to determine the optimal number of virtual processors for each interface/protocol combination.

Specify listen and poll threads for the client/server connection

When you start the database server, the **oninit** process starts an internal thread, called a *listen thread*, for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES parameters in the onconfig file. To specify a listen port for each of these dbservername entries, assign it a unique combination of **hostname** and **service name** entries in sqlhosts. For example, the sqlhosts file or registry entry shown in the following table causes the database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Table 4-3. A listen thread for each listen port

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1

The listen thread opens the port and requests one of the poll threads for the specified interface/protocol combination to monitor the port for client requests. The poll thread runs either in the CPU virtual processor or in the network virtual processor for the connection that is being used. For information about the number of poll threads, see "Specify the number of networking virtual processors."

For information about how to specify whether the poll threads for an interface/protocol combination run in CPU or network virtual processors, see "Run poll threads on CPU or network virtual processors" on page 4-21 and to the NETTYPE configuration parameter in the *IBM Informix Administrator's Reference*.

When a poll thread receives a connection request from a client, it passes the request to the listen thread for the port. The listen thread authenticates the user, establishes the connection to the database server, and starts an **sqlexec** thread, the session thread that performs the primary processing for the client. The following figure illustrates the roles of the listen and poll threads in establishing a connection

with a client application.
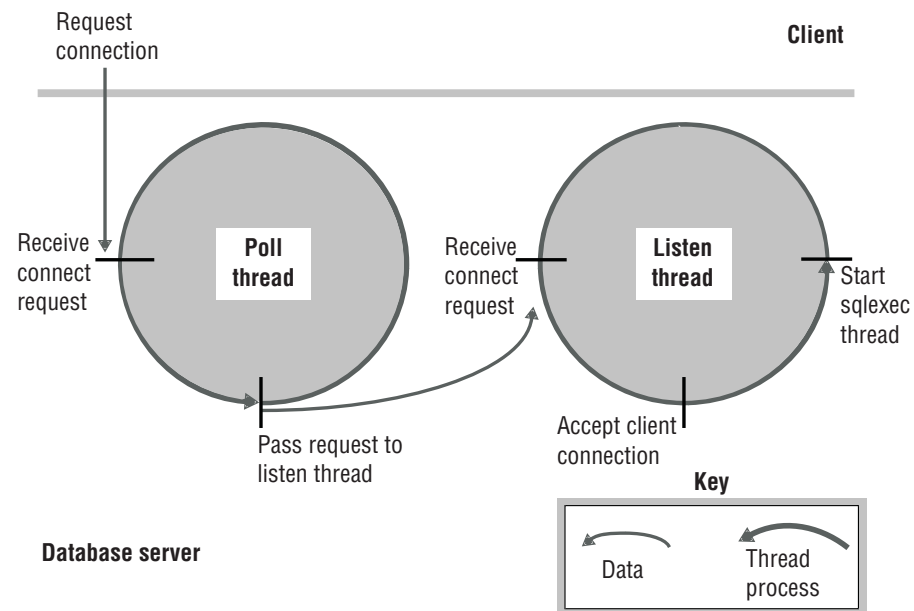
Figure 4-8. The roles of the poll and the listen threads in connecting to a client

A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlexec** thread. For network connections, the poll thread places the message in a queue in the shared-memory global pool. The poll thread then wakes up the **sqlexec** thread of the client to process the request. Whenever possible, the **sqlexec** thread writes directly back to the client without the help of the poll thread. In general, the poll thread reads data from the client, and the **sqlexec** thread sends data to the client.

UNIX only: For a shared-memory connection, the poll thread places the message in the communications portion of shared memory.

The following figure illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

Process

Client

Read data

Send
data
to client

Pass request
and data
to sqlexec

**Poll
thread**

**sqlexec
thread**

Process

Read data
from client

Wait for client
request

Key

Database server

Application
process

Thread
process

Data
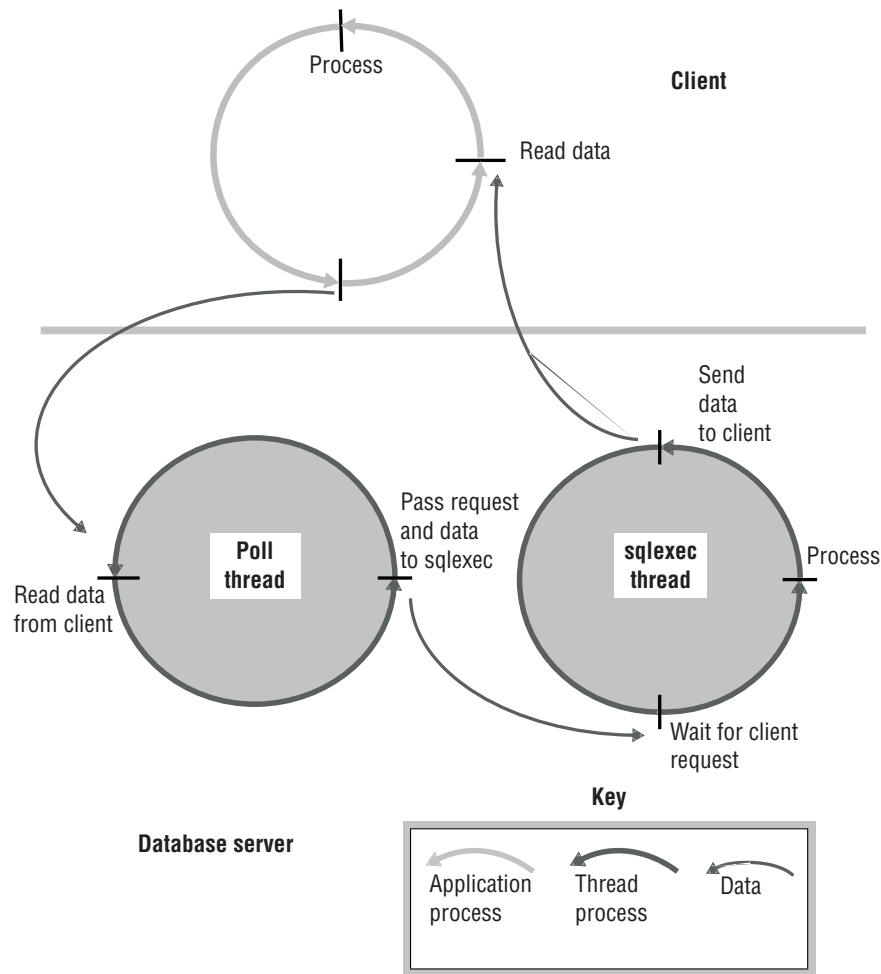
Figure 4-9. The roles of the poll and sqlexec threads in communicating with the client application

Fast polling

You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling. Fast polling is beneficial if you have many connections. For example, if you have more than 300 concurrent connections with the database server, you can enable the FASTPOLL configuration parameter for better performance. You can enable fast polling by setting the FASTPOLL configuration parameter to 1.

If your operating system does not support fast polling, IBM Informix ignores the FASTPOLL configuration parameter.

Multiple listen threads

You can improve service for connection requests by using multiple listen threads.

If the database server cannot service connection requests satisfactorily for a given interface/protocol combination with a single port and corresponding listen thread, you can improve service for connection requests in the following ways:

• By adding listen threads for additional ports.

- By adding listen threads to the same port if you have the **onimcsoc** or **onsoctcp** protocol
- By adding another network-interface card.
- By dynamically starting, stopping, or restarting listen threads for a SOCTCP or TLITCP network protocol, using SQL administration API or **onmode -P** commands.

If you have multiple listen threads for one port for the **onsoctcp** protocol, the database server can accept new connections if a CPU VP connection is busy.

Add listen threads:

When you start the database server, the **oninit** process starts a listen thread for servers with the server names and server alias names that you specify with the DBSERVERNAME and DBSERVERALIASES configuration parameters. You can add listen threads for additional ports.

You can also set up multiple listen threads for one service (port) for the **onimcsoc** or **onsoctcp** protocol.

To add listen threads for additional ports, you must first use the DBSERVERALIASES parameter to specify dbservernames for each of the ports. For example, the DBSERVERALIASES parameter in the following figure defines two additional dbservernames, **soc_ol2** and **soc_ol3**, for the database server instance identified as **soc_ol1**.

```
DBSERVERNAME      soc_ol1
DBSERVERALIASES   soc_ol2,soc_ol3
```

After you define additional dbservernames for the database server, you must specify an interface/protocol combination and port for each of them in the `sqlhosts` file or registry. Each port is identified by a unique combination of **hostname** and **servicename** entries. For example, the `sqlhosts` entries shown in the following table cause the database server to start three listen threads for the **onsoctcp** interface/protocol combination, one for each of the ports defined.

Table 4-4. The sqlhosts entries to listen to multiple ports for a single interface/protocol combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1
soc_ol2	onsoctcp	myhost	port2
soc_ol3	onsoctcp	myhost	port3

If you include a NETTYPE parameter for an interface/protocol combination, it applies to all the connections for that interface/protocol combination. In other words, if a NETTYPE parameter exists for **onsoctcp** in the previous table, it applies to all of the connections shown. In this example, the database server runs one *poll* thread for the **onsoctcp** interface/protocol combination unless the NETTYPE parameter specifies more. For more information about entries in the `sqlhosts` file or registry, see "Connectivity files" on page 2-9.

Setting up multiple listen threads for one port for the onimcsoc or onsoctcp protocol

To set up multiple listen threads for one service (port) for the **onimcsoc** or **onsoctcp** protocol, specify DBSERVERNAME and DBSERVERALIASES information as follows:
- DBSERVERNAME *<name>-<n>*
- DBSERVERALIASES *<name1>-<n>,<name2>*

For example:
- To bring up two listen threads for the server with the DBSERVERNAME of ifx, specify:

  DBSERVERNAME ifx-2
- To bring up two listen threads for DBSERVERALIASES ifx_a and ifx_b, specify:

  DBSERVERALIASES ifx_a-2,ifx_b-2

Add a network-interface card:

You can add a network-interface card to improve performance or connect the database server to multiple networks.

You might want to improve performance if the network-interface card for the host computer cannot service connection requests satisfactorily.

To support multiple network-interface cards, you must assign each card a unique **hostname** (network address) in sqlhosts.

For example, using the same dbservernames shown in "Add listen threads" on page 4-25, the sqlhosts file or registry entries shown in the following table cause the database server to start three listen threads for the same interface/protocol combination (as did the entries in "Add listen threads" on page 4-25). In this case, however, two of the threads are listening to ports on one interface card (**myhost1**), and the third thread is listening to a port on the second interface card (**myhost2**).

Table 4-5. Example of sqlhosts entries to support two network-interface cards for the onsoctcp interface/protocol combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost1	port1
soc_ol2	onsoctcp	myhost1	port2
soc_ol3	onsoctcp	myhost2	port1

Dynamically starting, stopping, or restarting a listen thread:

You can dynamically start, stop, or stop and start a listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections. For example, you might want to stop listen threads that are unresponsive and then start new ones in situations when other server functions are performing normally and you do not want to shut down the server.

The listen thread must be defined in the sqlhosts file for the server. If necessary, before start, stop, or restart a listen thread, you can revise the sqlhosts entry.

To dynamically start, stop, or restart listen threads:

1. Run one of the following **onmode -P** commands:
   * **onmode -P start** *server_name*
   * **onmode -P stop** *server_name*
   * **onmode -P restart** *server_name*
2. Alternatively, if you are connected to the **sysadmin** database, either directly or remotely, you can run one of the following commands:
   * An **admin()** or **task()** command with the `start listen` argument, using the format
     ```
     EXECUTE FUNCTION task("start listen", "server_name");
     ```
   * An **admin()** or **task()** command with the `stop listen` argument, using the format
     ```
     EXECUTE FUNCTION task("stop listen" ,"server_name");
     ```
   * An **admin()** or **task()** command with the `restart listen` argument, using the format
     ```
     EXECUTE FUNCTION task("restart listen", "server_name");
     ```

For example, either of the following commands starts a new listen thread for a server named **ifx_serv2**:
```
onmode -P start ifx_serv2
EXECUTE FUNCTION task("start listen", "ifx_serv2");
```

Communications support module virtual processor

The communications support module (CSM) class of virtual processors performs communications support service and communications support module functions.

The database server starts the same number of CSM virtual processors as the number of CPU virtual processors that it starts, unless the communications support module is set to GSSCSM to support single sign-on. When the communications support module is GSSCSM, the database server starts only one CSM virtual processor.

For more information about the communications support service, see Chapter 2, "Client/server communication," on page 2-1.

Encrypt virtual processors

If the `encrypt` option of the VPCLASS parameter is not defined in the `onconfig` configuration file, the database server starts one ENCRYPT VP the first time that any encryption or decryption functions defined for column-level encryption are called. You can define multiple ENCRYPT VPs if necessary to decrease the time required to start the database server.

Use the VPCLASS configuration parameter with the `encrypt` keyword to configure encryption VPs. For example, to add five ENCRYPT VPs, add information in the `onconfig` file as follows:
```
VPCLASS encrypt,num=5
```

You can modify the same information using the **onmode** utility, as follows:
```
onmode -p 5 encrypt
```

For more information, see the configuration parameters and the **onmode** utility topics in the *IBM Informix Administrator's Reference*. For more information about column-level encryption, see the *IBM Informix Security Guide*.

Optical virtual processor

The optical class (OPT) of virtual processors is used only with the Optical Subsystem. The Optical Subsystem starts one virtual processor in the optical class if the STAGEBLOB configuration parameter is present. For more information about the Optical Subsystem, see the *IBM Informix Optical Subsystem Guide*.

Audit virtual processor

The database server starts one virtual processor in the audit class (ADT) when you turn on audit mode by setting the ADTMODE parameter in the `onconfig` file to 1. For more information about database server auditing, see the *IBM Informix Security Guide*.

Miscellaneous virtual processor

The miscellaneous virtual processor services requests for system calls that might require a very large stack, such as fetching information about the current user or the host-system name. Only one thread runs on this virtual processor; it executes with a stack of 128 KB.

Basic text search virtual processors

A basic text search virtual processor is required to run basic text search queries.

A basic text search virtual processor is added automatically when you create a basic text search index.

A basic text search virtual processor runs without yielding; it processes one index operation at a time. To run multiple basic text search index operations and queries simultaneously, create additional basic text search virtual processors.

Use the VPCLASS configuration parameter with the BTS keyword to configure basic text search virtual processors. For example, to add five BTS virtual processors, add the following line to the `onconfig` and restart the database server:

```
VPCLASS bts,num=5
```

You can dynamically add BTS virtual processors by using the **onmode -p** command, for example:

```
onmode -p 5 bts
```

Related reference:

▸ VPCLASS configuration parameter (Administrator's Reference)

▸ onmode -p: Add or remove virtual processors (Administrator's Reference)

▸ Basic Text Search (Database Extensions User's Guide)

MQ messaging virtual processor

An MQ virtual processor is required to use MQ messaging.

When you perform MQ messaging transactions, an MQ virtual processor is created automatically.

An MQ virtual processor runs without yielding; it processes one operation at a time. To perform multiple MQ messaging transactions simultaneously, create additional MQ virtual processors.

Use the VPCLASS configuration parameter with the MQ keyword to configure MQ virtual processors. For example, to add five MQ virtual processors, add the following line to the `onconfig` and restart the database server:

```
VPCLASS mq,noyield,num=5
```

For more information about the VPCLASS configuration parameter, see the *IBM Informix Administrator's Reference*. For more information about MQ messaging, see the *IBM Informix Database Extensions User's Guide*.

Web feature service virtual processor

A web feature service virtual processor is required to use web feature service for geospatial data.

When you run a WFS routine, a WFS virtual processor is created automatically.

A WFS virtual processor runs without yielding; it processes one operation at a time. To run multiple WFS routines simultaneously, create additional WFS virtual processors.

Use the VPCLASS configuration parameter with the WFSVP keyword to configure WFS virtual processors. For example, to add five WFS virtual processors, add the following line to the `onconfig` and restart the database server:

```
VPCLASS wfsvp,noyield,num=5
```

For more information about the VPCLASS configuration parameter, see the *IBM Informix Administrator's Reference*. For more information about WFS, see the *IBM Informix Database Extensions User's Guide*.

XML virtual processor

An XML virtual processor is required to perform XML publishing.

When you run an XML function, an XML virtual processor is created automatically.

An XML virtual processor runs one XML function at a time. To run multiple XML functions simultaneously, create additional XML virtual processors.

Use the VPCLASS configuration parameter with the IDSXMLVP keyword to configure XML virtual processors. For example, to add five XML virtual processors, add the following line to the `onconfig` and restart the database server:

```
VPCLASS idsxmlvp,num=5
```

You can dynamically add XML virtual processors by using the **onmode -p** command, for example:

```
onmode -p 5 idsxmlvp
```

For more information about the VPCLASS configuration parameter and the **onmode** utility, see the *IBM Informix Administrator's Reference*. For more information about XML publishing, see the *IBM Informix Database Extensions User's Guide*.

Chapter 5. Manage virtual processors

These topics describe how to set the configuration parameters that affect database server virtual processors, and how to start and stop virtual processors.

For descriptions of the virtual-processor classes and for advice on how many virtual processors you must specify for each class, see Chapter 4, "Virtual processors and threads," on page 4-1.

Set virtual-processor configuration parameters

As user **root** or **informix**, use a text editor to set the configuration parameters for the database server virtual processors.

To implement any changes that you make to configuration parameters, you must shut down and restart the database server. For more information, see "Set up shared memory" on page 7-6.

Set virtual processor parameters with a text editor

You can use a text editor program to set ONCONFIG parameters at any time. Use the editor to locate the parameter that you want to change, enter the new value, and rewrite the file to disk.

The table lists the ONCONFIG parameters that are used to configure virtual processors. For more information about how these parameters affect virtual processors, see "Virtual-processor classes" on page 4-11.

Table 5-1. Configuration parameters for configuring virtual processors

Parameter	Subparameters	Purpose
MULTIPROCESSOR		Specifies that you are running on a multiprocessor computer
NETTYPE		Specifies parameters for network protocol threads (and virtual processors)
SINGLE_CPU_VP		Specifies that you are running a single CPU virtual processor
VPCLASS	adm kio shm adt lio soc aio msc str cpu ntk tli encrypt opt jvp pio	Specifies a predefined class name for the virtual processors. For example, jvp specifies a Java virtual processor (JVP).
VPCLASS	num=*number*	Specifies the number of virtual processors of the specified class that the database server starts
VPCLASS	max=*number*	Specifies the maximum number of virtual processors allowed for a class
VPCLASS	noage	Specifies the disabling of the aging priority
VPCLASS	aff = (*processor_number, start_range - end_range, start_range - end_range / increment*)	Specifies the assignment of virtual processors to CPUs, if processor affinity is available
VPCLASS	*user_defined*	Specifies user-defined virtual processor

Table 5-1. Configuration parameters for configuring virtual processors (continued)

Parameter	Subparameters	Purpose
VPCLASS	noyield	Specifies non-yielding virtual processor
VP_MEMORY_ CACHE_KB		Speeds access to memory blocks.

Specify a virtual processor class

Use the **VPCLASS** configuration parameter to designate a class of virtual processors (VPs), create a user-defined virtual processor, and specify options such as the number of VPs that the server starts, the maximum number of VPs allowed for the class, and the assignment of VPs to CPUs if processor affinity is available.

You can specify a **VPCLASS** name of up to 128 bytes. The **VPCLASS** name must begin with a letter or underscore, and must contain letters, digits, underscores, or $ characters.

Note: On Windows systems, the number of crypto virtual processors is always set to 1, regardless of the value set by the **VPCLASS crypto** configuration parameter in the onconfig.std file.

Related concepts:

"Disk I/O virtual processors" on page 4-16

Related reference:

VPCLASS configuration parameter (Administrator's Reference)

"User-defined classes of virtual processors" on page 4-14

Disable priority aging (UNIX)

Use the VPCLASS parameter with the *noage* option to disable process priority aging on platforms that allow this feature.

For recommended values for these database server parameters on UNIX, see your machine notes file.

Start and stop virtual processors

When you start the database server, the **oninit** utility starts the number and types of virtual processors that you have specified directly and indirectly. You configure virtual processors primarily through ONCONFIG parameters and, for network virtual processors, through parameters in the sqlhosts file or registry. For descriptions of the virtual-processor classes, see "Virtual-processor classes" on page 4-11.

You can use the database server to start a maximum of 1000 virtual processors.

After the database server is in online mode, you can start additional virtual processors to improve performance, if necessary. For more information, see "Add virtual processors in online mode" on page 5-3.

While the database server is in online mode, you can drop virtual processors of the CPU and user-defined classes. For more information, see "Drop CPU and user-defined virtual processors" on page 5-4.

To terminate the database server and thereby terminate all virtual processors, use the **onmode -k** command. For more information about using **onmode -k**, see the *IBM Informix Administrator's Reference*.

Add virtual processors in online mode

While the database server is in online mode, you can start additional virtual processors for the following classes: CPU, AIO, PIO, LIO, SHM, STR, TLI, SOC, JVP, and user-defined. The database server automatically starts one virtual processor each in the LIO and PIO classes unless mirroring is used, in which case it starts two.

You can start these additional virtual processors with the -p option of the **onmode** utility.

You can start these additional virtual processors with the -p option of the **onmode** utility or ISA.

You can also start additional virtual processors for user-defined classes to run user-defined routines. For more information about user-defined virtual processors, see "Assign a UDR to a user-defined virtual-processor class" on page 4-15.

Add virtual processors in online mode with onmode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in online mode. Specify the number of virtual processors that you want to add with a positive number. As an option, you can precede the number of virtual processors with a plus sign (+). Following the number, specify the virtual processor class in lowercase letters. For example, either of the following commands starts four additional virtual processors in the AIO class:

```
onmode -p 4 aio
```

```
onmode -p +4 aio
```

The **onmode** utility starts the additional virtual processors immediately.

You can add virtual processors to only one class at a time. To add virtual processors for another class, you must run **onmode** again.

Add network virtual processors

When you add network virtual processors, you add poll threads, each of which requires its own virtual processor to run.

If you try to add poll threads for a protocol while the database server is in online mode, and you specify in the NETTYPE configuration parameter that the poll threads run in the CPU class, the database server does not start the new poll threads if CPU virtual processors are not available to run them.

In the following example, the poll threads handle a total of 240 connections:

```
NETTYPE ipcshm,4,60,CPU # Configure poll thread(s) for nettype
```

For ipcshm, the number of poll threads correspond to the number of memory segments. For example, if NETTYPE is set to 3,100 and you want one poll thread, set the poll thread to 1,300.

Drop CPU and user-defined virtual processors

While the database server is in online mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU and user-defined classes.

Drop CPU virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the CPU class in lowercase letters. For example, the following command drops two CPU virtual processors:

```
% onmode -p -2 cpu
```

If you attempt to drop a CPU virtual processor that is running a poll thread, you receive the following message:

```
onmode: failed when trying to change the number of cpu virtual processor by -number.
```

For more information, see "Run poll threads on CPU or network virtual processors" on page 4-21.

Drop user-defined virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the user-defined class in lowercase letters. For example, the following command drops two virtual processors of the class *usr*:

```
onmode -p -2 usr
```

Windows only: In Windows, you can have only one user-defined virtual processor class at a time. Omit the *number* parameter in the **onmode -p** *vpclass* command.

For information about how to create a user-defined class of virtual processors and assign user-defined routines to it, see "User-defined classes of virtual processors" on page 4-14.

Monitor virtual processors

Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity. For more information about these **onstat -g** options, see the topics on the effect of configuration on CPU utilization in the *IBM Informix Performance Guide*.

For examples of output for the **onstat -g** commands, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitor virtual processors with command-line utilities

You can use the following **onstat -g** options to monitor virtual processors:

The onstat -g ath command
The **onstat -g ath** command displays information about system threads and the virtual-processor classes.

The onstat -g glo command

Use the **onstat -g glo** command to display information about each virtual processor that is currently running, and cumulative statistics for each virtual processor class. For an example of **onstat -g glo** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The onstat -g ioq command

Use the **onstat -g ioq** option to determine whether you must allocate additional virtual processors. The command **onstat -g ioq** displays the length and other statistics about I/O queues.

If the length of the I/O queue is growing, I/O requests are accumulating faster than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.

For an example of **onstat -g ioq** output, see information in the *IBM Informix Administrator's Reference*.

The onstat -g rea command

Use the **onstat -g rea** option to monitor the number of threads in the ready queue. If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might be required to add more virtual processors to your configuration.

For an example of **onstat -g rea** output, see information in the *IBM Informix Administrator's Reference*.

Monitor virtual processors with SMI tables

Query the **sysvpprof** table to obtain information about the virtual processors that are currently running. This table contains the following columns.

Column	Description
vpid	Virtual-processor ID number
class	Virtual-processor class
usercpu	Minutes of user CPU used
syscpu	Minutes of system CPU used

Chapter 6. Shared memory

These topics describe the content of database server shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory. For information about how to change the database server configuration parameters that determine shared memory allocations, see Chapter 7, "Manage shared memory," on page 7-1.

Shared memory

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory. The database server uses shared memory for the following purposes:

- To reduce memory usage and disk I/O
- To perform high-speed communication between processes

Shared memory enables the database server to reduce overall memory usage because the participating processes, in this case, virtual processors, do not require maintaining private copies of the data that is in shared memory.

Shared memory reduces disk I/O, because buffers, which are managed as a common pool, are flushed on a database server-wide basis instead of a per-process basis. Furthermore, a virtual processor can often avoid reading data from disk because the data is already in shared memory as a result of an earlier read operation. The reduction in disk I/O reduces execution time.

Shared memory provides the fastest method of interprocess communication, because it processes read and write messages at the speed of memory transfers.

Shared-memory use

The database server uses shared memory for the following purposes:

- To enable virtual processors and utilities to share data
- To provide a fast communications channel for local client applications that use IPC communication

The following figure illustrates the shared-memory scheme.
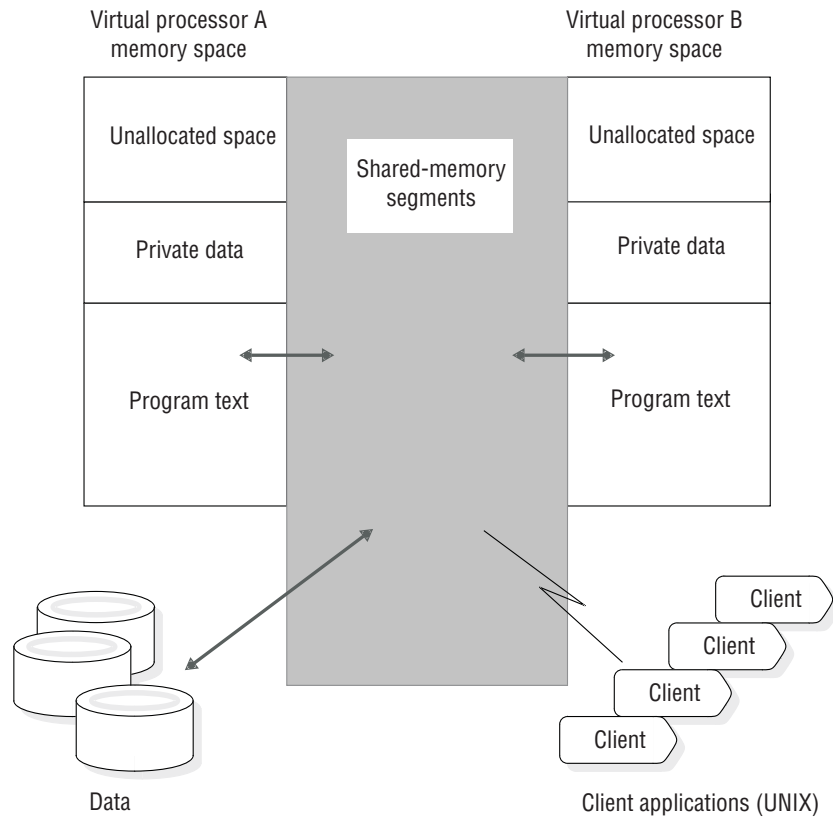
Figure 6-1. How the database server uses shared memory

Shared-memory allocation

The database server creates the following portions of shared memory:

- The *resident* portion
- The *virtual* portion
- The IPC *communications* or message portion

  If the sqlhosts file specifies shared-memory communications, the database server allocates memory for the communications portion.
- The *virtual-extension* portion

The database server adds operating-system segments, as required, to the virtual and virtual-extension portions of shared memory.

For more information about shared-memory settings for your platform, see the machine notes. The following figure shows the contents of each portion of shared memory.

All database server virtual processors have access to the same shared-memory segments. Each virtual processor manages its work by maintaining its own set of pointers to shared-memory resources such as buffers, locks, and latches. Virtual processors attach to shared memory when you take the database server from offline mode to quiescent, administration, or online. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple

threads.
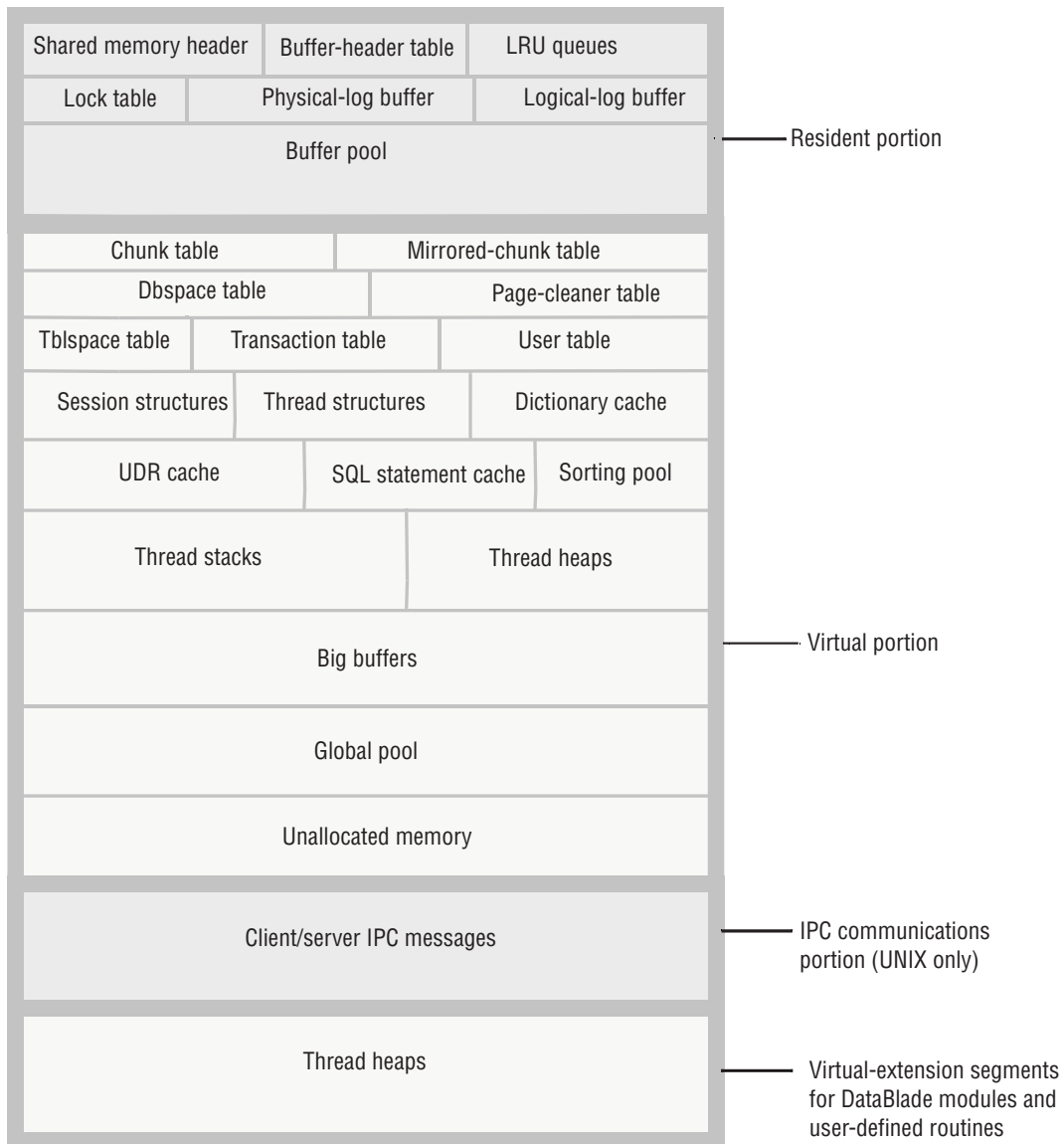
Figure 6-2. Contents of database server shared memory

Shared-memory size

Each portion of the database server shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks that are 4 KB in size and managed by a bitmap.

The header-line output by the **onstat** utility contains the size of the database server shared memory, expressed in KB. You can also use **onstat -g seg** to monitor how much memory the database server allocates for each portion of shared memory. For information about how to use **onstat**, see the *IBM Informix Administrator's Reference*.

You can set the SHMTOTAL parameter in the `onconfig` file to limit the amount of memory overhead that the database server can place on your computer or node. The SHMTOTAL parameter specifies the total amount of shared memory that the

database server can use for all memory allocations. However, certain operations might fail if the database server requires more memory than the amount set in SHMTOTAL. If this condition occurs, the database server displays the following message in the message log:

```
size of resident + virtual segments x + y > z
   total allowed by configuration parameter SHMTOTAL
```

In addition, the database server returns an error message to the application that initiated the offending operation. For example, if the database server requires more memory than you specify in SHMTOTAL while it tries to perform an operation such as an index build or a hash join, it returns an error message to the application that is similar to one of the following messages:

```
-567    Cannot write sorted rows.
-116    ISAM error: cannot allocate memory.
```

After the database server sends these messages, it rolls back any partial results performed by the offending query.

Internal operations, such as page-cleaner or checkpoint activity, can also cause the database server to exceed the SHMTOTAL ceiling. When this situation occurs, the database server sends a message to the message log. For example, suppose that the database server attempts and fails to allocate additional memory for page-cleaner activity. As a consequence, the database server sends information to the message log that is similar to the following messages:

```
17:19:13        Assert Failed: WARNING! No memory available for page cleaners
17:19:13        Who: Thread(11, flush_sub(0), 9a8444, 1)
17:19:13        Results: Database server may be unable to complete a checkpoint
17:19:13        Action: Make more virtual memory available to database server
17:19:13        See Also: /tmp/af.c4
```

After the database server informs you about the failure to allocate additional memory, it rolls back the transactions that caused it to exceed the SHMTOTAL limit. Immediately after the rollback, operations no longer fail from lack of memory, and the database server continues to process transactions as usual.

Action to take if SHMTOTAL is exceeded

When the database server requires more memory than SHMTOTAL allows, a transient condition occurs, perhaps caused by a burst of activity that exceeds the normal processing load. Only the operation that caused the database server to run out of memory temporarily fails. Other operations continue to be processed in a normal fashion.

If messages indicate on a regular basis that the database server requires more memory than SHMTOTAL allows, you have not configured the database server correctly. Lowering DS_TOTAL_MEMORY or the **buffers** value in the BUFFERPOOL configuration parameter is one possible solution; increasing the value of SHMTOTAL is another.

Processes that attach to shared memory

The following processes attach to the database server shared memory:
- Client-application processes that communicate with the database server through the shared-memory communications portion (**ipcshm**)
- Database server virtual processors
- Database server utilities

The following topics describe how each type of process attaches to the database server shared memory.

How a client attaches to the communications portion (UNIX)

Client-application processes that communicate with the database server through shared memory (nettype `ipcshm`) attach transparently to the communications portion of shared memory. System-library functions that are automatically compiled into the application enable it to attach to the communications portion of shared memory. For information about specifying a shared-memory connection, see Chapter 2, "Client/server communication," on page 2-1, and "Network virtual processors" on page 4-21.

If the **INFORMIXSHMBASE** environment variable is not set, the client application attaches to the communications portion at an address that is platform-specific. If the client application attaches to other shared-memory segments (not database server shared memory), the user can set the **INFORMIXSHMBASE** environment variable to specify the address at which to attach the database server shared-memory communications segments. When you specify the address at which to address the shared-memory communications segments, you can prevent the database server from colliding with the other shared-memory segments that your application uses. For information about how to set the **INFORMIXSHMBASE** environment variable, see the *IBM Informix Guide to SQL: Reference*.

Related reference:

"Shared-memory connections (UNIX)" on page 2-6

"Environment variables for network connections" on page 2-45

How utilities attach to shared memory

Database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory through one of the following files.

Operating system	File
UNIX	`$INFORMIXDIR/etc/.infos.`*servername*
Windows	`%INFORMIXDIR%\etc\.infos.`*servername*

The variable **servername** is the value of the DBSERVERNAME parameter in the `onconfig` file. The utilities obtain the **servername** portion of the file name from the **INFORMIXSERVER** environment variable.

The **oninit** process reads the `onconfig` file and creates the file `.infos.servername` when it starts the database server. The file is removed when the database server terminates.

How virtual processors attach to shared memory

The database server virtual processors attach to shared memory during setup. During this process, the database server must satisfy the following two requirements:

- Ensure that all virtual processors can locate and access the same shared-memory segments
- Ensure that the shared-memory segments are located in physical memory locations that are different than the shared-memory segments assigned to other instances of the database server, if any, on the same computer

The database server uses two configuration parameters, SERVERNUM and SHMBASE, to meet these requirements.

When a virtual processor attaches to shared memory, it performs the following major steps:
- Accesses the SERVERNUM parameter from the `onconfig` file
- Uses SERVERNUM to calculate a shared-memory key value
- Requests a shared-memory segment using the shared-memory key value
  The operating system returns the shared-memory identifier for the first shared-memory segment.
- Directs the operating system to attach the first shared-memory segment to its process space at SHMBASE
- Attaches additional shared-memory segments, if required, to be contiguous with the first segment

The following topics describe how the database server uses the values of the SERVERNUM and SHMBASE configuration parameters in the process of attaching shared-memory segments.

Obtain key values for shared-memory segments

The values of the SERVERNUM configuration parameter and *shmkey*, an internally calculated number, determine the unique key value for each shared-memory segment.

To see the key values for shared-memory segments, run the **onstat -g seg** command. For more information, see the sections on SHMADD and the buffer pool in your *IBM Informix Performance Guide*.

When a virtual processor requests that the operating system attach the first shared-memory segment, it supplies the unique key value to identify the segment. In return, the operating system passes back a shared-memory segment identifier associated with the key value. Using this identifier, the virtual processor requests that the operating system attach the segment of shared memory to the virtual-processor address space.

Specify where to attach the first shared-memory segment

The SHMBASE parameter in the `onconfig` file specifies the virtual address where each virtual processor attaches the first, or base, shared-memory segment. Each virtual processor attaches to the first shared-memory segment at the same virtual address. This situation enables all virtual processors within the same database server instance to reference the same locations in shared memory without calculating shared-memory addresses. All shared-memory addresses for an instance of the database server are relative to SHMBASE.

Warning: Do not change the value of SHMBASE.

The value of SHMBASE is sensitive for the following reasons:
- The specific value of SHMBASE depends on the platform and whether the processor is a 32-bit or 64-bit processor. The value of SHMBASE is not an arbitrary number and is intended to keep the shared-memory segments safe when the virtual processor dynamically acquires additional memory space.
- Different operating systems accommodate additional memory at different virtual addresses. Some architectures extend the highest virtual address of the

virtual-processor data segment to accommodate the next segment. In this case, the data segment might grow into the shared-memory segment.

- Some versions of UNIX require the user to specify an SHMBASE parameter of virtual address zero. The zero address informs the UNIX kernel that the kernel picks the best address at which to attach the shared-memory segments. However, not all UNIX architectures support this option. Moreover, on some systems, the selection that the kernel makes might not be the best selection.

For information about SHMBASE, see your IBM Informix machine notes.

Attach additional shared-memory segments

Each virtual processor must attach to the total amount of shared memory that the database server has acquired. After a virtual processor attaches each shared-memory segment, it calculates how much shared memory it has attached and how much remains. The database server facilitates this process by writing a shared-memory header to the first shared-memory segment. Sixteen bytes into the header, a virtual processor can obtain the following data:

- The total size of shared memory for this database server
- The size of each shared-memory segment

To attach additional shared-memory segments, a virtual processor requests them from the operating system in much the same way that it requested the first segment. For the additional segments, however, the virtual processor adds 1 to the previous value of *shmkey*. The virtual processor directs the operating system to attach the segment at the address that results from the following calculation:

```
SHMBASE + (seg_size x number of attached segments)
```

The virtual processor repeats this process until it has acquired the total amount of shared memory.

Given the initial key value of (SERVERNUM * 65536) + *shmkey*, the database server can request up to 65,536 shared-memory segments before it can request a shared-memory key value used by another database server instance on the same computer.

Define the shared-memory lower-boundary address

If your operating system uses a parameter to define the lower boundary address for shared memory, and the parameter is set incorrectly, it can prevent the shared-memory segments from being attached contiguously.

The following figure illustrates the problem. If the lower-boundary address is less than the ending address of the previous segment plus the size of the current segment, the operating system attaches the current segment at a point beyond the end of the previous segment. This action creates a gap between the two segments. Because shared memory must be attached to a virtual processor so that it looks like contiguous memory, this gap creates problems. The database server receives errors when this situation occurs.

To correct the problem, check the operating-system kernel parameter that specifies the lower-boundary address or reconfigure the kernel to allow larger shared-memory segments.
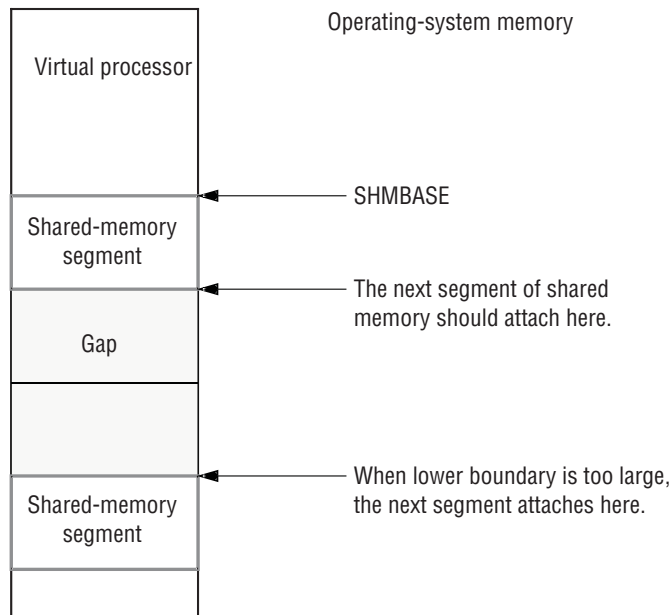
Virtual processor

Operating-system memory

SHMBASE

Shared-memory segment

The next segment of shared memory should attach here.

Gap

When lower boundary is too large, the next segment attaches here.

Shared-memory segment

Figure 6-3. Shared-memory lower-boundary address overview

Resident shared-memory segments

The operating system, as it switches between the processes running on the system, normally swaps the contents of portions of memory to disk. When a portion of memory is designated as *resident*, however, it is not swapped to disk. Keeping frequently accessed data resident in memory improves performance because it reduces the number of disk I/O operations that would otherwise be required to access that data.

The database server requests that the operating system keep the virtual portions in physical memory when the following two conditions exist:

- The operating system supports shared-memory residency.
- The RESIDENT parameter in the onconfig file is set to **-1** or a value that is greater than 0.

Warning: You must consider the use of shared memory by all applications when you consider whether to set the RESIDENT parameter to **-1**. Locking all shared memory for the use of the IBM Informix database server can adversely affect the performance of other applications, if any, on the same computer.

For more information about the RESIDENT configuration parameter, see the *IBM Informix Administrator's Reference*.

Resident portion of shared memory

The resident portion of the database server shared memory stores the following data structures that do not change in size while the database server is running:

- Shared-memory header
- Buffer pool
- Logical-log buffer

- Physical-log buffer
- Lock table

Shared-memory header

The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool.

The shared-memory header also contains pointers to the locations of these structures. When a virtual processor first attaches to shared memory, it reads address information in the shared-memory header for directions to all other structures.

The size of the shared-memory header is about 200 KB, but the size varies depending on the computer platform. You cannot tune the size of the header.

Shared-memory buffer pool

The buffer pool in the resident portion of shared memory contains buffers that store dbspace pages read from disk. The pool of buffers comprises the largest allocation of the resident portion of shared memory.

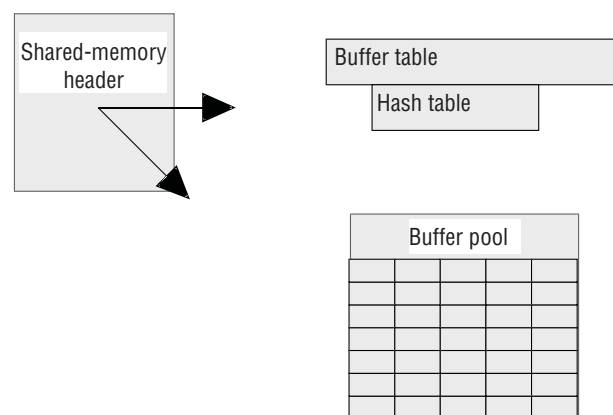The following figure illustrates the shared-memory header and the buffer pool.

Figure 6-4. Shared-memory buffer pool

You use the BUFFERPOOL configuration parameter to specify information about a buffer pool, including the number of buffers in the buffer pool. To allocate the appropriate number of buffers, start with at least four buffers per user. For more than 500 users, the minimum requirement is 2000 buffers. Too few buffers can severely affect performance, so you must monitor the database server and tune the value of buffers to determine an acceptable value. For more information about tuning the number of buffers, see the *IBM Informix Performance Guide*.

If a buffer pool for a non-default page size does not exist, the database server automatically creates a large-page buffer.

If you are creating a dbspace with a non-default page size, the dbspace must have a corresponding buffer pool. For example, if you create a dbspace with a page size of 6 KB, you must create a buffer pool with a size of 6 KB.

Automatic LRU (least recently used) tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

For more information about setting the BUFFERPOOL configuration parameter, see the *IBM Informix Administrator's Reference*.

The status of the buffers is tracked through the buffer table. Within shared memory, buffers are organized into FIFO/LRU buffer queues. Buffer acquisition is managed through the use of latches, called *mutexes*, and lock-access information.

For a description of how LRU queues work, see "FIFO/LRU queues" on page 6-21. For a description of mutexes, see "Mutexes" on page 4-10.

Buffer overflow to the virtual portion
Because the maximum number of buffers in 64-bit addressing can be as large as 2^{31}-1, the resident portion of shared memory might not be able to hold all of the buffers in a large buffer pool. In this case, the virtual portion of database server shared memory might hold some of the buffers.

Buffer size
Each buffer is the size of one database server page.

In general, the database server performs I/O in full-page units, the size of a buffer. The exceptions are I/O performed from big buffers, from blobspace buffers, or from lightweight I/O buffers. (See "Big buffers" on page 6-17 and "Creation of blobpage buffers" on page 6-30.) For information about when to use private buffers, see information about lightweight I/O operations in the *IBM Informix Performance Guide*.

The **onstat -b** command displays information about the buffers. For information about **onstat**, see the *IBM Informix Administrator's Reference*.

Logical-log buffer
The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server. The logical log contains the following five types of log records, in addition to many others:
- SQL data definition statements for all databases
- SQL data manipulation statements for databases that were created with logging
- Record of a change to the logging status of a database
- Record of a checkpoint
- Record of a change to the configuration

The database server uses only one of the logical-log buffers at a time. This buffer is the current logical-log buffer. Before the database server flushes the current logical-log buffer to disk, it makes the second logical-log buffer the current one so that it can continue writing while the first buffer is flushed. If the second logical-log buffer fills before the first one finishes flushing, the third logical-log buffer becomes the current one. This process is illustrated in the following figure.
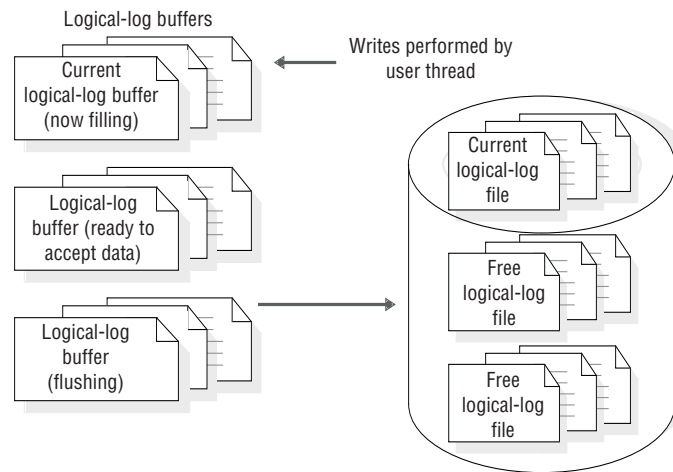
Figure 6-5. The logical-log buffer and its relation to the logical-log files on disk

For a description of how the database server flushes the logical-log buffer, see "Flush the logical-log buffer" on page 6-28.

The LOGBUFF configuration parameter specifies the size of the logical-log buffers. Small buffers can create problems if you store records larger than the size of the buffers (for example, TEXT or BYTE data in dbspaces). The recommended value for the size of a logical log buffer is 64 KB. Whenever the setting is less than the recommended value, the database server suggests a value during server startup. For the possible values that you can assign to this configuration parameter, see the *IBM Informix Administrator's Reference*.

For information about the affect of TEXT and BYTE data on shared memory buffers, see "Buffer large-object data" on page 6-29.

Physical-log buffer

The database server uses the physical-log buffer to hold before-images of some of the modified dbspace pages. The before-images in the physical log and the logical-log records enable the database server to restore consistency to its databases after a system failure.

The physical-log buffer is actually two buffers. Double buffering permits the database server processes to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. For a description of how the database server flushes the physical-log buffer, see "Flush the physical-log buffer" on page 6-26. For information about monitoring the physical-log file, see "Monitor physical and logical-logging activity" on page 16-2.

The PHYSBUFF parameter in the `onconfig` file specifies the size of the physical-log buffers. A write to the physical-log buffer writes exactly one page. If the specified size of the physical-log buffer is not evenly divisible by the page size, the database server rounds the size down to the nearest value that is evenly divisible by the page size. Although some operations require the buffer to be flushed sooner, in general the database server flushes the buffer to the physical-log file on disk when the buffer fills. Thus, the size of the buffer determines how frequently the database server must flush it to disk.

The default value for the physical log buffer size is 512 KB. If you decide to use a smaller value, the database server displays a message indicating that optimal

performance might not be attained. Using a physical log buffer smaller than 512 KB affects performance only, not transaction integrity.

For more information about this configuration parameter, see the *IBM Informix Administrator's Reference*.

High-Availability Data-Replication buffer

Data replication requires two instances of the database server, a primary instance and a secondary instance, running on two computers. If you implement data replication for your database server, the primary database server holds logical-log records in the data replication buffers before it sends them to the secondary database server. A data replication buffer is always the same size as the logical-log buffer. For information about the size of the logical-log buffer, see the preceding topic, "Logical-log buffer" on page 6-10. For more information about how the data replication buffer is used, see "How data replication works" on page 22-1.

Lock table

A lock is created when a user thread writes an entry in the lock table. The lock table is the pool of available locks. A single transaction can own multiple locks. For an explanation of locking and the SQL statements associated with locking, see the *IBM Informix Guide to SQL: Tutorial*.

The following information, which is stored in the lock table, describes the lock:
- The address of the transaction that owns the lock
- The type of lock (exclusive, update, shared, byte, or intent)
- The page or rowid that is locked
- The table space where the lock is placed
- Information about the bytes locked (byte-range locks for smart large objects):
  - Smart-large-object ID
  - Offset into the smart large object where the locked bytes begin
  - The number of bytes locked, starting at the offset

To specify the initial size of the lock table, set the LOCKS configuration parameter. For information about using the LOCKS configuration parameter to specify the number of locks for a session, see the topics about configuration parameters in the *IBM Informix Administrator's Reference* and the topics about configuration effects on memory utilization in your *IBM Informix Performance Guide*.

If the number of locks allocated by sessions exceeds the value specified in the LOCKS configuration parameter, the database server doubles the size of the lock table, up to 15 times. The database server increases the size of the lock table by attempting to double the lock table on each increase. However, the amount added during each increase is limited to a maximum value. For 32-bit platforms, a maximum of 100,000 locks can be added during each increase. Therefore, the total maximum locks allowed for 32-bit platforms is 8,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 100,000 (maximum number of locks added per lock table extension). For 64-bit platforms, a maximum of 1,000,000 locks can be added during each increase. Therefore, the total maximum locks allowed is 500,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 1,000,000 (maximum number of locks added per lock table extension).

Use the DEF_TABLE_LOCKMODE configuration parameter to set the lock mode to page or row for new tables.

Locks can prevent sessions from reading data until after a concurrent transaction is committed or rolled back. For databases created with transaction logging, you can use the USELASTCOMMITTED configuration parameter in the `onconfig` file to specify whether the database server uses the last committed version of the data. The last committed version of the data is the version of the data that existed before any updates occurred. The value you set with the USELASTCOMMITTED configuration parameter overrides the isolation level that is specified in the SET ISOLATION TO COMMITTED READ statement of SQL. For more information about using the USELASTCOMMITTED configuration parameter, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

For more information about using and monitoring locks, see the topics about locking in your *IBM Informix Performance Guide* and the *IBM Informix Guide to SQL: Tutorial*.

Virtual portion of shared memory

The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system. As the database server executes, it automatically attaches additional operating-system segments, as necessary, to the virtual portion.

Management of the virtual portion of shared memory

The database server uses memory *pools* to track memory allocations that are similar in type and size. Keeping related memory allocations in a pool helps to reduce memory fragmentation. It also enables the database server to free a large allocation of memory at one time, as opposed to freeing each piece that makes up the pool.

All sessions have one or more memory pools. When the database server requires memory, it looks first in the specified pool. If insufficient memory is available in a pool to satisfy a request, the database server adds memory from the system pool. If the database server cannot find enough memory in the system pool, it dynamically allocates more segments to the virtual portion.

The database server allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, system catalog, SPL routine caches, SQL statement cache, sort pools, and message buffers) from pools that track free space through a linked list. When the database server allocates a portion of memory, it first searches the pool free-list for a *fragment* of sufficient size. If it finds none, it brings new blocks into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is deleted. When the database server starts a session for a client application, for example, it allocates memory for the session pool. When the session terminates, the database server returns the allocated memory as free fragments.

Size of the virtual portion of shared memory
Use configuration parameters to specify the initial size of the virtual portion of shared memory, the size of segments to be added later, and the amount of memory available for PDQ queries.

To specify the initial size of the virtual shared-memory portion, set the SHMVIRTSIZE configuration parameter. To specify the size of segments that are added later to the virtual shared memory, set the SHMADD and the EXTSHMADD configuration parameter.

To specify the amount of memory available for PDQ queries, set the DS_TOTAL_MEMORY parameter.

If you want to increase the amount of memory that is available for a query that is not a PDQ query and the PDQ priority is set to 0 (zero), you can change the amount in any of the following ways:

• Set the DS_NONPDQ_QUERY_MEM configuration parameter
• Run the **onmode -wm** or the **onmode -wf** command

For example, if you use the **onmode** utility, specify a value as shown in the following example:

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

The minimum value for DS_NONPDQ_QUERY_MEM is 128 KB. The maximum supported value is 25 percent of the value of DS_TOTAL_MEMORY.

Related reference:

"Add a segment to the virtual portion of shared memory" on page 7-7

DS_TOTAL_MEMORY configuration parameter (Administrator's Reference)

DS_NONPDQ_QUERY_MEM configuration parameter (Administrator's Reference)

SHMVIRTSIZE configuration parameter (Administrator's Reference)

SHMADD configuration parameter (Administrator's Reference)

EXTSHMADD configuration parameter (Administrator's Reference)

Components of the virtual portion of shared memory

The virtual portion of shared memory stores the following data:

• Internal tables
• Big buffers
• Session data
• Thread data (stacks and heaps)
• Data-distribution cache
• Dictionary cache
• SPL routine cache
• SQL statement cache
• Sorting pool
• Global pool

Shared-memory internal tables

The database server shared memory contains seven internal tables that track shared-memory resources. The shared-memory internal tables are as follows:

• Buffer table
• Chunk table
• Dbspace table

- Page-cleaner table
- Tblspace table
- Transaction table
- User table

Buffer table: The buffer table tracks the addresses and status of the individual buffers in the shared-memory pool. When a buffer is used, it contains an image of a data or index page from disk. For more information about the purpose and content of a disk page, see "Pages" on page 8-5.

Each buffer in the buffer table contains the following control information, which is required for buffer management:

Buffer status
> Buffer status is described as empty, unmodified, or modified. An unmodified buffer contains data, but the data can be overwritten. A modified (dirty) buffer contains data that must be written to disk before it can be overwritten.

Current lock-access level
> Buffers receive lock-access levels depending on the type of operation that the user thread is executing. The database server supports two buffer lock-access levels: shared and exclusive.

Threads waiting for the buffer
> Each buffer header maintains a list of the threads that are waiting for the buffer and the lock-access level that each waiting thread requires.

Each database server buffer has one entry in the buffer table.

For information about the database server buffers, see "Resident portion of shared memory" on page 6-8. For information about how to monitor the buffers, see "Monitor buffers" on page 7-10.

The database server determines the number of entries in the buffer-table hash table, based on the number of allocated buffers. The maximum number of hash values is the largest power of 2 that is less than the value of **buffers**, which is specified in one of the BUFFERPOOL configuration parameter fields.

Chunk table: The chunk table tracks all chunks in the database server. If mirroring has been enabled, a corresponding mirror chunk table is also created when shared memory is set up. The mirror chunk table tracks all mirror chunks.

The chunk table in shared memory contains information that enables the database server to locate chunks on disk. This information includes the number of the initial chunk and the number of the next chunk in the dbspace. Flags also describe chunk status: mirror or primary; offline, online, or recovery mode; and whether this chunk is part of a blobspace. For information about monitoring chunks, see "Monitor chunks" on page 9-44.

The maximum number of entries in the chunk table might be limited by the maximum number of file descriptors that your operating system allows per process. You can usually specify the number of file descriptors per process with an operating-system kernel-configuration parameter. For details, consult your operating-system manuals.

Dbspace table: The dbspace table tracks storage spaces in the database server. The dbspace-table information includes the following information about each dbspace:

*   Dbspace number
*   Dbspace name and owner
*   Dbspace mirror status (mirrored or not)
*   Date and time that the dbspace was created

If the storage space is a blobspace, flags indicate the media where the blobspace is located: magnetic, removable, or optical media. If the storage space is an sbspace, it contains internal tables that track metadata for smart large objects and large contiguous blocks of pages containing user data.

For information about monitoring dbspaces, see "Monitor disk usage" on page 9-44.

Page-cleaner table: The page-cleaner table tracks the state and location of each of the page-cleaner threads. The number of page-cleaner threads is specified by the CLEANERS configuration parameter in the onconfig file. For advice on how many page-cleaner threads to specify, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

The page-cleaner table always contains 128 entries, regardless of the number of page-cleaner threads specified by the CLEANERS parameter in the onconfig file.

For information about monitoring the activity of page-cleaner threads, see information about the **onstat -F** option in the *IBM Informix Administrator's Reference*.

Tblspace table: The tblspace table tracks all active tblspaces in a database server instance. An active tblspace is one that is currently in use by a database session. Each active table accounts for one entry in the tblspace table. Active tblspaces include database tables, temporary tables, and internal control tables, such as system catalog tables. Each tblspace table entry includes header information about the tblspace, the tblspace name, and pointers to the tblspace **tblspace** in dbspaces on disk. (The shared-memory active tblspace table is different from the tblspace **tblspace**.) For information about monitoring tblspaces, see "Monitor tblspaces and extents" on page 9-48.

The database server manages one tblspace table for each dbspace.

Transaction table: The transaction table tracks all transactions in the database server.

Tracking information derived from the transaction table is shown in the **onstat -x** display. For an example of the output that **onstat -x** displays, see monitoring transactions in your *IBM Informix Performance Guide*.

The database server automatically increases the number of entries in the transaction table, up to a maximum of 32,767, based on the number of current transactions.

For more information about transactions and the SQL statements that you use with transactions, see the *IBM Informix Guide to SQL: Tutorial*, the *IBM Informix Guide to SQL: Reference*, and the *IBM Informix Guide to SQL: Syntax*.

UNIX only: The transaction table also specifically supports the X/Open environment. Support for the X/Open environment requires TP/XA.

User table: The user table tracks all user threads and system threads. Each client session has one primary thread and zero-to-many secondary threads, depending on the level of parallelism specified. System threads include one to monitor and control checkpoints, one to process **onmode** commands, the B-tree scanner threads, and page-cleaner threads.

The database server increases the number of entries in the user table as necessary. You can monitor user threads with the **onstat -u** command.

Big buffers

A big buffer is a single buffer that is made up of several pages. The actual number of pages is platform-dependent. The database server allocates big buffers to improve performance on large reads and writes.

The database server uses a big buffer whenever it writes to disk multiple pages that are physically contiguous. For example, the database server tries to use a big buffer to perform a series of sequential reads (light scans) or to read into shared memory simple large objects that are stored in a dbspace.

Users do not have control over the big buffers. If the database server uses light scans, it allocates big buffers from shared memory.

For information about monitoring big buffers with the **onstat** command, see the topics about configuration effects on I/O activity in your *IBM Informix Performance Guide*.

Session data

When a client application requests a connection to the database server, the database server begins a *session* with the client and creates a data structure for the session in shared memory called the *session-control block*. The session-control block stores the session ID, the user ID, the process ID of the client, the name of the host computer, and various status flags.

The database server allocates memory for session data as necessary.

Thread data

When a client connects to the database server, in addition to starting a session, the database server starts a primary session thread and creates a *thread-control block* for it in shared memory.

The database server also starts internal threads on its own behalf and creates thread-control blocks for them. When the database server switches from running one thread to running another one (a context switch), it saves information about the thread— such as the register contents, program counter (address of the next instruction), and global pointers—in the thread-control block. For more information about the thread-control block and how it is used, see "Context switching" on page 4-7.

The database server allocates memory for thread-control blocks as necessary.

Stacks: Each thread in the database server has its own stack area in the virtual portion of shared memory. For a description of how threads use stacks, see

"Stacks" on page 4-8. For information about how to monitor the size of the stack for a session, see monitoring sessions and threads section in your *IBM Informix Performance Guide*.

The size of the stack space for user threads is specified by the STACKSIZE parameter in the `onconfig` file. The default size of the stack is 32 KB. You can change the size of the stack for all user threads, if necessary, by changing the value of STACKSIZE. For information and a warning on setting the size of the stack, see STACKSIZE in the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

To alter the size of the stack for the primary thread of a specific session, set the **INFORMIXSTACKSIZE** environment variable. The value of **INFORMIXSTACKSIZE** overrides the value of STACKSIZE for a particular user. For information about how to override the stack size for a particular user, see the description of the **INFORMIXSTACKSIZE** environment variable in the *IBM Informix Guide to SQL: Reference*.

To more safely alter the size of stack space, use the **INFORMIXSTACKSIZE** environment variable rather than alter the configuration parameter STACKSIZE. The **INFORMIXSTACKSIZE** environment variable affects the stack space for only one user, and it is less likely to affect new client applications that initially were not measured.

Heaps: Each thread has a heap to hold data structures that it creates while it is running. A heap is dynamically allocated when the thread is created. The size of the thread heap is not configurable.

Data-distribution cache

The database server uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost. When the database server accesses the distribution statistics for a specific column the first time, it reads the distribution statistics from the **sysdistrib** system catalog table on disk and stores the statistics in the data-distribution cache. These statistics can then be read for the optimization of subsequent queries that access the column.

Performance improves if these statistics are efficiently stored and accessed from the data-distribution cache. You can configure the size of the data-distribution cache with the DS_HASHSIZE and DS_POOLSIZE configuration parameters. For information about changing the default size of the data-distribution cache, see the topics about queries and the query optimizer in your *IBM Informix Performance Guide*.

Dictionary cache

When a session executes an SQL statement that requires access to a system catalog table, the database server reads data from the system catalog tables. The database server stores the catalog data for each queried table in structures that it can access more efficiently during subsequent queries on that table. These structures are created in the virtual portion of shared memory for use by all sessions. These structures constitute the dictionary cache.

You can configure the size of the dictionary cache with the DD_HASHSIZE and DD_HASHMAX configuration parameters. For more information about these parameters, see the chapter on configuration effects on memory in your *IBM Informix Performance Guide*.

SQL statement cache

The SQL statement cache reduces memory usage and preparation time for queries. The database server uses the SQL statement cache to store parsed and optimized SQL statements that a user executes. When users execute a statement stored in the SQL statement cache, the database server does not parse and optimize the statement again, so performance improves.

For more information, see "Set SQL statement cache parameters" on page 7-5. For details on how these parameters affect the performance of the SQL statement cache, see the *IBM Informix Performance Guide*.

Sort memory

The following database operations can use large amounts of the virtual portion of shared memory to sort data:

- Decision-support queries that involve joins, groups, aggregates and sort operations
- Index builds
- UPDATE STATISTICS statement in SQL

The amount of virtual shared memory that the database server allocates for a sort depends on the number of rows to be sorted and the size of the row, along with other factors.

For information about parallel sorts, see your *IBM Informix Performance Guide*.

SPL routine and the UDR cache

The database server converts an SPL routine to executable format and stores the routine in the UDR cache, where it can be accessed by any session.

When a session is required to access an SPL routine or other user-defined routine for the first time, the database server reads the definition from the system catalog tables and stores the definition in the UDR cache.

You can configure the size of the UDR cache with the PC_HASHSIZE and PC_POOLSIZE configuration parameters. For information about changing the default size of the UDR cache, see the chapter on queries and the query optimizer in your *IBM Informix Performance Guide*.

Global pool

The global pool stores structures that are global to the database server. For example, the global pool contains the message queues where poll threads for network communications deposit messages from clients. The **sqlexec** threads pick up the messages from the global pool and process them.

For more information, see the sections on network buffer pools and virtual portion of shared memory in your *IBM Informix Performance Guide*.

Communications portion of shared memory (UNIX)

The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you set up shared memory. The communications portion contains the message buffers for local client applications that use shared memory to communicate with the database server.

The size of the communications portion of shared memory equals approximately 12 KB multiplied by the expected number of connections required for shared-memory communications (**nettype** `ipcshm`). If **nettype** `ipcshm` is not present, the expected number of connections defaults to 50. For information about how a client attaches to the communications portion of shared memory, see "How a client attaches to the communications portion (UNIX)" on page 6-5.

Related reference:

"Shared-memory connections (UNIX)" on page 2-6

Virtual-extension portion of shared memory

The virtual-extension portion of shared memory contains additional virtual segments and virtual-extension segments.

Virtual-extension segments contain thread heaps for DataBlade modules and user-defined routines that run in user-defined virtual processors.

The EXTSHMADD configuration parameter sets the size of virtual-extension segments. The SHMADD and SHMTOTAL configuration parameters apply to the virtual-extension portion of shared memory, just as they do to the other portions of shared memory.

Concurrency control

The database server threads that run on the same virtual processor and on separate virtual processors share access to resources in shared memory. When a thread writes to shared memory, it uses mechanisms called *mutexes* and *locks* to prevent other threads from simultaneously writing to the same area. A mutex gives a thread the right to access a shared-memory resource. A lock prevents other threads from writing to a buffer until the thread that placed the lock is finished with the buffer and releases the lock.

Shared-memory mutexes

The database server uses *mutexes* to coordinate threads as they attempt to modify data in shared memory. Every modifiable shared-memory resource is associated with a mutex. Before a thread can modify a shared-memory resource, it must first acquire the mutex associated with that resource. After the thread acquires the mutex, it can modify the resource. When the modification is complete, the thread releases the mutex.

If a thread tries to obtain a mutex and finds that it is held by another thread, the incoming thread must wait for the mutex to be released.

For example, two threads can attempt to access the same slot in the chunk table, but only one can acquire the mutex associated with the chunk table. Only the thread that holds the mutex can write its entry in the chunk table. The second thread must wait for the mutex to be released and then acquire it.

For information about monitoring mutexes (which are also called latches), see "Monitor the shared-memory profile and latches" on page 7-9.

Shared-memory buffer locks

A primary benefit of shared memory is the ability of database server threads to share access to disk pages stored in the shared-memory buffer pool. The database

server maintains thread isolation while it achieves this increased concurrency through a strategy for locking the data buffers.

Types of buffer locks

The database server uses two types of locks to manage access to shared-memory buffers:

- Share locks
- Exclusive locks

Each of these lock types enforces the required level of thread isolation during execution.

Share lock: A buffer is in share mode, or has a share lock, if multiple threads have access to the buffer to read the data but none intends to modify the data.

Exclusive lock: A buffer is in exclusive mode, or has an exclusive lock, if a thread demands exclusive access to the buffer. All other thread requests that access the buffer are placed in the wait queue. When the executing thread is ready to release the exclusive lock, it wakes the next thread in the wait queue.

Database server thread access to shared buffers

Database server threads access shared buffers through a system of queues, using mutexes and locks to synchronize access and protect data.

FIFO/LRU queues

A buffer holds data for the purpose of caching. The database server uses the least-recently used (LRU) queues to replace the cached data. IBM Informix also has a first-in first-out (FIFO) queue. When you set the number of LRU queues, you are actually setting the number of FIFO/LRU queues.

Use the BUFFERPOOL configuration parameter to specify information about the buffer pool, including information about the number of LRU queues to create when database server shared memory is set up and values for **lru_min_dirty** and **lru_max_dirty**, which control how frequently the shared-memory buffers are flushed to disk.

To improve transaction throughput, increase the **lru_min_dirty** and **lru_max_dirty** values. However, do not change the gap between the **lru_min_dirty** and **lru_max_dirty** values.

Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters before Version 10.0 is now specified using the BUFFERPOOL configuration parameter.

Components of LRU queue

Each LRU queue is composed of a pair of linked lists, as follows:

- FLRU (free least-recently used) list, which tracks free or unmodified pages in the queue
- MLRU (modified least-recently used) list, which tracks modified pages in the queue

The free or unmodified page list is called the FLRU queue of the queue pair, and the modified page list is called the MLRU queue. The two separate lists eliminate the task of searching a queue for a free or unmodified page. The following figure

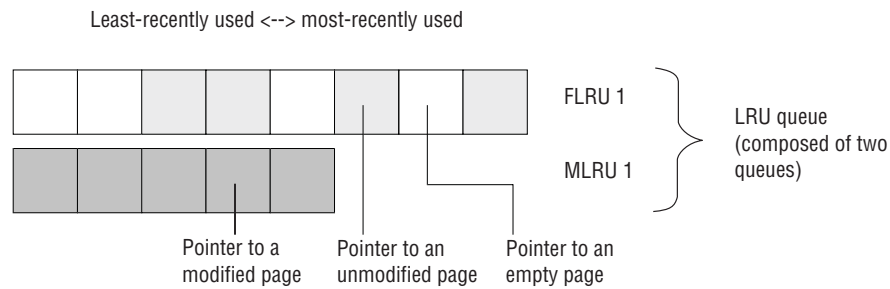illustrates the structure of the LRU queues.

Figure 6-6. LRU queue

Pages in least-recently used order

When the database server processes a request to read a page from disk, it must decide which page to replace in memory. Rather than select a page randomly, the database server assumes that recently referenced pages are more likely to be referenced in the future than pages that it has not referenced for some time. Thus, rather than replacing a recently accessed page, the database server replaces a least-recently accessed page. By maintaining pages in least-recently to most-recently used order, the database server can easily locate the least-recently used pages in memory.

LRU queues and buffer-pool management

Before processing begins, all page buffers are empty, and every buffer is represented by an entry in one of the FLRU queues. The buffers are evenly distributed among the FLRU queues. To calculate the number of buffers in each queue, divide the total number of buffers by the number of LRU queues. The number of buffers and LRU queues are specified in the BUFFERPOOL configuration parameter.

When a user thread is required to acquire a buffer, the database server randomly selects one of the FLRU queues and uses the oldest or least-recently used entry in the list. If the least-recently used page can be latched, that page is removed from the queue.

If the FLRU queue is locked, and the end page cannot be latched, the database server randomly selects another FLRU queue.

If a user thread is searching for a specific page in shared memory, it obtains the LRU-queue location of the page from the control information stored in the buffer table.

After an executing thread finishes its work, it releases the buffer. If the page has been modified, the buffer is placed at the most-recently used end of an MLRU queue. If the page was read but not modified, the buffer is returned to the FLRU queue at its most-recently used end. For information about how to monitor LRU queues, see "Monitor buffer-pool activity" on page 7-12.

Number of LRU queues to configure

Multiple LRU queues have two purposes:
- They reduce user-thread contention for the queues.
- They allow multiple cleaners to flush pages from LRU queues and maintain the percentage of dirty pages at an acceptable level.

Initial values for the LRUS are recommended based on the number of CPUs that are available on your computer. If your computer is a uniprocessor, start by setting the **lrus** value in the BUFFERPOOL configuration parameter to 4. If your computer is a multiprocessor, use the following formula:

```
LRUS = max(4, (number_CPU_VPs))
```

After you provide an initial value for **lrus** in the BUFFERPOOL configuration parameter, monitor your LRU queues with **onstat -R**. If you find that the percentage of dirty LRU queues consistently exceeds the value specified for **lru_max_dirty**, increase the value specified for **lrus** to add more LRU queues.

For example, suppose you set **lru_max_dirty** to 70 and find that your LRU queues are consistently 75 percent dirty. Consider increasing the value of the lrus. If you increase the number of LRU queues, you shorten the length of the queues, thereby reducing the work of the page cleaners. However, you must allocate a sufficient number of page cleaners with the CLEANERS configuration parameter, as explained in the following section. Retain the same gap between **lru_max_dirty** and **lru_min_dirty**.

Important: Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters before Version 10.0 is now specified using the BUFFERPOOL configuration parameter.

Number of cleaners to allocate

In general, you must configure one cleaner for each disk that your applications update frequently. However, you must also consider the length of your LRU queues and frequency of checkpoints, as explained in the following paragraphs.

In addition to insufficient LRU queues, another factor that influences whether page cleaners keep up with the number of pages that require cleaning is whether you have enough page-cleaner threads allocated. The percent of dirty pages might exceed the BUFFERPOOL value specified for **lru_max_dirty** in some queues because no page cleaners are available to clean the queues. After a while, the page cleaners might be too far behind to catch up, and the buffer pool becomes dirtier than the percent that you specified in **lru_max_dirty**.

For example, suppose that the CLEANERS parameter is set to 8, and you increase the number of LRU queues from 8 to 12. You can expect little in the way of a performance gain because the 8 cleaners must now share the work of cleaning an additional 4 queues. If you increase the number of CLEANERS to 12, each of the now-shortened queues can be more efficiently cleaned by a single cleaner.

Setting CLEANERS too low can cause performance to suffer whenever a checkpoint occurs because page cleaners must flush all modified pages to disk during checkpoints. If you do not configure a sufficient number of page cleaners, checkpoints take longer, causing overall performance to suffer.

For more information, see "Flush buffer-pool buffers" on page 6-26.

Number of pages added to the MLRU queues

Periodically, the page-cleaner threads flush the modified buffers in an MLRU queue to disk. To specify the point at which cleaning begins, use the BUFFERPOOL configuration parameter to specify a value for **lru_max_dirty**.

By specifying when page cleaning begins, the **lru_max_dirty** value limits the number of page buffers that can be appended to an MLRU queue. The initial

setting of **lru_max_dirty** is 60.00, so page cleaning begins when 60 percent of the buffers managed by a queue are modified.

In practice, page cleaning begins under several conditions, only one of which is when an MLRU queue reaches the value of **lru_max_dirty**. For more information about how the database server performs buffer-pool flushing, see "Flush data to disk" on page 6-25.

The following example shows how the value of **lru_max_dirty** is applied to an LRU queue to specify when page cleaning begins and thereby limit the number of buffers in an MLRU queue.

```
Buffers specified as 8000
lrus specified as 8
lru_max_dirty specified as 60 percent

Page cleaning begins when the number of buffers in the MLRU
   queue is equal to lru_max_dirty.

Buffers per lru queue = (8000/8) = 1000

Max buffers in MLRU queue and point at which page cleaning
   begins: 1000 x 0.60 = 600
```

End of MLRU cleaning

You can also specify the point at which MLRU cleaning can end. The **lru_min_dirty** value in the BUFFERPOOL configuration parameter specifies the acceptable percentage of buffers in an MLRU queue. For example, if **lru_min_dirty** is set to 50.00, page cleaning is not required when 50 percent of the buffers in an LRU queue are modified. In practice, page cleaning can continue beyond this point, as directed by the page-cleaner threads.

The following example shows how the value of **lru_min_dirty** is applied to the LRU queue to specify the acceptable percent of buffers in an MLRU queue and the point at which page cleaning ends.

```
Buffers specified as 8000
lrus specified as 8
lru_min_dirty specified as 50 percent

The acceptable number of buffers in the MLRU queue and
   the point at which page cleaning can end is equal
   to lru_min_dirty.

Buffers per LRU queue = (8000/8) = 1000

Acceptable number of buffers in MLRU queue and the point
   at which page cleaning can end: 1000 x .50 = 500
```

You can use decimals for the **lru_max_dirty** and the **lru_min_dirty** values. For example, if you set **lru_max_dirty** to 1.0333 and **lru_min_dirty** to 1.0, this triggers the LRU to write at 3,100 dirty buffers and to stop at 3,000 dirty buffers.

For more information about how the database server flushes the buffer pool, see "Flush data to disk" on page 6-25.

Automatic read-ahead operations

The server automatically reads several pages ahead of the current pages that are being processed for a query, unless you disable automatic read ahead operations. Reading ahead enables applications to run faster because they spend less time waiting for disk I/O.

Automatic *read-ahead* requests for pages to be brought into the bufferpool cache during sequential scans of data records improves the performance of a query, including OLTP queries and index scans, when the server detects that the query is encountering I/O.

By default, the database server automatically determines when to issue read-ahead requests and when to stop based on when the query is encountering i/o from disk:

*   If queries encounter I/O, the server issues read-ahead requests to improve the performance of the query. This performance improvement occurs because read-ahead requests can greatly increase the speed of database processing by compensating for the slowness of I/O processing relative to the speed of CPU processing.
*   If queries are mostly cached, the server detects that no I/O is occurring and does not read ahead.

Use the AUTO_READAHEAD configuration parameter to change the automatic read-ahead mode or to disable automatic read ahead for a query. You can:

*   Dynamically change the value of the AUTO_READAHEAD configuration parameter by running an **onmode -wm** or **onmode -wf** command.
*   Run a SET ENVIRONMENT AUTO_READAHEAD statement to change the mode or enable or disable automatic read-ahead for a session.

You can use the **onstat -p** command to view database server reads and writes and monitor number of times that a thread was required to wait for a shared-memory latch. The RA-pgsused output field shows the number of pages used that the database server read ahead and monitor the database server use of read-ahead.

Use the **onstat -g rah** command to display statistics about read-ahead requests.

Related reference:

⇨ AUTO_READAHEAD configuration parameter (Administrator's Reference)

⇨ RA_THRESHOLD configuration parameter (Administrator's Reference)

⇨ RA_PAGES configuration parameter (Administrator's Reference)

⇨ onstat -p command: Print profile counts (Administrator's Reference)

⇨ onstat -g rah command: Print read-ahead request statistics (Administrator's Reference)

Database server thread access to buffer pages

The database server uses shared-lock buffering to allow more than one database server thread to access the same buffer concurrently in shared memory.

The database server uses two types of buffer locks to provide this concurrency without a loss in thread isolation. The two types of lock access are share and exclusive. (For more information, see "Types of buffer locks" on page 6-21.)

Flush data to disk

Writing a buffer to disk is called *buffer flushing*. When a user thread modifies data in a buffer, it marks the buffer as *dirty*. When the database server flushes the buffer to disk, it subsequently marks the buffer as *not dirty* and allows the data in the buffer to be overwritten.

The database server flushes the following buffers:

- Buffer pool (covered in this section)
- Physical-log buffer

  See "Flush the physical-log buffer."
- Logical-log buffer

  See "Flush the logical-log buffer" on page 6-28.

Page-cleaner threads manage buffer flushing. The database server always runs at least one page-cleaner thread. If the database server is configured for more than one page-cleaner thread, the LRU queues are divided among the page cleaners for more efficient flushing. For information about specifying how many page-cleaner threads the database server runs, see the CLEANERS configuration parameter in the *IBM Informix Administrator's Reference*.

Flushing the physical-log buffer, the modified shared-memory page buffers, and the logical-log buffer must be synchronized with page-cleaner activity according to specific rules designed to maintain data consistency.

Flush buffer-pool buffers

Flushing of the buffers is initiated by any one of the following conditions:

- The number of buffers in an MLRU queue reaches the number specified by the **lru_max_dirty** value in the BUFFERPOOL configuration parameter.
- The page-cleaner threads cannot keep up. In other words, a user thread must acquire a buffer, but no unmodified buffers are available.
- The database server must execute a checkpoint. (See "Checkpoints" on page 15-4.)

Automatic LRU tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

Flush before-images first

The before-images of modified pages are flushed to disk before the modified pages themselves.

In practice, the physical-log buffer is flushed first and then the buffers that contain modified pages. Therefore, even when a shared-memory buffer page must be flushed because a user thread is trying to acquire a buffer but none is available (a foreground write), the buffer pages cannot be flushed until the before-image of the page has been written to disk.

Flush the physical-log buffer

The database server temporarily stores before-images of some of the modified disk pages in the physical-log buffer. If the before-image is written to the physical-log buffer but not to the physical log on disk, the server flushes the physical-log buffer to disk before flushing the modified page to disk.

The database server always flushes the contents of the physical-log buffer to disk before any data buffers.

The following events cause the active physical-log buffer to flush:

- The active physical-log buffer becomes full.

- A modified page in shared memory must be flushed, but the before-image is still in the active physical-log buffer.
- A checkpoint occurs.

The database server uses only one of the two physical-log buffers at a time. This buffer is the active (or current) physical-log buffer. Before the database server flushes the active physical-log buffer to disk, it makes the other buffer the active physical-log buffer so that the server can continue writing to a buffer while the first buffer is being flushed.

Both the physical-log buffer and the physical log help maintain the physical and logical consistency of the data. For information about physical logging, checkpoints, and fast recovery, see Chapter 15, "Physical logging, checkpoints, and fast recovery," on page 15-1.

Synchronize buffer flushing

When shared memory is first set up, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.

Describing flushing activity

To provide you with information about the specific condition that prompted buffer-flushing activity, the database server defines three types of writes and counts how often each write occurs:
- Foreground write
- LRU write
- Chunk write

To display the write counts that the database server maintains, use **onstat -F** as described in the *IBM Informix Administrator's Reference*.

If you implement mirroring for the database server, data is always written to the primary chunk first. The write is then repeated on the mirror chunk. Writes to a mirror chunk are included in the counts. For more information about monitoring the types of writes that the database server performs, see "Monitor buffer-pool activity" on page 7-12.

Foreground write

Whenever an **sqlexec** thread writes a buffer to disk, it is termed a *foreground write*. A foreground write occurs when an **sqlexec** thread searches through the LRU queues on behalf of a user but cannot locate an empty or unmodified buffer. To make space, the **sqlexec** thread flushes pages, one at a time, to hold the data to be read from disk. (For more information, see "FIFO/LRU queues" on page 6-21.)

If the **sqlexec** thread must perform buffer flushing just to acquire a shared-memory buffer, performance can suffer. Foreground writes must be avoided. To display a count of the number of foreground writes, run **onstat -F**. If you find that foreground writes are occurring on a regular basis, tune the value of the page-cleaning parameters. Either increase the number of page cleaners or decrease the BUFFERPOOL **lru_max_dirty** value.

LRU write

Unlike foreground writes, LRU writes are performed by page cleaners rather than by **sqlexec** threads. The database server performs LRU writes as background

writes that typically occur when the percentage of dirty buffers exceeds the percent that is specified for **lru_max_dirty** in the BUFFERPOOL configuration parameter.

In addition, a foreground write can trigger an LRU write. When a foreground write occurs, the **sqlexec** thread that performed the write alerts a page-cleaner to wake up and clean the LRU for which it performed the foreground write.

In an appropriately tuned system, page cleaners ensure that enough unmodified buffer pages are available for storing pages to be read from disk. Thus, **sqlexec** threads that perform a query are not required to flush a page to disk before they read in the disk pages required by the query. This condition can result in significant performance gains for queries that do not make use of foreground writes.

LRU writes are preferred over foreground writes because page-cleaner threads perform buffer writes much more efficiently than **sqlexec** threads do. To monitor both types of writes, use **onstat -F**.

Chunk write

Chunk writes are commonly performed by page-cleaner threads during a checkpoint or, possibly, when every page in the shared-memory buffer pool is modified. Chunk writes, which are performed as sorted writes, are the most efficient writes available to the database server.

During a chunk write, each page-cleaner thread is assigned to one or more chunks. Each page-cleaner thread reads through the buffer headers and creates an array of pointers to pages that are associated with its specific chunk. (The page cleaners have access to this information because the chunk number is contained within the physical page number address, which is part of the page header.) This sorting minimizes head movement (disk seek time) on the disk and enables the page-cleaner threads to use the big buffers during the write, if possible.

In addition, because user threads must wait for the checkpoint to complete, the page-cleaner threads are not competing with many threads for CPU time. As a result, the page-cleaner threads can finish their work with less context switching.

Flush the logical-log buffer

The database server uses the shared-memory logical-log buffer as temporary storage for records that describe modifications to database server pages. From the logical-log buffer, these records of changes are written to the current logical-log file on disk and eventually to the logical-log backup media. For a description of logical logging, see Chapter 13, "Logical log," on page 13-1.

Five events cause the current logical-log buffer to flush:
- The current logical-log buffer becomes full.
- A transaction is prepared or committed in a database with unbuffered logging.
- A nonlogging database session terminates.
- A checkpoint occurs.
- A page is modified that does not require a before-image in the physical log.

The following topics explain each of these events in detail.

After a transaction is prepared or terminated in a database with unbuffered logging

The following log records cause flushing of the logical-log buffers in a database with unbuffered logging:

- COMMIT
- PREPARE
- XPREPARE
- ENDTRANS

For a comparison of buffered versus unbuffered logging, see the SET LOG statement in the *IBM Informix Guide to SQL: Syntax*.

When a session that uses nonlogging databases or unbuffered logging terminates

Even for nonlogging databases, the database server logs certain activities that alter the database schema, such as the creation of tables or extents. When the database server terminates sessions that use unbuffered logging or nonlogging databases, the logical-log buffer is flushed to make sure that any logging activity is recorded.

When a checkpoint occurs

For a detailed description of the events that occur during a checkpoint, see "Checkpoints" on page 15-4.

When a page is modified that does not require a before-image in the physical-log file

When a page is modified that does not require a before-image in the physical log, the logical-log buffer must be flushed before that page is flushed to disk.

Buffer large-object data

Simple large objects (TEXT or BYTE data) can be stored in either dbspaces or blobspaces. Smart large objects (CLOB or BLOB data) are stored only in sbspaces. The database server uses different methods to access each type of storage space. The following topics describe buffering methods for each.

Write simple large objects

The database server writes simple large objects to disk pages in a dbspace in the same way that it writes any other data type. For more information, see "Flush data to disk" on page 6-25.

You can also assign simple large objects to a blobspace. The database server writes simple large objects to a blobspace differently from the way that it writes other data to a shared-memory buffer and then flushes it to disk. For a description of blobspaces, see the chapter on disk structure and storage in the *IBM Informix Administrator's Reference*.

Blobpages and shared memory

Blobspace blobpages store large amounts of data. Consequently, the database server does not create or access blobpages by way of the shared-memory buffer pool, and it does not write blobspace blobpages to either the logical or physical logs.

If blobspace data passed through the shared-memory pool, it might dilute the effectiveness of the pool by driving out index pages and data pages. Instead, blobpage data is written directly to disk when it is created.

To reduce logical-log and physical-log traffic, the database server writes blobpages from magnetic media to dbspace backup tapes and logical-log backup tapes in a different way than it writes dbspace pages. For a description of how blobspaces are logged, see "Log blobspaces and simple large objects" on page 13-6.

Blobpages stored on optical media are not written to dbspace and logical-log backup tapes due to the high reliability of optical media.

Creation of simple large objects

When simple-large-object data is written to disk, the row to which it belongs might not exist yet. During an insert, for example, the simple large object is transferred before the rest of the row data. After the simple large object is stored, the data row is created with a 56-byte descriptor that points to its location. For a description of how simple large objects are stored physically, see the structure of a dbspace blobpage in the disk storage and structure chapter of the *IBM Informix Administrator's Reference*.

Creation of blobpage buffers

To receive simple large object data from the application process, the database server creates a pair of blobspace buffers, one for reading and one for writing, each the size of one blobspace blobpage. Each user has only one set of blobspace buffers and, therefore, can access only one simple large object at a time.

Simple large object data is transferred from the client-application process to the database server in 1 KB segments. The database server begins filling the blobspace buffers with the 1 KB pieces and attempts to buffer two blobpages at a time. The database server buffers two blobpages so that it can determine when to add a forwarding pointer from one page to the next. When it fills the first buffer and discovers that more data remains to transfer, it adds a forward pointer to the next page before it writes the page to disk. When no more data remains to transfer, the database server writes the last page to disk without a forward pointer.

When the thread begins writing the first blobspace buffer to disk, it attempts to perform the I/O based on the user-defined blobpage size. For example, if the blobpage size is 32 KB, the database server attempts to read or write the data in 32,768-byte increments. If the underlying hardware (such as the disk controller) cannot transfer this amount of data in a single operation, the operating-system kernel loops internally (in kernel mode) until the transfer is complete.

The blobspace buffers remain until the thread that created them is finished. When the simple large object is written to disk, the database server deallocates the pair of blobspace buffers. The following figure illustrates the process of writing a simple large object to a blobspace.
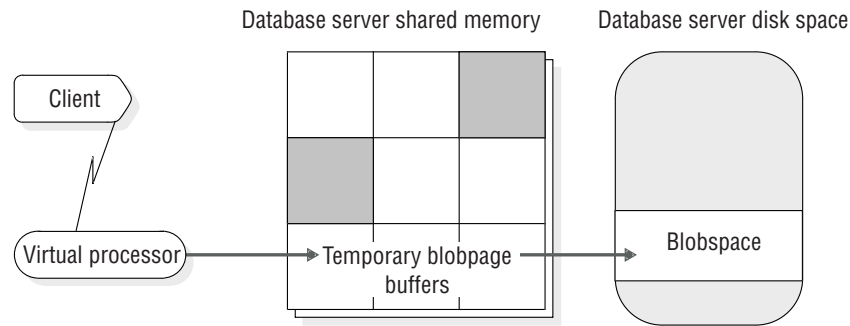
Figure 6-7. Writing simple large object to a blobspace

Blobspace blobpages are allocated and tracked with the free-map page. Links that connect the blobpages and pointers to the next blobpage segments are created as necessary.

A record of the operation (insert, update, or delete) is written to the logical-log buffer.

Access smart large objects

The database server accesses smart large objects through the shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. However, the user-data portion of a smart large object is buffered at a lower priority than normal buffer pages to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

A smart large object is stored in an sbspace. You cannot store simple large objects in an sbspace, and you cannot store smart large objects in a blobspace. An sbspace consists of a user-data area and a metadata area. The user-data area contains the smart-large-object data. The metadata area contains information about the content of the sbspace. For more information about sbspaces, see "Sbspaces" on page 8-13.

Because smart large objects pass through the shared-memory buffer pool and can be logged, you must consider them when you allocate buffers. Use the BUFFERPOOL configuration parameter to allocate shared-memory buffers. As a general rule, try to have enough buffers to hold two smart-large-object pages for each concurrently open smart large object. (The additional page is available for read-ahead purposes.) For more information about tuning buffers for smart large objects, see your *IBM Informix Performance Guide*.

Use the LOGBUFF configuration parameter to specify the size of the logical-log buffer. For information about setting each of the following configuration parameters, see the *IBM Informix Administrator's Reference*:
- BUFFERPOOL
- LOGBUFF

The user-data area of smart large objects that are logged does not pass through the physical log, so changing the PHYSBUFF parameter is not required for smart large objects.

For more information about the structure of an sbspace, see sbspace structure in the disk structures and storage chapter of the *IBM Informix Administrator's*

Reference. For information about creating an sbspace, see information about the
onspaces utility in the *IBM Informix Administrator's Reference*.

Memory use on 64-bit platforms

With 64-bit addressing, you can have larger buffer pools to reduce the amount of
I/O operations to obtain data from disks. Because 64-bit platforms allow for larger
memory-address space, the maximum values for the following memory-related
configuration parameters are larger on 64-bit platforms:

*   BUFFERPOOL
*   CLEANERS
*   DS_MAX_QUERIES
*   DS_TOTAL_MEMORY
*   LOCKS
*   LRUS
*   SHMADD
*   SHMVIRTSIZE

The machine notes for each 64-bit platform lists the maximum values for these
configuration parameters and platform-specific parameters such as SHMMAX. For
more information about the configuration parameters, see the *IBM Informix
Administrator's Reference* and the chapter on shared memory in the *IBM Informix
Performance Guide*.

Chapter 7. Manage shared memory

These topics inform you how to perform the following tasks, which concern managing shared memory:

- Setting the shared-memory configuration parameters
- Setting up shared memory
- Turning residency on or off for the resident portion of the database server shared memory
- Adding a segment to the virtual portion of shared memory
- Reserving memory for critical activities
- Maintaining a targeted amount of memory in applications with memory limitations
- Monitoring shared memory

These topics do not cover the DS_TOTAL_MEMORY configuration parameter. This parameter places a ceiling on the allocation of memory for decision-support queries. For information about this parameter, see your *IBM Informix Performance Guide*.

Set operating-system shared-memory configuration parameters

Several operating-system configuration parameters can affect the use of shared memory by the database server. Parameter names are not provided because names vary among platforms, and not all parameters exist on all platforms. The following list describes these parameters by function:

- Maximum operating-system shared-memory segment size, expressed in bytes or KB
- Minimum shared-memory segment size, expressed in bytes
- Maximum number of shared-memory identifiers
- Lower-boundary address for shared memory
- Maximum number of attached shared-memory segments per process
- Maximum amount of systemwide shared memory

UNIX only:
- Maximum number of semaphore identifiers
- Maximum number of semaphores
- Maximum number of semaphores per identifier

On UNIX, the machine notes file contains recommended values that you use to configure operating-system resources. Use these recommended values when you configure the operating system. For information about how to set these operating-system parameters, consult your operating-system manuals.

For specific information about your operating-system environment, see the machine notes file that is provided with the database server.

Maximum shared-memory segment size

When the database server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible. The first

segment size that the database server tries to acquire is the size of the portion that it is allocating (resident, virtual, or communications), rounded up to the nearest multiple of 8 KB.

The database server receives an error from the operating system if the requested segment size exceeds the maximum size allowed. If the database server receives an error, it divides the requested size by two and tries again. Attempts at acquisition continue until the largest segment size that is a multiple of 8 KB can be created. Then the database server creates as many additional segments as it requires.

Using more than two gigabytes of memory (Windows)

The database server can access shared-memory segments larger than two gigabytes on Windows. However, you must enable this feature with an entry in the Windows boot file.

To add the entry, edit the `boot.ini` file (located in the top level, or root directory). You can either add a new boot option or use the currently existing boot option. To enable support for more than two gigabytes, add the following text to the end of the boot line:

```
/3GB
```

The following example has support for more than two gigabytes enabled:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows NT
Workstation Version 4.00"
/3GB
```

The maximum size of the shared-memory segment depends on the operating system, but it is approximately 3 gigabytes for Windows without additional drivers.

Maximum number of shared-memory identifiers (UNIX)

Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory. The operating system identifies each shared-memory segment with a shared-memory identifier. For most operating systems, virtual processors receive identifiers on a first-come, first-served basis, up to the limit that is defined for the operating system as a whole. For more information about shared-memory identifiers, see "How virtual processors attach to shared memory" on page 6-5.

You might be able to calculate the maximum amount of shared memory that the operating system can allocate by multiplying the number of shared-memory identifiers by the maximum shared-memory segment size.

Semaphores (UNIX)

The database server operation requires one UNIX semaphore for each virtual processor, one for each user who connects to the database server through shared memory (**ipcshm** protocol), six for database server utilities, and sixteen for other purposes.

Set database server shared-memory configuration parameters

You can modify the configuration parameters that affect the resident or virtual portion of shared memory.

You can use a text editor to modify the shared-memory configuration parameters. For a list of these configuration parameters, see onconfig Portal: Configuration parameters by functional category

On UNIX, you must be **root** or user **informix** to use either method. On Windows, you must be a user in the **Informix Admin** group.

Set parameters for resident shared memory

The following list contains parameters in the `onconfig` file that specify the configuration of the buffer pool and the internal tables in the resident portion of shared memory. Before any changes that you make to the configuration parameters take effect, you must shut down and restart the database server. For a description of the configuration parameters, see the *IBM Informix Administrator's Reference*.

BUFFERPOOL
Specifies information about the buffer pool that must be defined for each different page size that a dbspace uses.

LOCKS
Specifies the initial number of locks for database objects; for example, rows, key values, pages, and tables.

LOGBUFF
Specifies the size of the logical-log buffers.

PHYSBUFF
Specifies the size of the physical-log buffers.

RESIDENT
Specifies residency for the resident portion of the database server shared memory.

SERVERNUM
Specifies a unique identification number for the database server on the local host computer.

SHMTOTAL
Specifies the total amount of memory to be used by the database server.

Set parameters for virtual shared memory

Several configuration parameters affect the virtual portion of shared memory.

The following list contains the configuration parameters that you use to configure the virtual portion of shared memory:

DS_HASHSIZE
Number of hash buckets for lists in the data-distribution cache.

DS_POOLSIZE
Maximum number of entries in the data-distribution cache.

PC_HASHSIZE
Specifies the number of hash buckets for the UDR cache and other caches that the database server uses.

PC_POOLSIZE
 Specifies the number of UDRs (SPL routines and external routines) that can be stored in the UDR cache. In addition, this parameter specifies the size of other database server caches, such as the typename cache and the opclass cache.

SHMADD
 Specifies the size of dynamically added shared-memory segments.

SHMNOACCES
 Specifies a list of virtual memory address ranges that are not used to attach shared memory. Use this parameter to avoid conflicts with other processes.

EXTSHMADD
 Specifies the size of a virtual-extension segment added when a user-defined routine or a DataBlade routine runs in a user-defined virtual processor.

SHMTOTAL
 Specifies the total amount of memory to be used by the database server.

SHMVIRTSIZE
 Specifies the initial size of the virtual portion of shared memory.

STACKSIZE
 Specifies the stack size for the database server user threads.

Related concepts:

▶ Changing the UDR cache (Performance Guide)

Related reference:

▶ Configuration parameters that affect memory utilization (Performance Guide)

▶ DD_HASHSIZE configuration parameter (Administrator's Reference)

▶ DS_POOLSIZE configuration parameter (Administrator's Reference)

▶ PC_HASHSIZE configuration parameter (Administrator's Reference)

▶ PC_POOLSIZE configuration parameter (Administrator's Reference)

▶ SHMADD configuration parameter (Administrator's Reference)

▶ SHMNOACCESS configuration parameter (Administrator's Reference)

▶ SHMTOTAL configuration parameter (Administrator's Reference)

▶ STACKSIZE configuration parameter (Administrator's Reference)

▶ EXTSHMADD configuration parameter (Administrator's Reference)

▶ SHMVIRTSIZE configuration parameter (Administrator's Reference)

Set parameters for shared-memory performance

You can modify the configuration parameters that specify shared-memory information.

The following configuration parameters affect shared-memory performance.

AUTO_READAHEAD
 Specifies the automatic read-ahead mode or disables automatic read-ahead operations for a query. Automatic read-ahead operations help improve query performance by issuing asynchronous page requests when the database server detects that the query is encountering I/O. Asynchronous page requests can

improve query performance by overlapping query processing with the processing necessary to retrieve data from disk and put it in the bufferpool.

CKPTINTVL

Specifies the maximum number of seconds that can elapse before the database server checks if a checkpoint is required and the RTO_SERVER_RESTART configuration parameter is not set to turn on automatic checkpoint tuning.

CLEANERS

Specifies the number of page-cleaner threads that the database server is to run.

RA_PAGES

Specifies the number of disk pages that the database server attempts to read ahead during sequential scans of data or index records. If the AUTO_READAHEAD configuration parameter is enabled, the server ignores information specified in the RA_PAGES configuration parameter.

Related concepts:

Shared memory (Performance Guide)

Related reference:

AUTO_READAHEAD configuration parameter (Administrator's Reference)

CKPTINTVL configuration parameter (Administrator's Reference)

RA_THRESHOLD configuration parameter (Administrator's Reference)

CLEANERS configuration parameter (Administrator's Reference)

RA_PAGES configuration parameter (Administrator's Reference)

Set shared-memory parameters with a text editor

You can use a text editor to set the configuration parameters for resident and virtual shared memory, and shared-memory performance. Locate the parameter in the `onconfig` file, enter the new value or values, and rewrite the file to disk. Before the changes take effect, however, you must shut down and restart the database server.

Set SQL statement cache parameters

The following table shows the different ways that you can configure the SQL statement cache.

Table 7-1. Configure the SQL statement cache

Configuration parameter	Purpose	The onmode command
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL statement cache can hold a parsed and optimized SQL statement.	**onmode -e** *mode*
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.	**onmode -W STMT_CACHE_HITS**
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.	**onmode -W STMT_CACHE_NOLIMIT**
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.	None
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.	None

Use the following **onstat** options to monitor the SQL statement cache:

- **onstat -g ssc** (same as **onstat -g cac stmt**)
- **onstat -g ssc all**
- **onstat -g ssc pool**

For more information about these configuration parameters, **onstat -g** options, and **onmode** commands, see the *IBM Informix Administrator's Reference*.

For more information about using the SQL statement cache, monitoring it with the **onstat -g** options, and tuning the configuration parameters, see improving query performance in the *IBM Informix Performance Guide*. For details on qualifying and identical statements, see the *IBM Informix Guide to SQL: Syntax*.

Set up shared memory

To set up shared memory, take the database server offline and then online. For information about how to take the database server from online mode to offline, see "Change from any mode immediately to offline mode" on page 3-14.

Turn residency on or off for resident shared memory

You can turn residency on or off for the resident portion of shared memory in either of the following two ways:

- Use the **onmode** utility to reverse the state of shared-memory residency immediately while the database server is in online mode.
- Change the RESIDENT parameter in the `onconfig` file to turn shared-memory residency on or off for the next time that you set up the database server shared memory.

For a description of the resident portion of shared memory, see "Resident portion of shared memory" on page 6-8.

Turn residency on or off in online mode

To turn residency on or off while the database server is in online mode, use the **onmode** utility.

To turn on residency immediately for the resident portion of shared memory, run the following command:**% onmode -r**

To turn off residency immediately for the resident portion of shared memory, run the following command: **% onmode -n**

These commands do not change the value of the RESIDENT parameter in the `onconfig` file. That is, this change is not permanent, and residency reverts to the state specified by the RESIDENT parameter the next time that you set up shared memory. On UNIX, you must be **root** or user **informix** to turn residency on or off. On Windows, you must be a user in the **Informix Admin** group to turn residency on or off.

Turn residency on or off when restarting the database server

You can use a text editor to turn residency on or off. To change the current state of residency, use a text editor to locate the RESIDENT parameter. Set RESIDENT to 1

to turn residency on or to 0 to turn residency off, and rewrite the file to disk. Before the changes take effect, you must shut down and restart the database server.

Add a segment to the virtual portion of shared memory

You can use the **-a** option of the **onmode** utility to add a segment of specified size to virtual shared memory.

You are not normally required to add segments to virtual shared memory because the database server automatically adds segments as necessary.

The option to add a segment with the **onmode** utility is useful if the number of operating-system segments is limited, and the initial segment size is so low, relative to the amount that is required, that the operating-system limit of shared-memory segments is nearly exceeded.

Related concepts:

"Size of the virtual portion of shared memory" on page 6-13

Reserve memory for critical activities

You can reserve a specific amount of memory for use when critical activities (such as rollback activities) are required and the database server has limited free memory. This prevents the database server from crashing if the server runs out of free memory during critical activities.

If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, critical activities, such as rollback activities, can complete even when a user is getting out of memory errors. If the value of LOW_MEMORY_RESERVE is 0, the low memory reserve functionality is turned off.

For example, 512 kilobytes is a reasonable amount of reserved memory. To reserve 512 kilobytes, specify:

```
LOW_MEMORY_RESERVE 512
```

You can also use the **onmode -wm** or **onmode -wf** command to dynamically adjust the value of the LOW_MEMORY_RESERVE configuration parameter.

Use the **onstat -g seg** command to monitor the LOW_MEMORY_RESERVE value. Look for the last two lines of output, which contain the phrase "low memory reserve." The first of these output lines shows the size of memory reserved in bytes. The second of these lines shows the number times that the database server has used this memory and the maximum memory required. Both of these values are reset when the server is restarted.

Related reference:

➭ LOW_MEMORY_RESERVE configuration parameter (Administrator's Reference)

➭ onstat -g seg command: Print shared memory segment statistics (Administrator's Reference)

➭ onmode -wf, -wm: Dynamically change certain configuration parameters (Administrator's Reference)

Configure the server response when memory is critically low

You can configure the actions that the server takes to continue processing when memory is critically low, instead of returning an out of memory error. You specify the criteria for terminating sessions based on idle time, memory usage, and other factors so that the targeted application can continue to process. Configuring the low memory response is useful for embedded applications that have memory limitations.

To set up automatic low memory management:

- Set the LOW_MEMORY_MGR configuration parameter to 1, which enables low memory management when the database server starts.
- Set the threshold parameters for the amount of memory to maintain by using an SQL administration API command with the `scheduler lmm enable` argument.

To disable automatic low memory management, run an SQL administration API command with the `scheduler lmm disable` argument.

Related reference:

➭ LOW_MEMORY_MGR configuration parameter (Administrator's Reference)

➭ scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API) (Administrator's Reference)

➭ scheduler lmm disable argument: Stop automatic low memory management (SQL administration API) (Administrator's Reference)

➭ onstat -g lmm command: Print low memory management information (Administrator's Reference)

Scenario for maintaining a targeted amount of memory

The scenario in this topic shows how you can maintain a targeted amount of memory in applications that have memory limitations.

Suppose you want to specify that when the database server has 10 MB or less of free memory, it starts running the low memory management processes that can stop applications and free memory. Suppose you also want to specify that the server stops running the low memory management processes when the server has 20 MB or more of free memory:

1. Set the LOW_MEMORY_MGR configuration parameter to 1 and restart the server, or run an **onmode -wf** command to change the value of the LOW_MEMORY_MGR configuration parameter.
2. Run an SQL administration API command with the `scheduler lmm enable` argument and low memory parameters, as follows:

```
EXECUTE FUNCTION task("scheduler lmm enable",
 "LMM START THRESHOLD", "10MB",
 "LMM STOP THRESHOLD", "20MB",
 "LMM IDLE TIME", "300");
```

3. Run the **onstat -g lmm** command to display information about automatic low memory management settings, including the amount of memory that the server is attempting to maintain, the amount of memory currently used by the server, the low memory start and stop thresholds, and other memory-related statistics.

   You can also view low memory management information in the online.log file.

Related reference:

➡️ LOW_MEMORY_MGR configuration parameter (Administrator's Reference)

➡️ scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API) (Administrator's Reference)

➡️ scheduler lmm disable argument: Stop automatic low memory management (SQL administration API) (Administrator's Reference)

➡️ onstat -g lmm command: Print low memory management information (Administrator's Reference)

Monitor shared memory

These topics describe how to monitor shared-memory segments, the shared-memory profile, and the use of specific shared-memory resources (buffers, latches, and locks).

You can use the **onstat -o** utility to capture a static snapshot of database server shared memory for later analysis and comparison.

Monitor shared-memory segments

Monitor the shared-memory segments to determine the number and size of the segments that the database server creates. The database server allocates shared-memory segments dynamically, so these numbers can change. If the database server is allocating too many shared-memory segments, you can increase the SHMVIRTSIZE configuration parameter. For more information, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

The **onstat -g seg** command lists information for each shared-memory segment, including the address and size of the segment, and the amount of memory that is free or in use. For an example of **onstat -g seg** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitor the shared-memory profile and latches

Monitor the database server profile to analyze performance and the use of shared-memory resources.

You can obtain statistics on latch use and information about specific latches. These statistics provide a measure of the system activity.

The ON-Monitor Profile screen maintains cumulative statistics on shared-memory use.

To reset these statistics to zero, use the **onstat -z** option. For a description of all the fields that **onstat** displays, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Command-line utilities to monitor shared memory and latches

You can use the following command-line utilities to monitor shared memory and latches:

onstat -s
> Use **onstat -s** command to obtain latch information.

onstat -p
> Run **onstat -p** to display statistics on database server activity and waiting latches (in the **lchwaits** field). For an example of **onstat -p** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitor the shared memory profile and latches with ON-Monitor (UNIX)

Select **Status** > **Profile**. The screen displays shared-memory statistics, and the current operating mode, the boot time, the current time, and latches.

SMI tables

Query the **sysprofile** table to obtain shared-memory statistics. This table contains all of the statistics available in **onstat -p** output except the **ovbuff**, **usercpu**, and **syscpu** statistics.

Monitor buffers

You can obtain both statistics on buffer use and information about specific buffers. The statistical information includes the percentage of data writes that are cached to buffers and the number of times that threads were required wait to obtain a buffer. The percentage of writes cached is an important measure of performance. (For information about how to use this statistic to tune the database server, see your *IBM Informix Performance Guide*.)

The number of waits for buffers gives a measure of system concurrency.

information about specific buffers includes a listing of all the buffers in shared memory that are held by a thread. You can use this information to track the status of a particular buffer. For example, you can determine if another thread is waiting for the buffer.

Command-line utilities to monitor buffers

You can use the following command-line utilities to monitor buffers:

The onstat -p utility:
Run **onstat -p** to obtain statistics about cached reads and writes. The following caching statistics are displayed in four fields on the top row of the output display:
- The number of reads from shared-memory buffers (**bufreads**)
- The percentage of reads cached (**%cached**)
- The number of writes to shared memory (**bufwrits**)
- The percentage of writes cached (**%cached**)
- information about generic pages (nonstandard pages in the buffer pool)

In the output, the number of reads or writes can be a negative number if the number of occurrences exceeds 2^{32} (depends on the platform).

The **onstat -p** option also displays a statistic (**bufwaits**) that indicates the number of times that sessions were required wait for a buffer.

For an example of **onstat -p** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The onstat -B utility:
Run **onstat -B** to obtain information about all of the buffers that are not on the free-list, including:
- The shared memory address of the buffer
- The address of the thread that currently holds the buffer
- The address of the first thread that is waiting for each buffer
- Information about buffer pools

For an example of **onstat -B** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Related tasks:

The onstat -b utility:
Run **onstat -b** to obtain the following information about each buffer:
- Address of each buffer currently held by a thread
- Page numbers for the page held in the buffer
- Type of page held in the buffer (for example, data page, tblspace page, and so on)
- Type of lock placed on the buffer (exclusive or shared)
- Address of the thread that is currently holding the buffer
- Address of the first thread that is waiting for each buffer
- information about buffer pools

You can compare the addresses of the user threads to the addresses that are displayed in the **onstat -u** display to obtain the session ID number.

For more information about the fields that **onstat** displays, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*

The onstat -X utility:
Run **onstat -X** to obtain the same information as for **onstat -b**, along with the complete list of all threads that are waiting for buffers, not just the first waiting thread.

The onstat -R utility:
Use **onstat -R** to display information about buffer pools, including information about buffers.

Monitor buffers with ON-Monitor (UNIX)
You can use ON-Monitor to obtain statistics about cached reads and writes.

To access the fields mentioned in the topic "The onstat -p utility" on page 7-10 for **onstat -p** (**bufreads**, **%cached**, **bufwrits %cached**), select the **Status** > **Profile** option.

Here is an example of cached read and write statistics in the **Profile** option of the ON-Monitor **Status** menu:

...

```
Disk Reads  Buff. Reads   %Cached  Disk Writes  Buff. Writes   %Cached
      177          330    46.36             4             0      0.00
```

...

SMI tables

Query the **sysprofile** table to obtain statistics on cached reads and writes and total buffer waits. The following rows are relevant.

dskreads

> Number of reads from disk

bufreads

> Number of reads from buffers

dskwrites

> Number of writes to disk

bufwrites

> Number of writes to buffers

buffwts

> Number of times that any thread was required to wait for a buffer

Monitor buffer-pool activity

You can obtain statistics that relate to buffer availability and information about the buffers in each LRU queue.

The statistical information includes the number of times that the database server attempted to exceed the maximum number of buffers and the number of writes to disk (categorized by the event that caused the buffers to flush). These statistics help you determine if the number of buffers is appropriate. For information about tuning database server buffers, see your *IBM Informix Performance Guide*.

information about the buffers in each LRU queue consists of the length of the queue and the percentage of the buffers in the queue that have been modified.

Command-line utilities to obtain information about buffer-pool activity

You can use the **onstat** utility to obtain information about buffer-pool activity. You also can run **onstat** options from IBM Informix Server Administrator.

For more information about the **onstat** options, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The onstat -p utility:
The **onstat -p** output contains a statistic (**ovbuff**) that indicates the number of times the database server attempted to exceed the maximum number of shared buffers specified by `buffers` value in the BUFFERPOOL configuration parameter.

The onstat -F utility:
Run **onstat-F** to obtain a count by write type of the writes performed. (For an explanation of the different write types, see "Describing flushing activity" on page 6-27.

The **onstat-F** command displays totals for the following write types:

- Foreground write
- LRU write
- Chunk write

The **onstat-F** command also lists the following information about the page cleaners:
- Page-cleaner number
- Page-cleaner shared-memory address
- Current state of the page cleaner
- LRU queue to which the page cleaner was assigned

For an example of **onstat -F** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The onstat -R utility:
Run **onstat -R** to obtain information about the number of buffers in each LRU queue and the number and percentage of the buffers that are modified or free.

For an example of **onstat -R** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*

SMI tables
Query the **sysprofile** table to obtain the statistics on write types that are held in the following rows.

fgwrites
> Number of foreground writes

lruwrites
> Number of LRU writes

chunkwrites
> Number of chunk writes

Deleting shared memory segments after a server failure

You must close shared memory segments after a database server failure.

Important: This procedure must be performed by a DBA with experience using IBM Informix. Consult technical support for assistance. This procedure is for UNIX systems only.

In the event of a failure of an Informix database server instance, follow this procedure to delete shared memory segments:
1. Log on as user **informix**.
2. Use the **onmode -k** command to take the database server to offline mode and remove shared memory.
3. If the **onmode -k** command fails and the server is not offline, either run the **onclean -k** command, or perform the following steps:
   a. Use the **onstat -g glo** command to display multithreading information.
   b. In the output from the previous command, find the process ID (pid) associated with the first instance of **cpu** in the class column. For example, in the following output from the **onstat -g glo** command, there are four occurrences of **cpu** in the class column, having pids of 2599, 2603, 2604, and 2605:

```
MT global info:
sessions threads  vps      lngspins
0        49       14       1
         sched calls       thread switches yield 0   yield n   yield forever
total:   900100            898846          1238      27763     423778
per sec: 327               325             2         12        151
Virtual processor summary:
 class       vps       usercpu   syscpu    total
 cpu         4         0.92      0.10      1.02
 aio         4         0.02      0.02      0.04
 lio         1         0.00      0.00      0.00
 pio         1         0.00      0.00      0.00
 adm         1         0.00      0.01      0.01
 msc         1         0.00      0.00      0.00
 fifo        2         0.00      0.00      0.00
 total       14        0.94      0.13      1.07
Individual virtual processors:
 vp    pid       class     usercpu   syscpu    total
 1     2599      cpu       0.25      0.06      0.31
 2     2602      adm       0.00      0.01      0.01
 3     2603      cpu       0.23      0.00      0.23
 4     2604      cpu       0.21      0.03      0.24
 5     2605      cpu       0.23      0.01      0.24
 6     2606      lio       0.00      0.00      0.00
 7     2607      pio       0.00      0.00      0.00
 8     2608      aio       0.02      0.02      0.04
 9     2609      msc       0.00      0.00      0.00
 10    2610      fifo      0.00      0.00      0.00
 11    2611      fifo      0.00      0.00      0.00
 12    2612      aio       0.00      0.00      0.00
 13    2613      aio       0.00      0.00      0.00
 14    2614      aio       0.00      0.00      0.00
                 tot       0.94      0.13      1.07
```

   c. Use the **kill** command to terminate (in order) process IDs 2599, 2603, 2604, and 2605.

4. If the shared segments have not been removed then follow these steps:

   a. Determine the server number. The server number can be found by examining the `onconfig` file of the Informix instance

   b. Add the server number to 21078. For example, if the server number is 1, then add 1 to 21078, giving 21079.

   c. Convert the sum from the previous step to hexadecimal. In the previous example, 21079 is 5257 hexadecimal.

   d. Concatenate 48 to the hex value from the previous step. For example, 525748.

   e. Run the **ipcs** utility as root to display the shared memory segments, if any, left open by the server. Search the key column for the number from 4d.

   f. Remove each shared memory ID associated with the number from 4d.

For more information about the **onclean** utility, see the *IBM Informix Administrator's Reference*.

Consult your operating system documentation for the correct **ipcm** syntax for your system.

Chapter 8. Data storage

These topics define terms and explain the concepts that you must understand to perform the tasks described in Chapter 9, "Manage disk space," on page 9-1. These topics cover the following areas:

- Definitions of the physical and logical units that the database server uses to store data on disk
- Instructions on how to calculate the amount of disk space that you require to store your data
- Guidelines on how to lay out your disk space and where to place your databases and tables
- Instructions on using external tables

See the current IBM Informix release notes for any supplementary information about the maximum values related to the storage units explained in these topics.

Physical and logical units of storage

The database server uses the physical units of storage to allocate disk space. Unlike the logical units of storage whose size fluctuates, each of the physical units has a fixed or assigned size that is determined by the disk architecture. The database server uses the following physical units to manage disk space:

- Chunk
- Page
- Extent
- Blobpage
- Sbpage

The database server stores data in the following logical units:

- Dbspace
- Temporary dbspace
- Blobspace
- Sbspace
- Temporary sbspace
- Extspace
- Database
- Table
- Tblspace
- Partition

The database server maintains the following storage structures to ensure physical and logical consistency of data:

- Logical log
- Physical log
- Reserved pages

The following topics describe the various data-storage units that the database server supports and the relationships between those units. For information about reserved pages, see the disk structures and storage topics in the *IBM Informix Administrator's Reference*.

Chunks

A *chunk* is the largest unit of physical disk dedicated to database server data storage.

Chunks provide administrators with a significantly large unit for allocating disk space. The maximum size of an individual chunk is 4 TB. The number of allowable chunks is 32,766. If you have upgraded from a version before version 10.00, you must run the **onmode -BC 2** command to enable the maximum size of a chunk and the maximum number allowable, otherwise, the maximum chunk size is 2 GB.

The following storage spaces are comprised of chunks:
- Dbspaces
- Blobspaces
- Sbspaces
- Temporary dbspaces
- Temporary sbspaces

When you create a chunk, you specify its path, size, and the associated storage space name.

The database server also uses chunks for mirroring. When you mirror a chunk, the database server maintains two copies of the data on the chunk. Every write operation to a primary chunk is automatically followed by an identical write operation to the mirror chunk. Read operations are evenly divided between the two chunks. If either the primary chunk or the mirror chunk fails, the chunk that failed is marked as down, and the other chunk performs all operations without interrupting the user access to data.

When you create tables, indexes, and other database objects, chunk space is allocated, or assigned, to those objects. Space that is allocated is not necessarily used. For example, when you create a table, you allocate space for it, but that space is not used until you add data to the table. When all the chunks in a dbspace report 0 free pages, you cannot create new database objects in that dbspace. However, you can continue to add data to existing database objects as long as they have unused space. You can monitor chunks by using the **onstat -d** command or the OpenAdmin Tool (OAT) for Informix.

Related concepts:

"Sbspaces" on page 8-13

"Blobspaces" on page 8-12

"Dbspaces" on page 8-9

Chapter 17, "Mirroring," on page 17-1

Related reference:

"Specify names for storage spaces and chunks" on page 9-5

⯈ onstat -d command: Print chunk information (Administrator's Reference)

⯈ onmode -BC: Allow large chunk mode (Administrator's Reference)

Disk allocation for chunks

The database server can use regular operating-system files or *raw disk devices* to store data. On UNIX, you must use raw disk devices to store data whenever performance is important. On Windows, using NTFS files to store data is recommended for ease of administration.

An IBM Informix storage space can be on an NFS-mounted file system using regular operating-system files.

Disk access on Windows

On Windows, both raw disks and NTFS use kernel asynchronous I/O (KAIO). The Windows file system manager adds additional overhead to disk I/O, so using raw disks provides slight performance advantages. Because NTFS files are a more standard method of storing data, you must use NTFS files instead of raw disks. Consider using raw disks if your database server requires a large amount of disk access.

Raw disk space on Windows: On Windows, *raw disk space* can be either a physical drive without a drive letter or a logical disk partition that has been assigned a drive letter using the **Disk Administrator**. The space can either be formatted or unformatted. If it contains data, the data is overwritten after the space has been allocated to the database server. For more information, see "Allocating raw disk space on Windows" on page 9-4.

NTFS files: You must use NTFS files, not FAT files, for disk space on Windows. For more information, see "Allocating NTFS file space on Windows" on page 9-4.

Unbuffered or buffered disk access on UNIX

You can allocate disk space in two ways. You can either use files that are buffered through the operating system, or you can use unbuffered disk access.

Files that are buffered through the operating system are often called *cooked* files.

Unbuffered disk access is also called *raw* disk space.

When dbspaces are located on *raw disk devices* (also called *character-special devices*), the database server uses unbuffered disk access.

To create a raw device, configure a *block device* (hard disk) with a raw interface. The storage space that the device provides is called *raw disk space*. A chunk of raw disk space is physically contiguous.

The name of the chunk is the name of the character-special file in the /dev directory. In many operating systems, you can distinguish the character-special file from the block-special file by the first letter in the file name (typically r). For example, /dev/rsd0f is the character-special device that corresponds to the /dev/sd0f block-special device.

For more information, see "Allocating raw disk space on UNIX" on page 9-3.

A *cooked file* is a regular file that the operating system manages. Cooked file chunks and raw disk chunks are equally reliable. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

You can more easily allocate cooked files than raw disk space. To allocate a cooked file, you must create the file on any existing partition. The name of the chunk is the complete path name of the file. These steps are described in "Allocating cooked file spaces on UNIX" on page 9-3.

In a learning environment, where performance is not critical, or for static data, cooked files can be convenient. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system with minimal activity.

For cooked file chunks, the operating system processes all chunk *I/O* from its own buffer pool and ensures that all writes to chunks are physically written to the disk.

Important: While you must generally use raw disk devices on UNIX to achieve better performance, if you enable the DIRECT_IO configuration parameter, the performance for cooked files can approach the performance of raw devices used for dbspace chunks. This occurs because direct I/O bypasses the use of the file system buffers. If you have an AIX® operating system, you can also enable concurrent I/O for IBM Informix to use with direct IO when reading and writing to chunks that use cooked files. For more information about using direct IO or concurrent IO, see the *IBM Informix Performance Guide*.

To determine the best device for performance, perform benchmark testing on the system with both types of devices for the dbspace and table layout.

When using raw disks, you are not required to take any special action to create chunks and files that are larger than two gigabytes. If you want to create large chunks in cooked files, or if you want to use the various database export and import utilities with large files, you must ensure that the files systems that hold the large files are appropriately configured.

Extendable chunks

Extendable chunks are chunks that Informix can automatically extend or you can manually extend when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space will run out of space and when it will run out of space.

Configuring Informix to automatically add more storage space prevents the error that can occur if a partition requires additional storage space and cannot find that space in one of the chunks in the space in which the partition is located.

An extendable chunk must be in a nonmirrored dbspace or temporary dbspace.

You use an SQL administration API command with the `modify space sp_sizes` argument to modify the extend size and the create size for the space in which your extendable chunk is located.

Related concepts:

"Automatic space management" on page 9-26

"The storage pool" on page 8-34

Related tasks:

"Marking a chunk as extendable or not extendable" on page 9-29

"Manually expanding a space or extending an extendable chunk" on page 9-31

Offsets

The system administrator might divide a physical disk into *partitions*, which are different parts of a disk that have separate path names. Although you must use an entire disk partition when you allocate a chunk on a raw disk device, you can subdivide partitions or cooked files into smaller chunks using *offsets*. For more information, see "Disk-layout guidelines" on page 8-35.

Tip: With a 4-terabyte limit to the size of a chunk, you can avoid partitioning a disk by assigning a single chunk per disk drive.

You can use an offset to indicate the location of a given chunk on the disk partition, file, or device. For example, suppose that you create a 1000 KB chunk that you want to divide into two chunks of 500 KB each. You can use an offset of 0 KB to mark the beginning of the first chunk and an offset of 500 KB to mark the beginning of the second chunk.

You can specify an offset whenever you create, add, or drop a chunk from a dbspace, blobspace, or sbspace.

You might also be required to specify an offset to prevent the database server from overwriting partition information. "Allocating raw disk space on UNIX" on page 9-3 explains when and how to specify an offset.

Pages

A *page* is the physical unit of disk storage that the database server uses to read from and write to IBM Informix databases. The following figure illustrates the concept of a page, represented by a darkened sector of a disk platter.
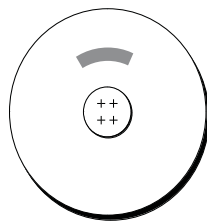
Figure 8-1. A page on disk

On most UNIX platforms, the page size is 2 KB. On Windows, the page size is 4 KB. Because your hardware determines the size of your page, you cannot alter this value.

A chunk contains a certain number of pages, as the following figure illustrates. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.
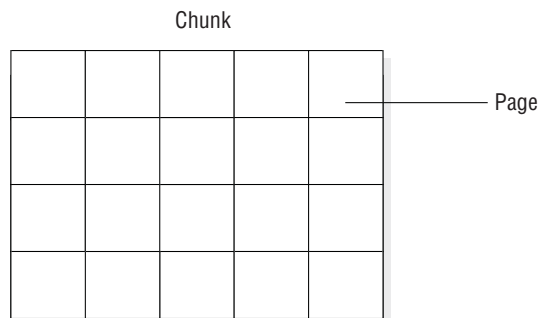
Chunk

Figure 8-2. A chunk, logically separated into a series of pages

For information about how the database server structures data within a page, see the chapter on disk structures and storage in the *IBM Informix Administrator's Reference*

Blobpages

A *blobpage* is the unit of disk-space allocation that the database server uses to store simple large objects (TEXT or BYTE data) within a blobspace. For a description of blobspaces, see "Blobspaces" on page 8-12.

You specify blobpage size as a multiple of the database server page size. Because the database server allocates blobpages as contiguous spaces, it is more efficient to store simple large objects in blobpages that are as close to the size of the data as possible. The following figure illustrates the concept of a blobpage, represented as a multiple (three) of a data page.
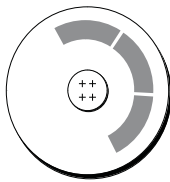
Figure 8-3. A blobpage on disk

For information about how IBM Informix structures data stored in a blobpage, see structure of a blobspace blobpage in the disk structures and storage topics of the *IBM Informix Administrator's Reference*.

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as the following figure illustrates. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.
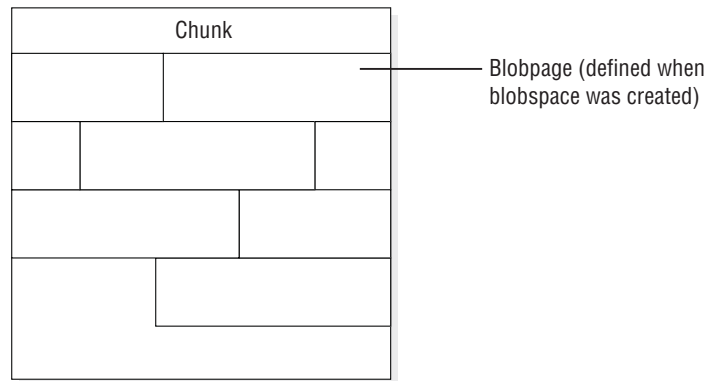
Figure 8-4. A chunk in a blobspace, logically separated into a series of blobpages

Instead of storing simple-large-object data in a blobspace, you can choose to store it in a dbspace. However, for a simple large object larger than two pages, performance improves when you store it in a blobspace. Simple large objects stored in a dbspace can share a page, but simple large objects stored in a blobspace do not share pages.

For information about how to determine the size of a blobpage, see "Determine blobpage size" on page 9-22.

Sbpages

An *sbpage* is the type of page that the database server uses to store smart large objects within an sbspace. For a description of sbspaces, see "Sbspaces" on page 8-13. Unlike blobpages, sbpages are not configurable. An sbpage is the same size as the database server page, which is usually 2 KB on UNIX and 4 KB on Windows.

The unit of allocation in an sbspace is an extent, whereas the unit of allocation in a blobspace is a blobpage. Just as with pages in a chunk, a certain number of smart large object extents compose a chunk in an sbspace, as the following figure illustrates. An extent is always entirely contained in a chunk and cannot cross chunk boundaries.
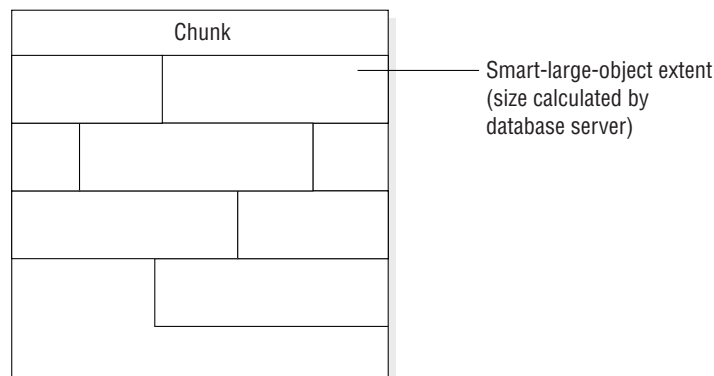
Figure 8-5. A chunk in an sbspace, logically separated into a series of extents

Smart large objects cannot be stored in a dbspace or blobspace. For more information, see "Sbspaces" on page 8-13, and sbspace structure in the disk structures and storage chapter of the *IBM Informix Administrator's Reference*.

The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For more information, see "Extent sizes for sbspaces" on page 8-16.

Extents

When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. When this space fills, the database server must allocate space for additional storage. The physical unit of storage that the database server uses to allocate both the initial and subsequent storage space is called an *extent*.

The following figure illustrates the concept of an extent.
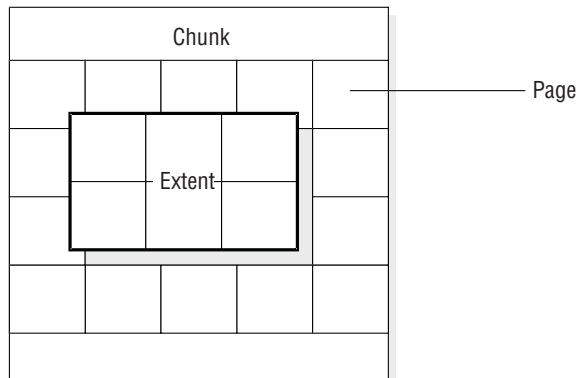
Figure 8-6. An extent that consists of six contiguous pages on a raw disk device

An extent consists of a collection of contiguous pages that store data for a given table. (See "Tables" on page 8-22.) Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of KB allocated to the table when it is first created. The *next-extent* size is the number of KB allocated to the table when the initial extent (and any subsequent extents) becomes full. For permanent tables and user-defined temporary tables, the next-extent size begins to double after each extent. For system-created temporary tables, the next-extent size begins to double after 4 extents have been added.

When you create a table, you can specify the size of the initial extent, and the size of the extents to be added as the table grows. You can also modify the size of an extent in a table in a dbspace, and you can modify the size of new subsequent extents. To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the *IBM Informix Guide to SQL: Syntax* and disk structures in the *IBM Informix Administrator's Reference*.

When you create a table with a column for CLOB or BLOB data types, you also define extents for an sbspace. For more information, see "Storage characteristics of sbspaces" on page 8-16.

The following figure shows how the database server allocates six pages for an extent:
- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.

- If the database server cannot find the contiguous disk space that is specified for the next-extent size, it searches the next chunk in the dbspace for contiguous space.

Chunk 1	Chunk 2

Extent

The database server decides to allocate an extent and begins a search for 6 contiguous free pages. In this illustration, free pages are clear (white) and used pages are shaded. → The database server cannot find 6 contiguous free pages in chunk 1. → The database server extends its search to the next chunk. → The database server finds 6 contiguous free pages and allocates an extent.

Figure 8-7. Process of extent allocation

Related reference:

➥ Extent size doubling (Administrator's Reference)

Dbspaces

A *dbspace* is a logical unit that can contain between 1 and 32,766 chunks. Place databases, tables, logical-log files, and the physical log in dbspaces.

Related concepts:

"Chunks" on page 8-2

Control of where simple large object data is stored

A key responsibility of the database server administrator is to control where the database server stores data. By storing high-use access tables or *critical dbspaces* (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on separate physical devices, you ensure that when one of the disks holding noncritical data fails, the failure affects only the availability of data on that disk.

As the following figure shows, to control the placement of databases or tables, you can use the IN *dbspace* option of the CREATE DATABASE or CREATE TABLE statements. (For more information, see "Tables" on page 8-22.)

% onspaces -c -d stores_space -p /dev/rsd0f -o 0 -s 10000

CREATE TABLE stores_demo IN stores_space

Dbspace on UNIX

/dev/rsd0f
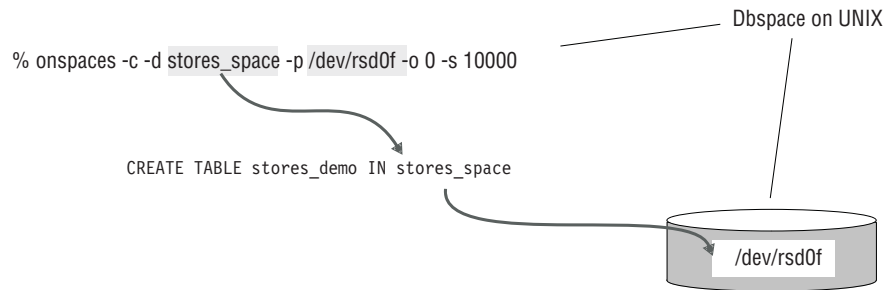
Figure 8-8. Control table placement with the CREATE TABLE... IN statement

Before you create a database or table in a dbspace, you must first create the dbspace. For more information about how to create a dbspace, see "Creating a dbspace that uses the default page size" on page 9-7.

A dbspace includes one or more chunks, as the following figure shows. You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor dbspace chunks for fullness and to anticipate the necessity to allocate more chunks to a dbspace. (See "Monitor disk usage" on page 9-44.) When a dbspace contains more than one chunk, you cannot specify the chunk in which the data is located.

Logical units of storage

Database

System catalog

Table 1

Table 2

Dbspace 1

Dbspace 2

Dbspace 3

Physical units of storage

Chunks

Chunk 1

Chunk 2

Chunk 3

Chunk 4
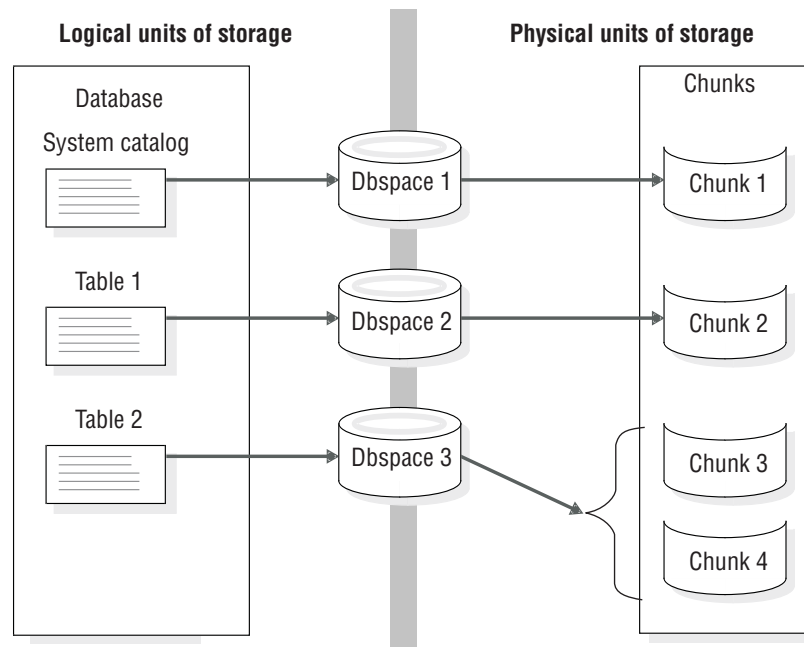
Figure 8-9. Dbspaces that link logical and physical units of storage

The database server uses the dbspace to store databases and tables. (See "Tables" on page 8-22.)

When you create a standard or temporary dbspace, you can specify the page size for the dbspace. You cannot specify a page size for blobspaces, sbspaces, or external spaces. If you do not specify a page size, the size of the root dbspace is the default page size. For more information, see "Creating a dbspace with a non-default page size" on page 9-10.

When you create a standard dbspace, you can specify the first and next extent sizes for the tblspace **tblspace** in the dbspace. Do this if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you must place the tblspace **tblspace** extents in non-primary chunks. For more information, see "Specifying the first and next extent sizes for the tblspace **tblspace**" on page 9-8.

You can mirror every chunk in a mirrored dbspace. As soon as the database server allocates a mirror chunk, it flags all space in that mirror chunk as full. See "Monitor disk usage" on page 9-44.

For information about using IBM Informix Server Administrator or **onspaces** to perform the following tasks, see Chapter 9, "Manage disk space," on page 9-1.
- Creating a dbspace
- Adding a chunk to a dbspace
- Renaming a dbspace
- Dropping a chunk
- Dropping a dbspace, blobspace, or sbspace

Root dbspace

The *root dbspace* is the initial dbspace that the database server creates. The root dbspace is special because it contains reserved pages and internal tables that describe and track all physical and logical units of storage. (For more information about these topics, see "Tables" on page 8-22 and the disk structures and storage chapter in the *IBM Informix Administrator's Reference*.) The initial chunk of the root dbspace and its mirror are the only chunks created during disk-space setup. You can add other chunks to the root dbspace after disk-space setup.

The following disk-configuration parameters in the `onconfig` configuration file refer to the first (initial) chunk of the root dbspace:
- ROOTPATH
- ROOTOFFSET
- ROOTNAME
- MIRRORPATH
- MIRROROFFSET
- TBLTBLFIRST
- TBLTBLNEXT

The root dbspace is also the default dbspace location for any database created with the CREATE DATABASE statement.

The root dbspace is the default location for all temporary tables created by the database server to perform requested data management.

See "Size of the root dbspace" on page 8-32 for information about how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you set up database server disk space.

Temporary dbspaces

A *temporary dbspace* is a dbspace reserved exclusively for the storage of temporary tables. You cannot mirror a temporary dbspace.

The database server never drops a temporary dbspace unless it is explicitly directed to do so. A temporary dbspace is temporary only in the sense that the database server does not preserve any of the dbspace contents when the database server shuts down abnormally.

Whenever you set up the database server, all temporary dbspaces are set up. The database server clears any tables that might remain since the last time that the database server shut down.

The database server does not perform logical or physical logging for temporary dbspaces. Because temporary dbspaces are not physically logged, fewer checkpoints and I/O operations occur, which improves performance.

The database server logs table creation, the allocation of extents, and the dropping of the table for a temporary table in a standard dbspace. In contrast, the database server does not log tables stored in temporary dbspaces. Logical-log suppression in temporary dbspaces reduces the number of log records to roll forward during logical recovery as well, thus improving the performance during critical downtime.

Using temporary dbspaces to store temporary tables also reduces the size of your storage-space backup, because the database server does not back up temporary dbspaces.

The database server uses temporary disk space to store the before images of data that are overwritten while backups are occurring and overflow from query processing that occurs in memory. Make sure that you have correctly set the DBSPACETEMP environment variable or parameter to specify dbspaces with enough space for your needs. If there is not enough room in the specified dbspaces, the backup fails, root dbspace is used, or the backup fails after filling the root dbspace.

If you have more than one temporary dbspace and execute a SELECT statement into a temporary table, the results of the query are inserted in round robin order.

For detailed instructions on how to create a temporary dbspace, see "Creating a temporary dbspace" on page 9-17.

Important: When the database server is running as a secondary database server, it requires a temporary dbspace to store any internal temporary tables generated by read-only queries.

Blobspaces

A *blobspace* is a logical storage unit composed of one or more chunks that store only TEXT and BYTE data. A blobspace stores TEXT and BYTE data in the most efficient way possible. You can store TEXT and BYTE columns associated with distinct tables (see "Tables" on page 8-22) in the same blobspace.

The database server writes data stored in a blobspace directly to disk. This data does not pass through resident shared memory. If it did, the volume of data might occupy so many of the buffer-pool pages that other data and index pages would be forced out. For the same reason, the database server does not write TEXT or BYTE objects that are assigned to a blobspace to either the logical or physical log. The database server logs blobspace objects by writing them directly from disk to the logical-log backup tapes when you back up the logical logs. Blobspace objects never pass through the logical-log files.

When you create a blobspace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of a database server administrator is to monitor the chunks for fullness and anticipate the necessity to allocate more chunks to a blobspace. For instructions on how to monitor chunks for fullness, see "Monitor simple large objects in a blobspace" on page 9-48. For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see Chapter 9, "Manage disk space," on page 9-1.

For information about the structure of a blobspace, see the topics about disk structures and storage in the *IBM Informix Administrator's Reference*.

Related concepts:

"Chunks" on page 8-2

Sbspaces

An *sbspace* is a logical storage unit composed of one or more chunks that store *smart large objects*. Smart large objects consist of CLOB (character large object) and BLOB (binary large object) data types. User-defined data types can also use sbspaces. For more information about data types, see the *IBM Informix Guide to SQL: Reference*.

Related concepts:

"Chunks" on page 8-2

Advantages of using sbspaces

Sbspaces have the following advantages over blobspaces:

- They have read, write, and seek properties similar to a standard UNIX file.

  Programmers can use functions similar to UNIX and Windows functions to read, write, and seek smart large objects. IBM Informix provides this smart-large-object interface in the DataBlade API and the Informix ESQL/C programming interface.

- They are recoverable.

  You can log all write operations on data stored in sbspaces. You can commit or roll back changes if a failure occurs during a transaction.

- They obey transaction isolation modes.

  You can lock smart large objects at different levels of granularity, and the lock durations obey the rules for transaction isolation levels. For more information about locking and concurrency, see your *IBM Informix Performance Guide*.

- Smart large objects within table rows are not required to be retrieved in one statement.

  An application can store or retrieve smart large objects in pieces using either the DataBlade API or the Informix ESQL/C programming interface. For more information about the DataBlade API functions, see the *IBM Informix DataBlade API Function Reference*. For more information about the Informix ESQL/C functions, see the *IBM Informix ESQL/C Programmer's Manual*.

Sbspaces and Enterprise Replication

Before you define a replication server for Enterprise Replication, you must create an sbspace. Enterprise Replication spools the replicated data to smart large objects. Specify the sbspace name in the CDR_QDATA_SBSPACE configuration parameter. Enterprise Replication uses the default log mode with which the sbspace was created for spooling the row data. The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise

Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. For more information, see the *IBM Informix Enterprise Replication Guide*.

Metadata, user data, and reserved area

As with blobspaces and dbspaces, when you create an sbspace, you assign to it one or more chunks. However, the first chunk of an sbspace always has three areas:

Metadata area

Metadata identifies key aspects of the sbspace and each smart large object stored in the sbspace, and enables the database server to manipulate and recover smart large objects stored within.

User-data area

User data is the smart large object data stored in the sbspace by user applications. The chunk has up to two user-data areas.

Reserved area

The database server allocates space from the reserved area to either the metadata or user-data area when more space is required. The chunk has up to two reserved areas.

For information about correctly allocating metadata and user data for sbspaces, see "Size sbspace metadata" on page 9-24 and the *IBM Informix Performance Guide*.

When you add a chunk to an sbspace, you can specify whether it contains a metadata area and user-data area or whether to reserve the chunk exclusively for user data. You can add more chunks at any time. If you are updating smart large objects, I/O to the user data is much faster on raw disks than cooked chunk files. For instructions on how to create an sbspace, add chunks to an sbspace, or drop a chunk from an sbspace, see Chapter 9, "Manage disk space," on page 9-1.

Important: Sbspace metadata is always logged, regardless of the logging setting of the database.

Control of where smart large object data is stored

You specify the data type of a column when you create the table. For smart large objects, you specify CLOB, BLOB, or user-defined data types. As the following figure shows, to control the placement of smart large objects, you can use the IN *sbspace* option in the PUT clause of the CREATE TABLE statement.

```
% onspaces -c -S s9_sbspc -p./s9_sbspc -o 0 -s 2000          sbspace on UNIX


        CREATE TABLE catalog
        (...advert_descr CLOB, ...)
               ...PUT advert_descr IN s9_sbspc

                                                                  s9_sbspc
```
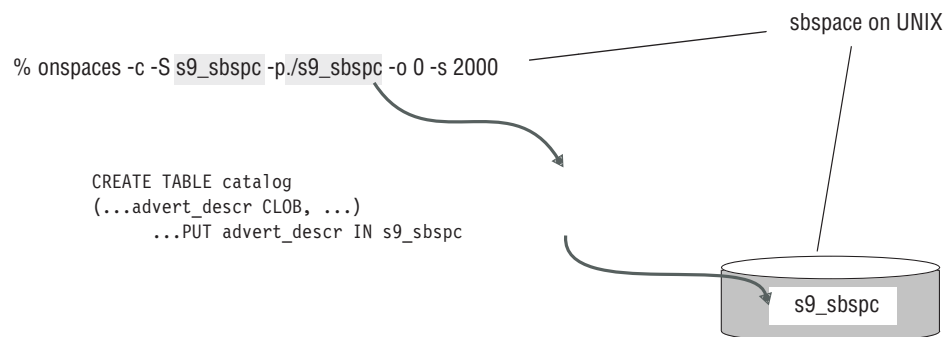
Figure 8-10. Control smart-large-object placement

Before you specify an sbspace in a PUT clause, you must first create the sbspace. For more information about how to create an sbspace with the **onspaces -c -S** command, see "Adding a chunk to a dbspace or blobspace" on page 9-18. For more information about how to specify smart large object characteristics in the PUT clause, see the CREATE TABLE statement in the *IBM Informix Guide to SQL: Syntax*.

If you do not specify the PUT clause, the database server stores the smart large objects in the default sbspace that you specify in the SBSPACENAME configuration parameter. For more information about SBSPACENAME, see the configuration parameter topics of the *IBM Informix Administrator's Reference*.

An sbspace includes one or more chunks, as the following figure shows. When an sbspace contains more than one chunk, you cannot specify the chunk in which the data is located.

You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor sbspace chunks for fullness and to anticipate the necessity to allocate more chunks to an sbspace. For more information about monitoring sbspaces, see your *IBM Informix Performance Guide*.
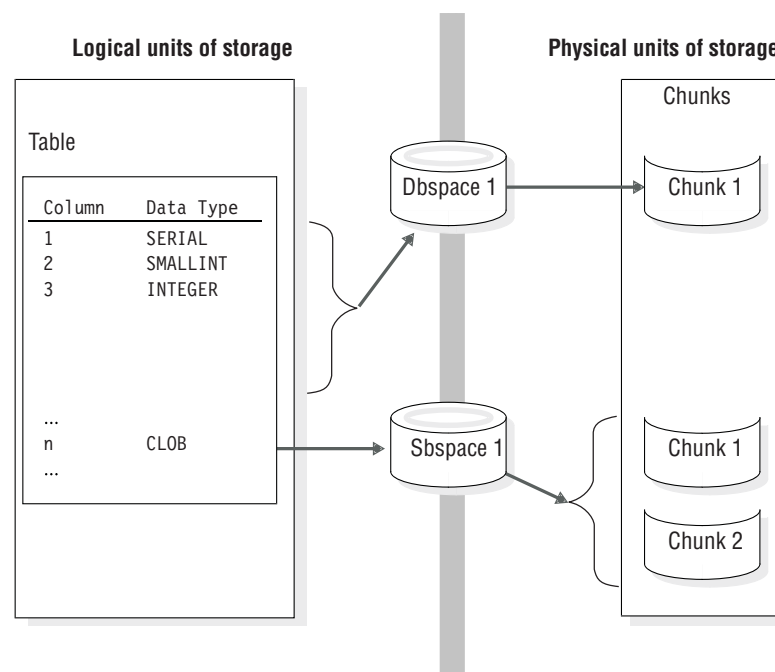
Figure 8-11. Sbspaces that link logical and physical units of storage

The database server uses sbspaces to store table columns that contain smart large objects. The database server uses dbspaces to store the rest of the table columns.

You can mirror an sbspace to speed recovery in event of a media failure. For more information, see "Mirroring" on page 17-1.

For information about using **onspaces** to perform the following tasks, see Chapter 9, "Manage disk space," on page 9-1.
- Creating an sbspace
- Adding a chunk to an sbspace

- Altering storage characteristics of smart large objects
- Creating a temporary sbspace
- Dropping an sbspace

Storage characteristics of sbspaces

As the database server administrator, you can use the system default values for these storage characteristics, or you can specify them in the **-Df** tags when you create the sbspace with **onspaces -c**. Later on, you can change these sbspace characteristics with the **onspaces -ch** option. The administrator or programmer can override these default values for storage characteristics and attributes for individual tables.

Extent sizes for sbspaces

Similar to extents in a table, an extent in an sbspace consists of a collection of contiguous pages that store smart large object data.

The unit of allocation in an sbspace is an extent. The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For example, if an operation asks to write 30 KB, the database server tries to allocate an extent the size of 30 KB.

Important: For most applications, you must use the values that the database server calculates for the extent size.

If you know the size of the smart large object, you can use one of the following functions to set the extent size. The database server allocates the entire smart large object as one extent (if an extent of that size is available in the chunk):

- The DataBlade API **mi_lo_specset_estbytes()** function

  For more information about the DataBlade API functions for smart large objects, see the *IBM Informix DataBlade API Function Reference*.

- The Informix ESQL/C **ifx_lo_specset_estbytes** function

  For more information about the Informix ESQL/C functions for smart large objects, see the *IBM Informix ESQL/C Programmer's Manual*.

For information about tuning extent sizes, see smart large objects in the chapter on configuration effects on I/O utilization in your *IBM Informix Performance Guide*.

Average smart-large-object size

Smart large objects usually vary in length. You can provide an average size of your smart large objects to calculate space for an sbspace. You specify this average size with the AVG_LO_SIZE tag of the **onspaces -c -Df** option.

To specify the size and location of the metadata area, specify the **-Ms** and **-Mo** flags in the **onspaces** command. If you do not use the **-Ms** flag, the database server uses the value of AVG_LO_SIZE to estimate the amount of space to allocate for the metadata area. For more information, see "Size sbspace metadata" on page 9-24.

Buffering mode

When you create an sbspace, the default buffering mode is on, which means to use the buffer pool in the resident portion of shared memory.

As the database administrator, you can specify the buffering mode with the BUFFERING tag of the **onspaces -c -Df** option. The default is "buffering=ON",

which means to use the buffer pool. If you turn off buffering, the database server uses private buffers in the virtual portion of shared memory.

Important: In general, if read and write operations to the smart large objects are less than 8 KB, do not specify a buffering mode when you create the sbspace. If you are reading or writing short blocks of data, such as 2 KB or 4 KB, leave the default of "buffering=ON" to obtain better performance.

For information about when to use private buffers, see the section on light-weight I/O operations in the topics about configuration effects on I/O utilization in your *IBM Informix Performance Guide*.

Last-access time

When you create an sbspace, you can specify whether the database server must keep the last time that the smart large object was read or updated with the ACCESSTIME tag of the **onspaces -c -Df** option. The default is "ACCESSTIME=OFF". The database server keeps this last-access time in the metadata area.

For more information about how programmers use this last-access time, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Lock mode

When you create an sbspace, you can specify whether the database server locks the whole smart large object or a range of bytes within a smart large object with the LOCK_MODE tag of the **onspaces -c -Df** option. The default is "LOCK_MODE=BLOB", which means to lock the entire smart large object. For more information, see the locking chapter in your *IBM Informix Performance Guide*.

Logging

When you create an sbspace, you can specify whether to turn on logging for the smart large objects. The default is no logging. For more information, see "Log sbspaces and smart large objects" on page 13-7.

Important: When you use logging databases, turn logging on for the sbspaces. If a failure occurs that requires log recovery, you can keep the smart large objects consistent with the rest of the database.

You specify the logging status with the LOGGING tag of the **onspaces -c -Df** option. The default is "LOGGING=off". You can change the logging status with the **onspaces -c -Df** option. You can override this logging status with the PUT clause in the SQL statements CREATE TABLE or ALTER TABLE. For more information about these SQL statements, see the *IBM Informix Guide to SQL: Syntax*.

The programmer can override this logging status with functions that the DataBlade API and Informix ESQL/C provide. For more information about the DataBlade API functions for smart large objects, see the *IBM Informix DataBlade API Function Reference*. For more information about the Informix ESQL/C functions for smart large objects, see the *IBM Informix ESQL/C Programmer's Manual*.

When you turn on logging for an sbspace, the smart large objects pass through the resident portion of shared memory. Although applications can retrieve pieces of a smart large object, you still must consider the larger size of data that might pass through the buffer pool and logical-log buffers. For more information, see "Access smart large objects" on page 6-31.

Levels of inheritance for sbspace characteristics

The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. You can use the system default values for sbspace attributes, or override them for specific sbspaces, columns in a table, or smart large objects. The following figure shows the storage-characteristics hierarchy for a smart large object.
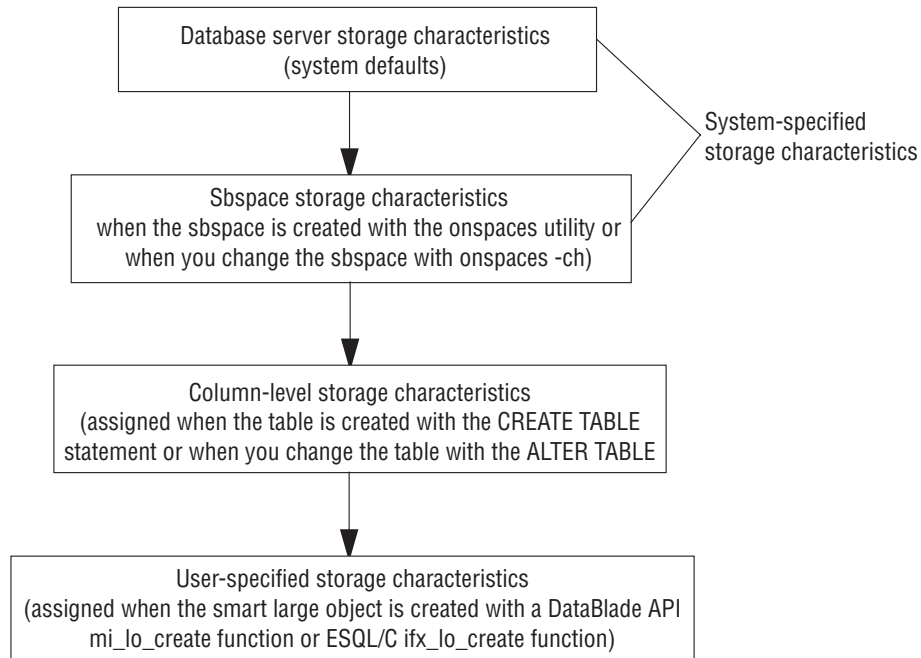
Figure 8-12. Storage-characteristics hierarchy

The figure shows that you can override the system default in the following ways:

- Use the **-Df** tags of the **onspaces -c -S** command to override the system default for a specific sbspace.

  You can later change these sbspace attributes for the sbspace with the **onspaces -ch** option. For more information about valid ranges for the **-Df** tags, see the **onspaces** topics in the *IBM Informix Administrator's Reference*.

- You override the system default for a specific column when you specify these attributes in the PUT clause of the CREATE TABLE or ALTER TABLE statements.

  For more information about these SQL statements, see the *IBM Informix Guide to SQL: Syntax*.

- The programmer can override the default values for sbspace attributes for specific smart large objects with functions that the DataBlade API and Informix ESQL/C programming interface provide.

More information about sbspaces

The following table lists sources of information about various tasks related to using and managing sbspaces.

Table 8-1. Finding information for sbspace tasks

Task	Reference
Setting memory configuration parameters for smart large objects	Chapter 7, "Manage shared memory," on page 7-1
Understanding sbpages	"Sbpages" on page 8-7
Specifying I/O characteristics for an sbspace	**onspaces** option in "Storage characteristics of sbspaces" on page 8-16
Allocating space for an sbspace	"Creating an sbspace" on page 9-23
Adding a chunk to an sbspace	"Adding a chunk to an sbspace" on page 9-24
Defining or altering storage characteristics for a smart large object	"Alter storage characteristics of smart large objects" on page 9-25 PUT clause of CREATE TABLE or ALTER TABLE statement in *IBM Informix Guide to SQL: Syntax*
Monitoring sbspaces	"Monitor sbspaces" on page 9-51 Topics about table performance considerations in *IBM Informix Performance Guide*
Setting up logging for an sbspace	"Log sbspaces and smart large objects" on page 13-7
Backing up an sbspace	"Back up sbspaces" on page 14-4
Checking consistency of an sbspace	"Validate metadata" on page 19-3
Understanding an sbspace structure	Topics about disk structures in the *IBM Informix Administrator's Reference*
Using **onspaces** for sbspaces	Topics about utilities in the *IBM Informix Administrator's Reference*
Creating a table with CLOB or BLOB data types	*IBM Informix Guide to SQL: Syntax*
Accessing smart large objects in an application	*IBM Informix DataBlade API Programmer's GuideIBM Informix ESQL/C Programmer's Manual*
Calculating the metadata area size Improving metadata I/O Changing storage characteristics	Topics about table performance in *IBM Informix Performance Guide*
Understanding smart-large-object locking	Topics about locking in *IBM Informix Performance Guide*
Configuring sbspaces for temporary smart large objects	Topics about configuration effects on I/O activity in *IBM Informix Performance Guide*

Temporary sbspaces

Use a *temporary sbspace* to store temporary smart large objects without metadata logging and user-data logging. If you store temporary smart large objects in a standard sbspace, the metadata is logged. Temporary sbspaces are similar to temporary dbspaces. To create a temporary sbspace, use the **onspaces -c -S** command with the **-t** option. For more information, see "Creating a temporary sbspace" on page 9-25.

You can store temporary large objects in a standard sbspace or temporary sbspace.
- If you specify a temporary sbspace in the SBSPACETEMP parameter, you can store temporary smart large objects there.

- If you specify a standard sbspace in the SBSPACENAME parameter, you can store temporary and permanent smart large objects there.
- If you specify a temporary sbspace name in the CREATE TEMP TABLE statement, you can store temporary smart large objects there.
- If you specify a permanent sbspace name in the CREATE TABLE statement, you can store temporary smart large objects there.
- If you omit the SBSPACETEMP and SBSPACENAME parameters and create a smart large object, error message -12053 might display.
- If you specify a temporary sbspace in the SBSPACENAME parameter, you cannot store a permanent smart large object in that sbspace. You can store temporary smart large objects in that sbspace.

Comparison of temporary and standard sbspaces

The following table compares standard and temporary sbspaces.

Table 8-2. Temporary and standard sbspaces

Characteristics	Standard sbspace	Temporary sbspace
Stores smart large objects	Yes	No
Stores temporary smart large objects	Yes	Yes
Logs metadata	Metadata is always logged	Metadata is not logged
Logs user data	User data is not logged for temporary smart large objects but is logged for permanent smart large objects if LOGGING=ON	User data is not logged Creation and deletion of space, and addition of chunks is logged
Fast recovery	Yes	No (the sbspace is emptied when the database server restarts) To set up shared memory without cleaning up temporary smart large objects, specify **oninit -p**. If you keep the temporary large objects, their state is indeterminate.
Backup and restore	Yes	No
Add and drop chunks	Yes	Yes
Configuration parameter	SBSPACENAME	SBSPACETEMP

Temporary smart large objects

Use *temporary smart large objects* to store text or image data (CLOB or BLOB) that do not require restoring from a backup or log replay in fast recovery. Temporary smart large objects last for the user session and are much faster to update than smart large objects.

You create a temporary smart large object in the same way as a permanent smart large object, except you set the LO_CREATE_TEMP flag in the **ifx_lo_specset_flags** or **mi_lo_specset_flags** function. Use **mi_lo_copy** or **ifx_lo_copy** to create a permanent smart large object from a temporary smart large object. For details on creating temporary smart large objects, see the *IBM Informix DataBlade API Programmer's Guide*.

Important: Store pointers to temporary large objects in temporary tables only. If you store them in standard tables and reboot the database server, it results in an error that says that the large object does not exist.

The following table compares standard and temporary smart large objects.

Table 8-3. Temporary and standard smart large objects

Characteristics	Smart large object	Temporary smart large object
Creation flags	LO_CREATE_LOG or LO_CREATE_NOLOG	LO_CREATE_TEMP
Rollback	Yes	No
Logging	Yes, if turned on	No
Duration	Permanent (until user deletes it)	Deleted at end of user session or transaction
Table type stored in	Permanent or temporary table	Temporary tables

Extspaces

An *extspace* is a logical name associated with an arbitrary string that signifies the location of external data. The resource that the extspace references depends on a user-defined access method for accessing its contents.

For example, a database user might require access to binary files encoded in a proprietary format. First, a developer creates an *access method*, which is a set of routines that access the data. These routines are responsible for all interaction between the database server and the external file. A DBA then adds an extspace that has the file as its target to the database. After the DBA creates a table in the extspace, the users can access the data in the proprietary files through SQL statements. To locate those files, use the extspace information.

An extspace is not required to be a file name. For example, it can be a network location. The routines that access the data can use information found in the string associated with the extspace in any manner.

For more information about user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*. For more information about creating functions and primary access methods, see the *IBM Informix Guide to SQL: Syntax*.

Databases

A database is a logical storage unit that contains tables and indexes. (See "Tables" on page 8-22.) Each database also contains a system catalog that tracks information about many of the elements in the database, including tables, indexes, SPL routines, and integrity constraints.

A database is located in the dbspace specified by the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database is located in the root dbspace. When you do specify a dbspace in the CREATE DATABASE statement, this dbspace is the location for the following tables:

- Database system catalog tables
- Any table that belongs to the database

The following figure shows the tables contained in the stores_demo database.

stores_demo database

| customer table | orders table | items table | stock table | catalog table |

| cust_calls table | call_type table | manufact table | state table |

system catalog

| systables table | sysviews table |

zip_ix index
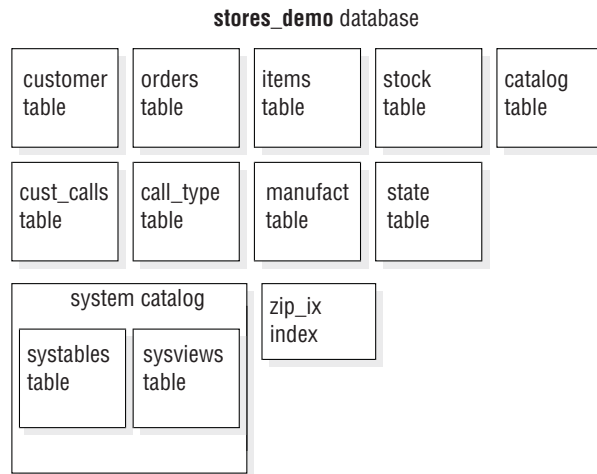
Figure 8-13. The stores_demo database

The size limits that apply to databases are related to their location in a dbspace. To be certain that all tables in a database are created on a specific physical device, assign only one chunk to the device, and create a dbspace that contains only that chunk. Place your database in that dbspace. When you place a database in a chunk assigned to a specific physical device, the database size is limited to the size of that chunk.

For instructions on how to list the databases that you create, see "Display databases" on page 9-43.

Tables

In relational database systems, a *table* is a row of column headings together with zero or more rows of data values. The row of column headings identifies one or more columns and a data type for each column.

When you create a table, the database server allocates disk space for the table in a block of pages called an extent. (See "Extents" on page 8-8.) You can specify the size of both the first and any subsequent extents.

You can place the table in a specific dbspace by naming the dbspace when the table is created (usually with the IN *dbspace* option of CREATE TABLE). When you do not specify the dbspace, the database server places the table in the dbspace where the database is located.

You can also:
- Fragment a table over more than one dbspace. However, you cannot put fragments into different page-size dbspaces. All fragments might be required to have the same page size.

  You must define a distribution scheme for the table that specifies which table rows are located in which dbspaces.
- Create multiple partitions of a fragmented table within a single dbspace if the fragmented table uses an expression-based or round-robin distribution scheme.

A table or table fragment is located completely in the dbspace in which it was created. The database server administrator can use this fact to limit the growth of a table by placing a table in a dbspace and then refusing to add a chunk to the dbspace when it becomes full.

For more information about distribution schemes, see the *IBM Informix Database Design and Implementation Guide*. For information about how to fragment tables and indexes over multiple disks to improve performance, concurrency, data availability, and backups, see your *IBM Informix Performance Guide*.

A table, composed of extents, can span multiple chunks, as the following figure shows.

Two extents, both allocated
to the same table
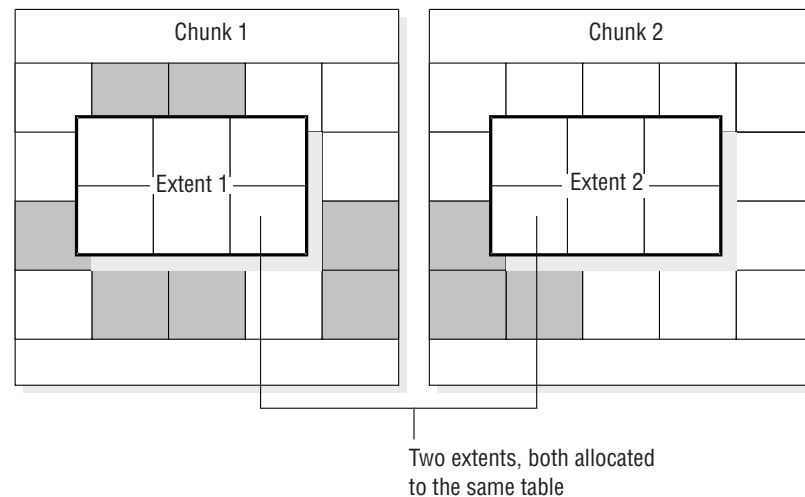
Figure 8-14. Table that spans more than one chunk

Simple large objects are located in blobpages in either the dbspace with the data pages of the table or in a separate blobspace. When you use the Optical Subsystem, you can also store simple large objects in an optical storage subsystem.

For advice on where to store your tables, see "Disk-layout guidelines" on page 8-35 and your *IBM Informix Performance Guide*.

Damaged tables

The following items can damage a table:
- An incorrect buffer flush
- A user error
- Mounting a files system or another chunk on top of a chunk
- Deleting or updating when the scope of the change is not as narrow as you require

Damaged indexes can cause a table to seem damaged, even though it is not.

The **oncheck** commands cannot fix most damaged tables. If a page is damaged, **oncheck** can detect and try to fix the page, but cannot correct the data within the page.

Table types for Informix

You can create logging or nonlogging tables in a logging database on IBM Informix. The two table types are STANDARD (logging tables) and RAW (nonlogging tables). The default standard table is like a table created in earlier versions without a special keyword specified. You can create either a STANDARD or RAW table and change tables from one type to another.

In a nonlogging database, both STANDARD tables and RAW tables are nonlogging. In a nonlogging database, the only difference between STANDARD and RAW tables is that RAW tables do not support primary-key constraints, unique constraints, referential constraints, or rollback. However, these tables can be indexed and updated.

The following table lists the properties of the types of tables available with Informix. The flag values are the hexadecimal values for each table type in the **flags** column of **systables**.

Table 8-4. Table types for Informix

Characteristic	STANDARD	RAW	TEMP
Permanent	Yes	Yes	No
Logged	Yes	No	Yes
Indexes	Yes	Yes	Yes
Constraints	Yes	No referential or unique constraints NULL and NOT NULL constraints are allowed	Yes
Rollback	Yes	No	Yes
Recoverable	Yes	Yes, if not updated	No
Restorable	Yes	Yes, if not updated	No
Loadable	Yes	Yes	Yes
Enterprise Replication servers	Yes	No	No
Primary servers in a high-availability cluster	Yes	Yes, cannot alter logging mode	Yes
Secondary servers in a high-availability cluster	Yes	Yes, but not accessible for any operation	Yes
Flag Value	None	0x10	None

Standard permanent tables

A STANDARD table is the same as a table in a logged database that the database server creates. STANDARD tables do not use light appends. All operations are logged, record by record, so STANDARD tables can be recovered and rolled back. You can back up and restore STANDARD tables. Logging enables updates since the last physical backup to be applied when you perform a warm restore or point-in-time restore. Enterprise Replication is allowed on STANDARD tables.

A STANDARD table is the default type on both logging and nonlogging databases. STANDARD tables are logged if stored in a logging database but are not logged if stored in a nonlogging database.

RAW tables

RAW tables are nonlogging permanent tables that are similar to tables in a nonlogging database. Update, insert, and delete operations on rows in a RAW table are supported but are not logged. You can define indexes on RAW tables, but you cannot define unique constraints, primary-key constraints, or referential constraints on RAW tables. Light appends are not supported for loading RAW tables, except in High-Performance Loader (HPL) operations and in queries that specify INTO TEMP ... WITH NO LOG.

A RAW table has the same attributes, whether it is stored in a logging database or in a nonlogging database. If you update a RAW table, you cannot reliably restore the data unless you perform a level-0 backup after the update. If the table has not been updated since that backup, you can restore the RAW table from the last physical backup, but backing up only the logical logs is not sufficient for a RAW table to be recoverable. Fast recovery can roll back incomplete transactions on STANDARD tables but not on RAW tables. For information about creating and altering RAW tables, see the *IBM Informix Guide to SQL: Syntax*.

RAW tables are intended for the initial loading and validation of data. To load RAW tables, you can use any loading utility, including **dbexport**, the LOAD statement of DB-Access, or the HPL in express mode. If an error or failure occurs while loading a RAW table, the resulting data is whatever was on the disk at the time of the failure.

Restriction: Do not use RAW tables within a transaction. After you have loaded the data, use the ALTER TABLE statement to change the table to type STANDARD and perform a level-0 backup before you use the table in a transaction.

Restriction: Do not use Enterprise Replication on RAW or TEMP tables.

There are some restrictions when using RAW tables in a high-availability cluster environment. Because modifications made to RAW tables are not logged, and because secondary servers (including HDR, RSS and SDS) use log records to stay synchronized with the primary server, you are restricted from performing certain operations on RAW tables:

- On a primary server, RAW tables can be created, dropped, and accessed; however, altering the table mode from unlogged to logged, or from logged to unlogged, is not allowed. Altering a table's mode in a high-availability cluster environment yields error -19845.
- On secondary servers (HDR, SDS, or RSS), RAW tables are not accessible for any operation. Attempting to access a RAW table from SQL yields error -19846.

Temp tables

Temp tables are temporary, logged tables that are dropped when the user session closes, the database server shuts down, or on reboot after a failure. Temp tables support indexes, constraints, and rollback. You cannot recover, back up, or restore temp tables. Temp tables support bulk operations such as light appends, which add rows quickly to the end of each table fragment. For more information about light appends, see your *IBM Informix Performance Guide*.

For more information, see "Temporary tables" on page 8-27.

Properties of table types

These topics explain loading tables, fast recovery, and backup and restore of table types.

Loading of data into a table

IBM Informix creates STANDARD tables that use logging by default. Data warehousing applications can have huge tables that take a long time to load. Nonlogging tables are faster to load than logging tables. You can use the CREATE RAW TABLE statement to create a RAW table or use the ALTER TABLE statement to change a STANDARD table to RAW before loading the table. After you load the table, run UPDATE STATISTICS on it.

For more information about how to improve the performance of loading very large tables, see your *IBM Informix Performance Guide*. For more information about using ALTER TABLE to change a table from logging to nonlogging, see the *IBM Informix Guide to SQL: Syntax*.

Fast recovery of table types

The following table shows fast recovery scenarios for the table types available with IBM Informix.

Table 8-5. Fast recovery of table types

Table type	Fast recovery behavior
Standard	Fast recovery is successful. All committed log records are rolled forward, and all incomplete transactions are rolled back.
RAW	If a checkpoint completed since the raw table was modified last, all the data is recoverable. Inserts, updates, and deletions that occurred after the last checkpoint are lost. Incomplete transactions in a RAW table are not rolled back.

Backup and restore of RAW tables

The following table explains backup scenarios for the table types available on IBM Informix.

Table 8-6. Backing up tables on Informix

Table type	Backup allowed?
Standard	Yes.
Temp	No.
RAW	Yes. If you update a RAW table, you must back it up so that you can restore all the data in it. Backing up only the logical logs is not enough.

Important: After you load a RAW table or change a RAW table to type STANDARD, you must perform a level-0 backup.

The following table shows restore scenarios for these table types.

Table 8-7. Restoring tables on Informix

Table type	Restore allowed?
Standard	Yes. Warm restore, cold restore, and point-in-time restore work.

Table 8-7. Restoring tables on Informix (continued)

Table type	Restore allowed?
Temp	No.
RAW	When you restore a RAW table, it contains only data that was on disk at the time of the last backup. Because RAW tables are not logged, any changes that occurred since the last backup are not restored.

Temporary tables

The database server must provide disk space for temporary tables of the following two kinds:

- Temporary tables that you create with an SQL statement, such as CREATE TEMP TABLE. . .
- Temporary tables that the database server creates as it processes a query

Make sure that your database server has configured enough temporary space for both user-created and database server-created temporary tables. Some uses of the database server might require as much temporary storage space as permanent storage space, or more.

By default, the database server stores temporary tables in the root dbspace. If you decide not to store your temporary tables in the root dbspace, use the **DBSPACETEMP** environment variable or the DBSPACETEMP configuration parameter to specify a list of dbspaces for temporary tables.

Temporary tables that you create

You can create temporary tables with any of the following SQL statements:

- TEMP TABLE option of the CREATE TABLE statement
- INTO TEMP clause of the SELECT statement, such as SELECT * FROM customer INTO TEMP cust_temp

Only the session that creates a temporary table can use the table. When the session exits, the table is dropped automatically.

When you create a temporary table, the database server uses the following criteria:

- If the query used to populate the TEMP table produces no rows, the database server creates an empty, unfragmented table.
- If the rows that the query produces do not exceed 8 KB, the temporary table is located in only one dbspace.
- If the rows exceed 8 KB, the database server creates multiple fragments and uses a round-robin fragmentation scheme to populate them unless you specify a fragmentation method and location for the table.

If you use the CREATE TEMP and SELECT...INTO TEMP SQL statements and DBSPACETEMP has been set:

- LOGGING dbspaces in the list are used to create the tables that specify or imply the WITH LOG clause.
- NON-LOGGING temporary dbspaces in the list are used to create the tables that specify the WITH NO LOG clause.

When CREATE TEMP and SELECT...INTO TEMP SQL statements are used and DBSPACETEMP has not been set or does not contain the correct type of dbspace,

IBM Informix uses the dbspace of the database to store the temporary table. See the *IBM Informix Guide to SQL: Syntax* for more information.

Where user-created temporary tables are stored:

If your application lets you specify the location of a temporary table, you can specify either logging spaces or nonlogging spaces that you create exclusively for temporary tables.

For information about creating temporary dbspaces, see the **onspaces** topics in the *IBM Informix Administrator's Reference*.

If you do not specify the location of a temporary table, the database server stores the temporary table in one of the spaces that you specify as an argument to the DBSPACETEMP configuration parameter or environment variable. The database server remembers the name of the last dbspace that it used for a temporary table. When the database server receives another request for temporary storage space, it uses the next available dbspace to spread I/O evenly across the temporary storage space.

For information about where the database stores temporary tables when you do not list any spaces as an argument to DBSPACETEMP, see the DBSPACETEMP section in the *IBM Informix Administrator's Reference*.

When you use an application to create a temporary table, you can use the temporary table until the application exits or performs one of the following actions:
* Closes the database in which the table was created and opens a database in a different database server
* Closes the database in which the table was created
* Explicitly drops the temporary table

Temporary tables that the database server creates
The database server sometimes creates temporary tables while running queries against the database or backing it up.

The database server might create a temporary table in any of the following circumstances:
* Statements that include a GROUP BY or ORDER BY clause
* Statements that use aggregate functions with the UNIQUE or DISTINCT keywords
* SELECT statements that use auto-index or hash joins
* Complex CREATE VIEW statements
* DECLARE statements that create a scroll cursor
* Statements that contain correlated subqueries
* Statements that contain subqueries that occur within an IN or ANY clause
* CREATE INDEX statements

When the process that initiated the creation of the table is complete, the database server deletes the temporary tables that it creates.

If the database server shuts down without removing temporary tables, the database server removes the temporary tables the next time it is started. To start the database server without removing temporary tables, run the **oninit** command with the **-p** option.

Applications and analytic tools can define queries in which a derived table contains multiple views joined with base tables, potentially including hundreds of columns. The database server attempts to fold views or derived tables into the main query. Any such views or derived tables that cannot be folded are materialized into a temporary table. The temporary table excludes all the columns that are not referenced in the main query. The temporary table is created with only the columns referenced in the Projection clause and in other clauses of the parent query, including the WHERE, HAVING, GROUP BY, and ON clauses.

By excluding from the system-generated temporary table any columns that are not referenced in the main query, this reduced schema can improve query performance by conserving storage resources, and by avoiding unnecessary I/O of data in the unused columns.

In a nested query, however, projected columns from views and derived table are checked only in the parent query, but not in the levels above the immediate parent query.

Where database server-created temporary tables are stored: When the database server creates a temporary table, it stores the temporary table in one of the dbspaces that you specify in the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable. The environment variable supersedes the configuration parameter.

When you do not specify any temporary dbspaces in DBSPACETEMP, or the temporary dbspaces that you specify have insufficient space, the database server creates the table in a standard dbspace according to the following rules:

- If you created the temporary table with CREATE TEMP TABLE, the database server stores this table in the dbspace that contains the database to which the table belongs.
- If you created the temporary table with the INTO TEMP option of the SELECT statement, the database server stores this table in the root dbspace.

For more information, see "Creating a temporary dbspace" on page 9-17.

Tblspaces

Database server administrators sometimes must track disk use by a particular table. A *tblspace* contains all the disk space allocated to a given table or table fragment (if the table is fragmented). A separate tblspace contains the disk space allocated for the associated index.

A tblspace, for example, does not correspond to any particular part of a chunk or even to any particular chunk. The indexes and data that make up a tblspace might be scattered throughout your chunks. The tblspace, however, represents a convenient accounting entity for space across chunks devoted to a particular table. (See "Tables" on page 8-22.)

Maximum number of tblspaces in a table

You can specify a maximum of 2**20 (or 1,048,576) tblspaces in a table.

Table and index tblspaces

The table tblspace contains the following types of pages:

- Pages allocated to data
- Pages allocated to indexes
- Pages used to store TEXT or BYTE data in the dbspace (but not pages used to store TEXT or BYTE data in a blobspace)
- Bitmap pages that track page use within the table extents

The index tblspace contains the following types of pages:

- Pages allocated to indexes
- Bitmap pages that track page use within the index extents

The following table illustrates the tblspaces for three tables that form part of the **stores_demo** database. Only one table (or table fragment) exists per tblspace. An index is located in a separate tblspace from the associated table. Blobpages represent TEXT or BYTE data stored in a dbspace.

stores_demo Database

customer data tblspace		
Data page	Data page	Data page
Data page	Data page	Data page
Data page	Data page	Data page
		Bitmap page

catalog data tblspace		
Data page	Data page	Data page
Data page	Data page	Data page
Data page	Data page	Data page
Blob-page	Blob-page	Blob-page
Blob-page	Blob-page	Bitmap page

orders data tblspace		
Data page	Data page	Data page
Data page	Data page	Data page
Data page	Data page	Data page
Index page	Index page	Bitmap page
Index page		

customer index tblspace		
Index page	Index page	Bitmap page

catalog index tblspace		
Index page	Index page	Index page
Index page	Index page	Bitmap page

orders index tblspace		
Index page	Index page	Index page
		Bitmap page

Figure 8-15. Sample tblspaces in the stores_demo database

Extent interleaving

The database server allocates the pages that belong to a tblspace as extents. Although the pages within an extent are contiguous, extents might be scattered throughout the dbspace where the table is located (even on different chunks).

The following figure depicts this situation with two noncontiguous extents that belong to the tblspace for **table_1** and a third extent that belongs to the tblspace for **table_2**. A **table_2** extent is located between the first table_1 extent and the second **table_1** extent. When this situation occurs, the extents are interleaved. Because sequential access searches across **table_1** require the disk head to seek

across the **table_2** extent, performance is worse than if the **table_1** extents were contiguous. For instructions on how to avoid and eliminate interleaving extents, see your *IBM Informix Performance Guide*.
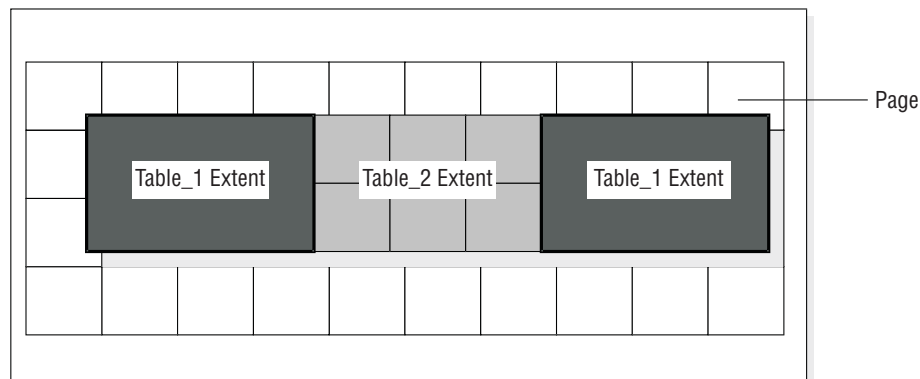
Figure 8-16. Three extents that belong to two different tblspaces in a single dbspace

Table fragmentation and data storage

The fragmentation feature gives you additional control over where the database stores data. You are not limited to specifying the locations of individual tables and indexes. You can also specify the location of table and index *fragments*, which are different parts of a table or index that are located on different storage spaces. You can fragment the following storage spaces:

* Dbspaces
* Sbspaces

Usually you fragment a table when you initially create it. The CREATE TABLE statement takes one of the following forms:

```
CREATE TABLE tablename ... FRAGMENT BY ROUND ROBIN IN dbspace1,
 dbspace2, dbspace3;

CREATE TABLE tablename ...FRAGMENT BY EXPRESSION
   <Expression 1> in dbspace1,
   <Expression 2> in dbspace2,
   <Expression 3> in dbspace3;
```

The FRAGMENT BY ROUND ROBIN and FRAGMENT BY EXPRESSION keywords refer to two different distribution schemes. Both statements associate fragments with dbspaces.

When you fragment a table, you can also create multiple partitions of the table within the same dbspace, as shown in this example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
        PARTITION part1 (a >=0 AND a < 5) in dbs1,
        PARTITION part2 (a >=5 AND a < 10) in dbs1
         ...
      ;
```

The following figure illustrates the role of fragments in specifying the location of data.
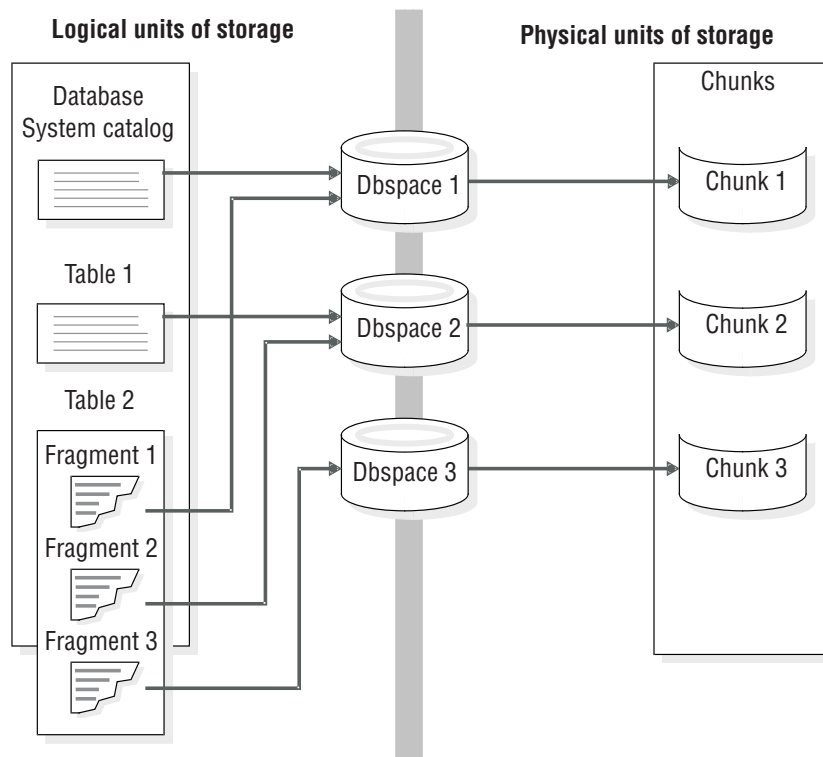
Figure 8-17. Dbspaces that link logical units (including table fragments) and physical units of storage

For information about spaces and partitions, see Chapter 9, "Manage disk space," on page 9-1.

For more information about fragmentation, see the *IBM Informix Database Design and Implementation Guide* and the *IBM Informix Performance Guide*.

Amount of disk space needed to store data

To determine how much disk space you require, follow these steps:

1. Calculate the size requirements of the root dbspace.
2. Estimate the total amount of disk space to allocate to all the database server databases, including space for overhead and growth.

The following topics explain these steps.

Size of the root dbspace

You can calculate the size of the root dbspace, which stores information that describes your database server.

To calculate the size of the root dbspace, take the following storage structures into account:

- The physical log (200 KB minimum)
- The logical-log files (200 KB minimum)
- Temporary tables
- Data

- System databases (**sysmaster**, **sysutils**, **syscdr**, **sysuuid**) and system catalogs (the size varies between versions)
- Reserved pages (~24 KB)
- Tblspace tblspace (100 to 200 KB minimum)
- Extra space

This estimate is the root dbspace size before you initialize the database server. The size of the root dbspace depends on whether you plan to store the physical log, logical logs, and temporary tables in the root dbspace or in another dbspace. The root dbspace must be large enough for the minimum size configuration during disk initialization.

Recommendation: Set up the system with a small log size (for example, three 1000 KB log files, or 3000 KB for the total log size). After setup is complete, create new dbspaces, move and resize the logical-log files, and drop the original logs in the root dbspace. Then move the physical log to another dbspace. This procedure minimizes the affect of the logs in the root dbspace because:

- A large amount of space is not left unused in the root dbspace after you move the logs.
- The logs do not contend for space and I/O on the same disk as the root dbspace.

For details on how to move the logs, see "Moving a logical-log file to another dbspace" on page 14-12 and "Change the physical-log location and size" on page 16-1.

If you need to make the root dbspace larger after the server has been initialized, you can add a new chunk to the root dbspace. In addition, you might be able to extend a chunk in the root dbspace by using "Automatic space management" on page 9-26.

Related concepts:

"Automatic space management" on page 9-26

"Manage dbspaces" on page 9-7

Related tasks:

"Marking a chunk as extendable or not extendable" on page 9-29

"Manually expanding a space or extending an extendable chunk" on page 9-31

Related reference:

⇨ ROOTSIZE configuration parameter (Administrator's Reference)

Physical and logical logs

The value stored in the ONCONFIG parameter PHYSFILE defines the size of your physical log when the database server is initially created. After you use the **oninit -i** command to initialize the disk space and bring the database server online, use the **onparams** utility to change the physical log location and size. Advice on sizing your physical log is contained in "Size and location of the physical log" on page 15-2.

To calculate the size of the logical-log files, multiply the value of the ONCONFIG parameter LOGSIZE by the number of logical-log files. For advice on sizing your logical log, see "Estimate the size and number of log files" on page 14-1.

Temporary tables

Analyze end-user applications to estimate the amount of disk space that the database server might require for temporary tables. Try to estimate how many of these statements are to run concurrently. The space occupied by the rows and columns that are returned provides a good basis for estimating the amount of space required.

The largest temporary table that the database server creates during a warm restore is equal to the size of your logical log. You calculate the size of your logical log by adding the sizes of all logical-log files.

You must also analyze end-user applications to estimate the amount of disk space that the database server might require for explicit temporary tables.

For more information, including a list of statements that require temporary space, see "Temporary tables" on page 8-27.

Critical data

Restriction: Do not store databases and tables in the root dbspace. Mirror the root dbspace and other dbspaces that contain critical data such as the physical log and logical logs. Estimate the amount of disk space, if any, that you require to allocate for tables stored in the root dbspace.

Extra space

Allow extra space in the root dbspace for the system databases to grow, for the extended reserved pages, and ample free space. The number of extended reserved pages depends on the number of primary chunks, mirror chunks, logical-log files, and storage spaces in the database server.

Amount of space that databases require

The amount of additional disk space required for the database server data storage depends on the requirements of users, plus overhead and growth. Every application that users run has different storage requirements. The following list suggests some of the steps that you can take to calculate the amount of disk space to allocate (beyond the root dbspace):

*   Decide how many databases and tables you must to store. Calculate the amount of space required for each one.
*   Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
*   Decide which databases and tables you want to mirror.

For instructions about calculating the size of your tables, see your *IBM Informix Performance Guide*.

The storage pool

Every instance of Informix has a storage pool. The storage pool contains information about the directories, cooked files, and raw devices that the server can use if necessary to automatically expand an existing dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

When the storage space falls below a threshold defined in the SP_THRESHOLD configuration parameter, Informix can automatically run a task that expands the space, either by extending an existing chunk in the space or by adding a new chunk.

You can use SQL administration API commands to:
- Add, delete, or modify an entry that describes one directory, cooked file, or raw device in the storage pool. The server can use the specified directory, cooked file, or raw device when necessary to automatically add space to an existing storage space.
- Control how a storage pool entry is used by modifying two different dbspace sizes that are associated with expanding a storage space, the extend size and the create size.
- Mark a chunk as extendable or not extendable.
- Immediately expand the size of a space, when you do not want Informix to automatically expand the space.
- Immediately extend the size of a chunk by a specified minimum amount.
- Create a storage space or chunk from an entry in the storage pool
- Return empty space from a dropped storage space or chunk to the storage pool

The **storagepool** table in **sysadmin** database contains information about all of the entries in a storage pool for an Informix instance.

Related concepts:

"Extendable chunks" on page 8-4

"Automatic space management" on page 9-26

Related tasks:

"Creating and managing storage pool entries" on page 9-27

Disk-layout guidelines

The following goals are typical for efficient disk layout:
- Limiting disk-head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

You must make some trade-offs among these goals when you design your disk layout. For example, separating the system catalog tables, the logical log, and the physical log can help reduce contention for these resources. However, this action can also increase the chances that you must perform a system restore. For detailed disk-layout guidelines, see the *IBM Informix Performance Guide*.

Dbspace and chunk guidelines

This topic lists some general strategies for disk layout that do not require any information about the characteristics of a particular database:
- Associate disk partitions with chunks and allocate at least one additional chunk for the root dbspace.

  A disk that is already partitioned might require the use of offsets. For details, see "Allocating raw disk space on UNIX" on page 9-3.

Tip: With the 4-terabyte maximum size of a chunk, you can avoid partitioning by assigning a chunk per disk drive.

- Mirror critical dbspaces: the root dbspace, the dbspaces that contain the physical log and the logical-log files. Also mirror high-use databases and tables.

  You specify mirroring at the dbspace level. Mirroring is either on or off for all chunks belonging to a dbspace. Locate the primary and the mirrored dbspaces on different disks. Ideally, different controllers handle the different disks.

- Spread temporary tables and sort files across multiple disks.

  To define several dbspaces for temporary tables and sort files, use **onspaces -t**. When you place these dbspaces on different disks and list them in the DBSPACETEMP configuration parameter, you can spread the I/O associated with temporary tables and sort files across multiple disks. For information about using the DBSPACETEMP configuration parameter or environment variable, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference*.

- Keep the physical log in the root dbspace but move the logical logs from the root dbspace. However, if you plan to store the system catalogs in the root dbspace, move the physical log to another dbspace.

  For advice on where to store your logs, see "Specify the location of the physical log" on page 15-2 and "Location of logical-log files" on page 13-1. Also see "Moving a logical-log file to another dbspace" on page 14-12 and "Change the physical-log location and size" on page 16-1.

- To improve backup and restore performance:
  - Cluster system catalogs with the data that they track.
  - If you use ON-Bar to perform parallel backups to a high-speed tape drive, store the databases in several small dbspaces.

    For additional performance recommendations, see the *IBM Informix Backup and Restore Guide*.

Table-location guidelines

This topic lists some strategies for optimizing the disk layout, given certain characteristics about the tables in a database. You can implement many of these strategies with a higher degree of control using table fragmentation:

- Isolate high-use tables on a separate disk.

  To isolate a high-use table on its own disk device, assign the device to a chunk, and assign the same chunk to a dbspace. Finally, place the frequently used table in the dbspace just created using the IN *dbspace* option of CREATE TABLE.

  To display the level of I/O operations against each chunk, run the **onstat -g iof** option.

- Fragment high-use tables over multiple disks.
- Group related tables in a dbspace.

  If a device that contains a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. Although you must perform a cold restore if a dbspace that contains critical information fails, you must only perform a warm restore if a noncritical dbspace fails.

- Place high-use tables on the middle partition of a disk.
- Optimize table extent sizes.

For more information, see the chapter on table performance considerations in your *IBM Informix Performance Guide*. For information about **onstat** options, see the *IBM Informix Administrator's Reference*.

Sample disk layouts

When setting out to organize disk space, the database server administrator usually has one or more of the following objectives in mind:

- High performance
- High availability
- Ease and frequency of backup and restore

Meeting any one of these objectives has trade-offs. For example, configuring your system for high performance usually results in taking risks regarding the availability of data. The sections that follow present an example in which the database server administrator must make disk-layout choices given limited disk resources. These sections describe two different disk-layout solutions. The first solution represents a performance optimization, and the second solution represents an availability-and-restore optimization.

The setting for the sample disk layouts is a fictitious sporting goods database that uses the structure (but not the volume) of the **stores_demo** database. In this example, the database server is configured to handle approximately 350 users and 3 gigabytes of data. The disk space resources are shown in the following table.

Disk drive	Size of drive	High performance
Disk 1	2.5 gigabytes	No
Disk 2	3 gigabytes	Yes
Disk 3	2 gigabytes	Yes
Disk 4	1.5 gigabytes	No

The database includes two large tables: **cust_calls** and **items**. Assume that both of these tables contain more than 1,000,000 rows. The **cust_calls** table represents a record of all customer calls made to the distributor. The items table contains a line item of every order that the distributor ever shipped.

The database includes two high-use tables: **items** and **orders**. Both of these tables are subject to constant access from users around the country.

The remaining tables are low-volume tables that the database server uses to look up data such as postal code or manufacturer.

Table name	Maximum size	Access rate
cust_calls	2.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low
state	50 megabytes	Low
call_type	50 megabytes	Low

Sample layout when performance is highest priority

The following figure shows a disk layout optimized for performance. This disk layout uses the following strategies to improve performance:

- Migration of the logical log from the rootdbs dbspace to a dbspace on a separate disk

  This strategy separates the logical log and the physical log and reduces contention for the root dbspace.

- Location of the two tables that undergo the highest use in dbspaces on separate disks

  Neither of these disks stores the logical log or the physical log. Ideally you might store each of the **items** and **orders** tables on a separate high-performance disk. However, in the present scenario, this strategy is not possible because one of the high-performance disks is required to store the very large **cust_calls** table (the other two disks are too small for this task).
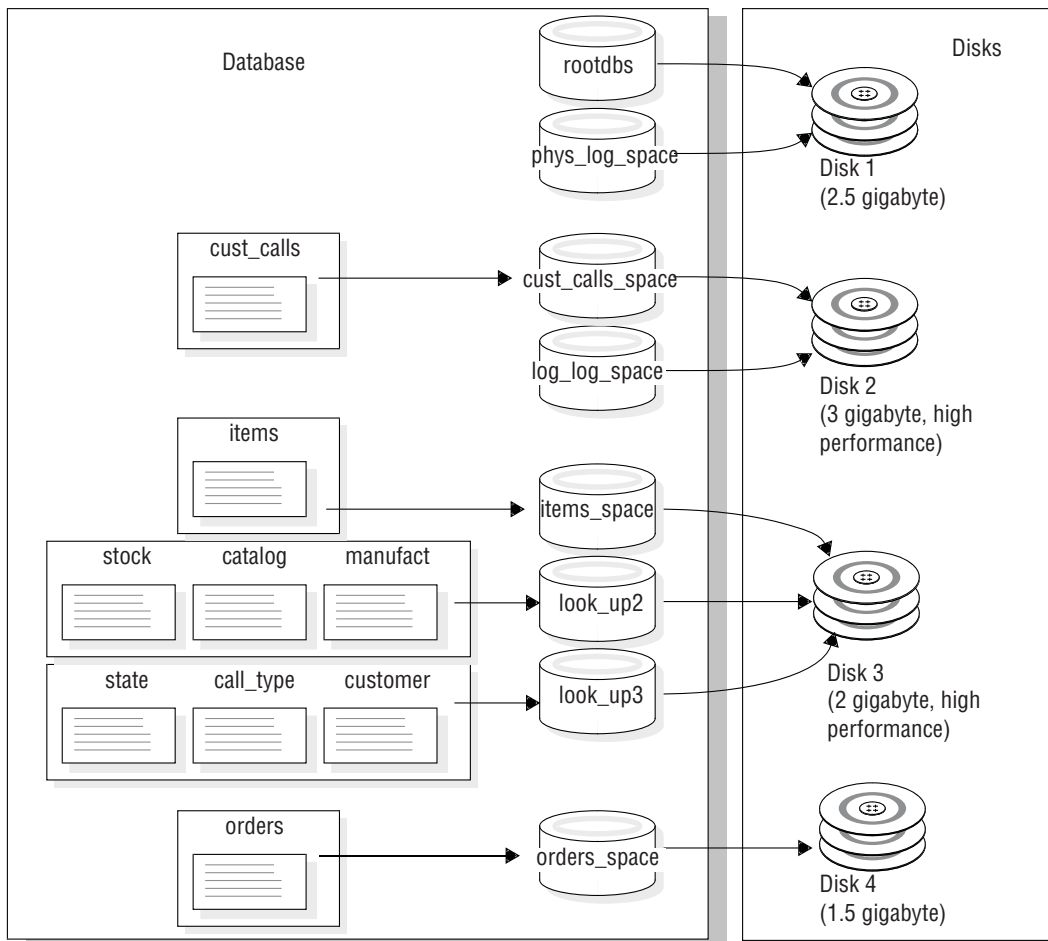
Figure 8-18. Disk layout optimized for performance

Sample layout when availability is highest priority

The weakness of the previous disk layout is that if either Disk 1 or Disk 2 fails, the whole database server goes down until you restore the dbspaces on these disks from backups. In other words, the disk layout is poor with respect to availability.

An alternative disk layout that optimizes for availability and involves mirroring is shown in following figure. This layout mirrors all the critical data spaces (the system catalog tables, the physical log, and the logical log) to a separate disk. Ideally you might separate the logical log and physical log (as in the previous layout) and mirror each disk to its own mirror disk. However, in this scenario, the required number of disks does not exist; therefore, the logical log and the physical log both are located in the root dbspace.
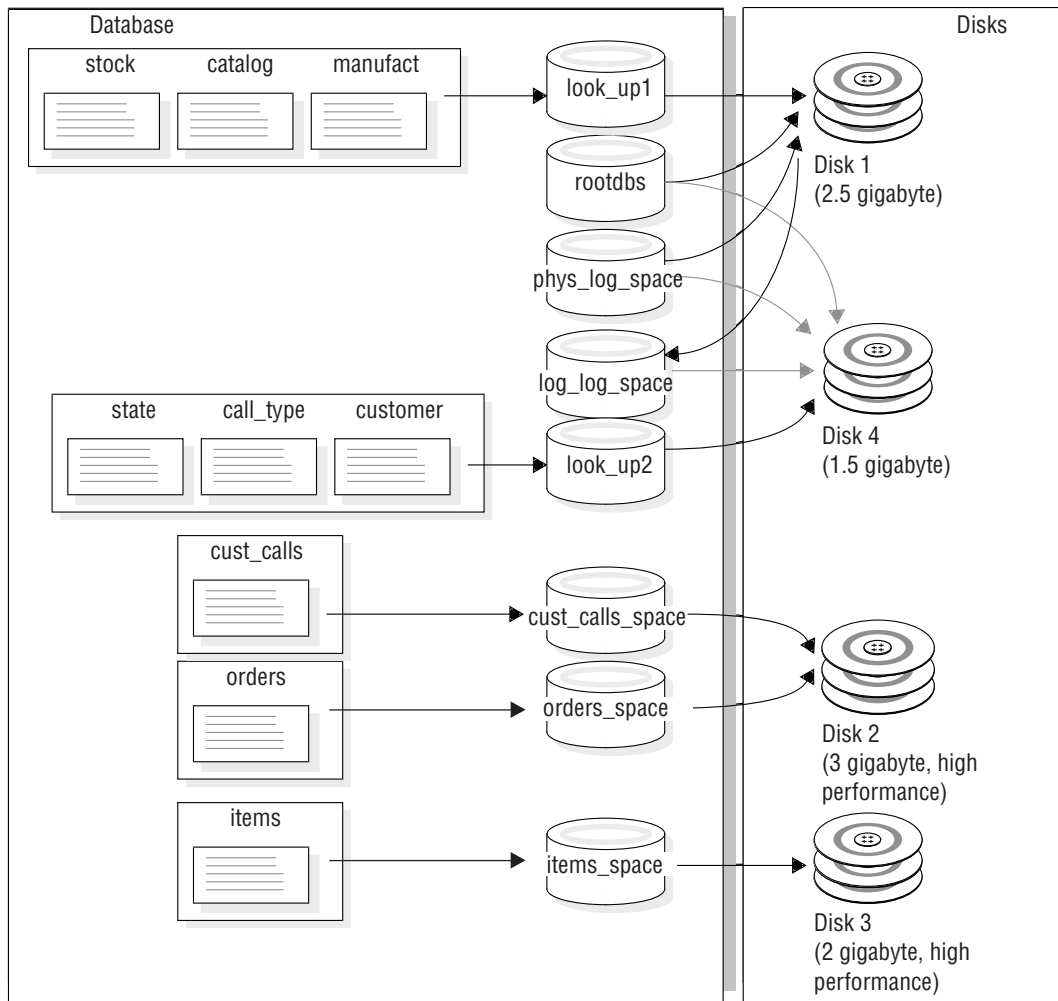
Figure 8-19. Disk layout optimized for availability

Logical-volume manager

You can use the logical-volume manager (LVM) utility to manage your disk space through user-defined logical volumes.

Many computer manufacturers ship their computers with a proprietary LVM. You can use the database server to store and retrieve data on disks that are managed by most proprietary LVMs. Logical-volume managers provide some advantages and some disadvantages, as explained in the remainder of this section.

Most LVMs can manage multiple gigabytes of disk space. The database server chunks are limited to a size of 4 terabytes, and this size can be attained only when

the chunk being allocated has an offset of zero. Consequently, you must limit the size of any volumes to be allocated as chunks to a size of 4 terabytes.

Because you can use LVMs to partition a disk drive into multiple volumes, you can control where data is placed on a given disk. You can improve performance by defining a volume that consists of the middle-most cylinders of a disk drive and placing high-use tables in that volume. (Technically, you do not place a table directly in a volume. You must first allocate a chunk as a volume, then assign the chunk to a dbspace, and finally place the table in the dbspace. For more information, see "Control of where simple large object data is stored" on page 8-9.)

Tip: If you choose to use large disk drives, you can assign a chunk to one drive and eliminate the necessity to partition the disk.

You can also improve performance by using a logical volume manager to define a volume that spreads across multiple disks and then placing a table in that volume.

Many logical volume managers also allow a degree of flexibility that standard operating-system format utilities do not. One such feature is the ability to reposition logical volumes after you define them. Thus getting the layout of your disk space right the first time is not so critical as with operating-system format utilities.

LVMs often provide operating-system-level mirroring facilities. For more information, see "Alternatives to mirroring" on page 17-2.

Chapter 9. Manage disk space

You can use several utilities and tools to manage disk spaces and the data that the database server controls.

These topics assume that you are familiar with the terms and concepts contained in Chapter 8, "Data storage," on page 8-1.

You can use the following utilities to manage storage spaces:
* **onspaces** utility commands
* OAT
* IBM Informix Server Administrator

Your *IBM Informix Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents, how to reclaim space in an empty extent, and how to improve disk I/O.

For information about using SQL administration API commands instead of some **onspaces** commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Administrator's Reference*.

You can generate SQL administration API or **onspaces** commands for reproducing the storage spaces, chunks, and logs that exist in a file. You do this with the **dbschema** utility.

Related reference:

↪ The onspaces utility (Administrator's Reference)

↪ Storage space, chunk, and log creation (Migration Guide)

Allocate disk space

This section explains how to allocate disk space for the database server. Read the following sections before you allocate disk space:
* "Unbuffered or buffered disk access on UNIX" on page 8-3
* "Amount of disk space needed to store data" on page 8-32
* "Disk-layout guidelines" on page 8-35

Before you can create a storage space or chunk, or mirror an existing storage space, you must allocate disk space for the chunk file. You can allocate either an empty file or a portion of raw disk for database server disk space.

UNIX only: On UNIX, if you allocate raw disk space, you must use the UNIX **ln** command to create a link between the character-special device name and another file name. For more information about this topic, see "Create symbolic links to raw devices (UNIX)" on page 9-3.

Using a UNIX file and its inherent operating-system interface for database server disk space is called using *cooked space*.

Windows only: On Windows, you must use NTFS files for database server disk space. For more information about this recommendation, see "Unbuffered or buffered disk access on UNIX" on page 8-3.

You can balance chunks over disks and controllers. Placing multiple chunks on a single disk can improve throughput.

Specify an offset

When you allocate a chunk of disk space to the database server, specify an offset for one of the following two purposes:

- To prevent the database server from overwriting the partition information
- To define multiple chunks on a partition, disk device, or cooked file

The maximum value for the offset is 4 terabytes.

Many computer systems and some disk-drive manufacturers keep information for a physical disk drive on the drive itself. This information is sometimes called a *volume table of contents* (VTOC) or disk label. The VTOC is commonly stored on the first track of the drive. A table of alternative sectors and bad-sector mappings (also called a *revectoring table*) might also be stored on the first track.

If you plan to allocate partitions at the start of a disk, you might be required to use offsets to prevent the database server from overwriting critical information required by the operating system. For the exact offset required, see your disk-drive manuals.

Important: If you are running two or more instances of the database server, be extremely careful not to define chunks that overlap. Overlapping chunks can cause the database server to overwrite data in one chunk with unrelated data from an overlapping chunk. This overwrite effectively deletes overlapping data.

Specify an offset for the initial chunk of root dbspace

For the initial chunk of root dbspace and its mirror, if it has one, specify the offsets with the ROOTOFFSET and MIRROROFFSET parameters, respectively. For more information, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

Specify an offset for additional chunks

To specify an offset for additional chunks of database server space, you must supply the offset as a parameter when you assign the space to the database server. For more information, see "Creating a dbspace that uses the default page size" on page 9-7.

Use offsets to create multiple chunks

You can create multiple chunks from a disk partition, disk device, or file, by specifying offsets and assigning chunks that are smaller than the total space available. The offset specifies the beginning location of a chunk. The database server determines the location of the last byte of the chunk by adding the size of the chunk to the offset.

For the first chunk, assign any initial offset, if necessary, and specify the size as an amount that is less than the total size of the allocated disk space. For each additional chunk, specify the offset to include the sizes of all previously assigned chunks, plus the initial offset, and assign a size that is less than or equal to the amount of space remaining in the allocation.

Allocating cooked file spaces on UNIX

The following procedure shows an example of allocating disk space for a cooked file, called usr/data/my_chunk, on UNIX.

To allocate cooked file space:
1. Log-in as user **informix**: **su informix**
2. Change directories to the directory where the cooked space will be located: **cd /usr/data**
3. Create your chunk by concatenating null to the file name that the database server will use for disk space: **cat /dev/null > my_chunk**
4. Set the file permissions to 660 (rw-rw----): **chmod 660 my_chunk**
5. You must set both group and owner of the file to **informix**:

   ```
   ls -l my_chunk   -rw-rw----
     1  informix   informix
     0  Oct 12 13:43 my_chunk
   ```
6. Use **onspaces** to create the storage space or chunk.

For information about how to create a storage space using the file you have allocated, see "Creating a dbspace that uses the default page size" on page 9-7, "Creating a blobspace" on page 9-20, and "Creating an sbspace" on page 9-23.

Allocating raw disk space on UNIX

To allocate raw space, you must have a disk partition available that is dedicated to raw space. To create raw disk space, you can either repartition your disks or unmount an existing file system. Back up any files before you unmount the device.

To allocate raw disk space
1. Create and install a raw device.

   For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation and "Unbuffered or buffered disk access on UNIX" on page 8-3.
2. Change the ownership and permissions of the character-special devices to **informix**.

   The file name of the character-special device usually begins with the letter r. For the procedure, see steps 4 and 5 in "Allocating cooked file spaces on UNIX."
3. Verify that the operating-system permissions on the character-special devices are crw-rw----.
4. Create a symbolic link between the character-special device name and another file name with the UNIX link command, **ln -s**. For details, see "Create symbolic links to raw devices (UNIX)."

Restriction: After you create the raw device that the database server uses for disk space, do not create file systems on the same raw device that you allocate for the database server disk space. Also, do not use the same raw device as swap space that you allocate for the database server disk space.

Create symbolic links to raw devices (UNIX)

Use symbolic links to assign standard device names and to point to the device. To create a link between the character-special device name and another file name, use the UNIX link command (usually **ln**). To verify that both the devices and the links exist, run the UNIX command **ls -l** (**ls -lg** on BSD) on your device directory. The

following example shows links to raw devices. If your operating system does not support symbolic links, hard links also work.

```
ln -s /dev/rxy0h /dev/my_root # orig_device link to symbolic_name
ln -s /dev/rxy0a /dev/raw_dev2
ls -l
crw-rw---  /dev/rxy0h
crw-rw---  /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Why use symbolic links? If you create chunks on a raw device and that device fails, you cannot restore from a backup until you replace the raw device and use the same path name. All chunks that were accessible at the time of the last backup must be accessible when you perform the restore.

Symbolic links simplify recovery from disk failure and enable you to replace quickly the disk where the chunk is located. You can replace a failed device with another device, link the new device path name to the same file name that you previously created for the failed device, and restore the data. You are not required to wait for the original device to be repaired.

Allocating NTFS file space on Windows

On Windows, the database server uses NTFS files by default. You can use standard file names for unbuffered files in the NTFS file system.

To allocate NTFS file space for database server disk space or mirrored space, the first step is to create a null (zero bytes) file.

To allocate NTFS file space for a dbspace, blobspace, or sbspace:
1. Log-in as a member of the **Informix-Admin** group.
2. Open an MS-DOS command shell.
3. Change to the directory where you want to allocate the space, as in the following example:
   ```
   c:> cd \usr\data
   ```
4. Create a null file with the following command: **c:> copy nul my_chunk**
5. If you want to verify that the file has been created, use the **dir** command to do so.

After you have allocated the file space, you can create the dbspace or other storage space as you normally would, using **onspaces**. For information about how to create a dbspace or a blobspace, see "Creating a dbspace that uses the default page size" on page 9-7 and "Creating a blobspace" on page 9-20.

Allocating raw disk space on Windows

You can configure raw disk space on Windows as a logical drive or physical drive. To find the drive letter or disk number, run the **Disk Administrator**. If the drives must be striped (multiple physical disks combined into one logical disk), only logical drive specification would work.

You must be a member of the **Informix-Admin** group when you create a storage space or add a chunk. The raw disk space can be formatted or unformatted disk space.

Important: If you allocate a formatted drive or disk partition as raw disk space and it contains data, the database server overwrites the data when it begins to use the disk space. You must ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.

To specify a logical drive:

1. Assign a drive letter to the disk partition.
2. Specify the following value for ROOTDBS in the `onconfig` file:
   `\\.\`*drive_letter*
3. To create a storage space or add a chunk, specify the logical drive partition.

   This example adds a chunk of 5000 KB on the `e:` drive, at an offset of 5200 KB, to dbspace **dpspc3**.

   ```
   onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
   ```

To specify a physical drive

1. If the disk partition has *not* been assigned a drive letter, specify the following value for ROOTDBS in the `onconfig` file: `\\.\PhysicalDrive<number>`
2. To create a storage space or add a chunk, specify the physical drive partition.

   This example adds a chunk of 5000 KB on **PhysicalDrive0**, at an offset of 5200 KB, to dbspace **dpspc3**.

   ```
   onspaces -a dbspc3 \\.\PhysicalDrive0 : -o 5200 -s 5000
   ```

Specify names for storage spaces and chunks

Chunk names follow the same rules as storage-space names. Specify an explicit path name for a storage space or chunk as follows:

- If you are using raw disks on UNIX, you must use a linked path name. (See "Create symbolic links to raw devices (UNIX)" on page 9-3.)
- If you are using raw disks on Windows, the path name takes the following form, where *x* specifies the disk drive or partition:

  `\\.\`*x*`:`

- If you are using a file for database server disk space, the path name is the complete path and file name.

Use these naming rules when you create storage spaces or add a chunk. The file name must have the following characteristics:

- Be unique and not exceed 128 bytes
- Begin with a letter or underscore
- Contain only letters, digits, underscores, or $ characters

The name is not case-sensitive unless you use quotation marks around it. By default, the database server converts uppercase characters in the name to lowercase. If you want to use uppercase in names, put quotation marks around them and set the **DELIMIDENT** environment variable to `ON`.

Related concepts:

"Chunks" on page 8-2

Specify the maximum size of chunks

On most platforms, the maximum chunk size is 4 terabytes, but on other platforms, the maximum chunk size is 8 terabytes.

To determine which chunk size your platform supports see your machine notes file. If you have upgraded from a version before version 10.00 and did not run the **onmode -BC 2** command, the maximum chunk size is 2 GB.

Specify the maximum number of chunks and storage spaces

You can specify a maximum of 32,766 chunks for a storage space, and a maximum of 32,766 storage spaces on the database server system.

The storage spaces can be any combination of dbspaces, blobspaces, and sbspaces.

Considering all limits that can apply to the size of an instance of the database server, the maximum size of an instance is approximately 8 petabytes.

If you have upgraded from a version before version 10.00, you must run **onmode -BC 2** to enable the maximum number of chunks and storage spaces.

Back up after you change the physical schema

You must perform a level-0 backup of the root dbspace and the modified storage spaces to ensure that you can restore the data when you:

- Add or drop mirroring
- Drop a logical-log file
- Change the size or location of the physical log
- Change your storage-manager configuration
- Add, move, or drop a dbspace, blobspace, or sbspace
- Add, move, or drop a chunk to a dbspace, blobspace, or sbspace

Important: When you add a new logical log, you no longer are required to perform a level-0 backup of the root dbspace and modified dbspace to use the new logical log. However, you must perform the level-0 backup to prevent level-1 and level-2 backups from failing.

You must perform a level-0 backup of the modified storage spaces to ensure that you can restore the unlogged data before you switch to a logging table type:

- When you convert a nonlogging database to a logging database
- When you convert a RAW table to standard

Monitor storage spaces

You can monitor the status of storage spaces and configure how you are notified when a storage space becomes full.

When a storage space or partition becomes full, a message is shown in the online message log file.

You can configure alarms that are triggered when storage spaces become full with the STORAGE_FULL_ALARM configuration parameter. You can specify how often alarms are sent and the minimum severity level of alarms to be sent. By default, the alarm interval is 600 seconds and the alarm severity level is 3. For more information about the STORAGE_FULL_ALARM configuration parameters and event alarms, see the *IBM Informix Administrator's Reference*.

If the primary server in a high-availability cluster encounters an out-of-space condition, and the STORAGE_FULL_ALARM configuration parameter is enabled,

the event alarm is triggered and an error status is returned on the primary server but not on any of the secondary servers. This is expected behavior because log records are no longer sent from the primary server to the secondary servers when the primary server encounters an out-of-space condition. In this case, the secondary servers never exceed their storage limits and thus do not trigger an event alarm or return an error status.

You can use the IBM Informix Scheduler to set up a task that automatically monitors the status of storage spaces. The properties of the task define the information that Scheduler collects and specifies how frequently the task runs. For example, you might define a task to monitor storage spaces every hour, five days a week. For more information, see Chapter 27, "The Scheduler," on page 27-1 and "Creating a task" on page 27-9.

Manage dbspaces

This section contains information about creating standard and temporary dbspaces with and without the default page size, specifying the first and next extent sizes for the tblspace **tblspace** in a dbspace when you create the dbspace, and adding a chunk to a dbspace or blobspace.

For information about monitoring dbspaces, see "Monitor storage spaces" on page 9-6.

Related concepts:

"Size of the root dbspace" on page 8-32

Related reference:

➥ The onspaces utility (Administrator's Reference)

Creating a dbspace that uses the default page size

You can use **onspaces** or ON-Monitor to create a standard dbspace and a temporary dbspace.

For information about creating a dbspace with a non-default page size, see "Creating a dbspace with a non-default page size" on page 9-10.

Any newly added dbspace (and its mirror, if one exists) is available immediately. If you are using mirroring, you can mirror the dbspace when you create it. Mirroring takes effect immediately.

To create a standard dbspace using onspaces:

1. On UNIX, you must be logged in as user **informix** or **root** to create a dbspace.
   On Windows, users in the **Informix-Admin** group can create a dbspace.
2. Ensure that the database server is in online, administration, or quiescent mode.
3. Allocate disk space for the dbspace, as described in "Allocate disk space" on page 9-1.
4. To create a dbspace, use the **onspaces -c -d** options.
   KB is the default unit for the **-s** *size* and **-o offset** options. To convert KB to megabytes, multiply the unit by 1024 (for example, 10 MB = 10 * 1024 KB).
   See "Creating a dbspace with a non-default page size" on page 9-10 for information about additional **onspaces** options if you are creating a dbspace with a non-default page size.

5. If you do not want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, go to 6.

   If you want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, see additional information in "Specifying the first and next extent sizes for the tblspace **tblspace**."

6. After you create the dbspace, you must perform a level-0 backup of the root dbspace and the new dbspace.

The following example shows how to create a 10-megabyte mirrored dbspace, **dbspce1**, with an offset of 5000 KB for both the primary and mirror chunks, using raw disk space on UNIX:

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10240 -m /dev/raw_dev2 5000
```

The following example shows how to create a 5-megabyte dbspace, **dbspc3**, with an offset of 200 KB, from raw disk space (drive e:) on Windows:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5120
```

For more information about creating a dbspace with **onspaces**, see "Dbspaces" on page 8-9 and information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

To create a dbspace with ON-Monitor (UNIX):

1. Select the **Dbspaces** > **Create** option.
2. Enter the name of the new dbspace in the field **Dbspace Name**.
3. If you want to create a mirror for the initial dbspace chunk, enter Y in the **Mirror** field. Otherwise, enter N.
4. If the dbspace that you are creating is a temporary dbspace, enter Y in the **Temp** field. Otherwise, enter N.
5. If you are specifying a page size for a standard dbspace, enter the size in KB in the **Page Size** field. The size must be a multiple of the page size of the root dbspace. For more information about specifying page sizes, see "Creating a dbspace with a non-default page size" on page 9-10.

   All tables, indexes, and other objects within the dbspace use pages of the specified size.

6. Enter the full path name for the initial primary chunk of the dbspace in the **Full Pathname** field of the primary-chunk section.
7. Specify an offset in the **Offset** field.
8. Enter the size of the chunk, in KB, in the **Size** field.
9. If you are mirroring this dbspace, enter the full path name, size, and optional offset of the mirror chunk in the mirror-chunk section of the screen.

For more information, see the ON-Monitor topics in the *IBM Informix Administrator's Reference*.

Specifying the first and next extent sizes for the tblspace tblspace

You can specify first and next extent sizes if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you must place the tblspace **tblspace** extents in non-primary chunks. (A *primary chunk* is the initial chunk in a dbspace.)

You can choose to specify the first extent size, the next extent size, both the first and the next extent size, or neither extent size. If you do not specify first or next extent sizes for the tblspace **tblspace**, IBM Informix uses the existing default extent sizes.

You can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the tblspace **tblspace** in the root dbspace that is created when the server is initialized.

You can use the **onspaces** utility to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces.

You can only specify the first and next extent sizes when you create dbspace. You cannot alter the specification of the first and next extent sizes after the creation of the dbspace. In addition, you cannot specify extent sizes for temporary dbspaces, sbspaces, blobspaces, or external spaces. You cannot alter the specification of the first and next extents sizes after the creation of the dbspace.

To specify the first and next extent sizes:
1. Determine the total number of pages required in the tblspace **tblspace**. The number of pages is equal to the sum of the number of tables, detached indexes, and table fragments likely to be located in the dbspace plus one page for the tblspace **tblspace**.
2. Calculate the number of KB required for the number of pages. This number depends on the number of KB to a page on the system.
3. Determine the space management requirements on your system by considering the importance of having all of the extents for the tblspace **tblspace** allocated during dbspace creation and whether the extents must be allocated contiguously. The more important these issues are, the larger the first extent size must be. If you are less concerned with having non-contiguous extents, possibly in secondary chunks, then the first and next extent sizes can be smaller.
4. Specify the extent size as follows:
   - If the space requirement is for the root dbspace, specify the first extent size in the TBLTBLFIRST configuration parameter and the next extent size in the TBLTBLNEXT configuration parameter. Then initialize the database server instance.
   - If the space requirement is for a non-root dbspace, indicate the first and next extent sizes on the command line using the **onspaces** utility to create the dbspace.

Extent sizes must be in KB and must be multiples of the page size. When you specify first and next extent sizes, follow these guidelines:

Type of extent	Minimum size	Maximum size
First extent in a non-root dbspace	The equivalent of 50 pages, specified in KB. This is the system default. For example, for a 2 KB page system, the minimum length is 100.	The size of the initial chunk, minus the space required for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.

Type of extent	Minimum size	Maximum size
First extent in a root dbspace	The equivalent of 250 pages specified in KB. This is the system default.	The size of the initial chunk, minus the space required for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.
Next Extent	Four times the disk-page size on the system. The default is 50 pages on any type of dbspace.	The maximum chunk size minus three pages.

You use the following **onspaces** utility **-ef** and **-en** options to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces:

- First extent size: **-ef** *size_in_kbytes*
- Next extent size: **-en** *size_in_kbytes*

For example, you can specify:

```
onspaces -c -d dbspace1 -p /usr/data/dbspace1 -o 0 -s 1000000 -e 2000 -n 1000
```

You can use **Oncheck -pt** and **oncheck -pT** to show the first and next extent sizes of a tblspace **tblspace**.

If data replication is being used and a dbspace is created on the primary database server, the first and next extent sizes are passed to the secondary database server through the ADDCHK log record.

For more information about the **onspaces** utility, **oncheck** commands, and specifying the first and next extent sizes for the tblspace **tblspace**, see the *IBM Informix Administrator's Reference*.

Related reference:

➡ TBLTBLFIRST configuration parameter (Administrator's Reference)

➡ TBLTBLNEXT configuration parameter (Administrator's Reference)

Creating a dbspace with a non-default page size

You can specify a page size for a standard or temporary dbspace if you want a longer key length than is available for the default page size.

The root dbspace is the default page size. If you want to specify a page size, the size must be an integral multiple of the default page size, and cannot be greater than 16 KB.

For systems with sufficient storage, the performance advantages of a larger page size include:

- Reduced depth of b-tree indexes, even for smaller index keys.
- Decreased checkpoint time, which typically occurs with larger page sizes.

Additional performance advantages occur because you can:

- Group on the same page long rows that currently span multiple pages of the default page size.

- Define a different page size for temporary tables, so the temporary tables have a separate buffer pool.

You can use the BUFFERPOOL configuration parameter to create a buffer pool that corresponds to the page size of the dbspace. (You might want to do this to implement a form of "private buffer pool.")

A table can be in one dbspace and the index for that table can be in another dbspace. The page size for these partitions can be different.

If you want to specify the page size for the dbspace, perform these tasks:
1. If you have upgraded from a version before version 10.00, run the **onmode -BC 2** command to enable the large chunk mode. By default, when IBM Informix is first initialized or restarted, Informix starts with the large chunk mode enabled. For information about the **onmode** utility, see the *IBM Informix Administrator's Reference*.
2. Create a buffer pool that corresponds to the page size of the dbspace. You can use the **onparams** utility or the BUFFERPOOL configuration parameter. You must do this before you create the dbspace.

   If you create a dbspace with a page size that does not have a corresponding buffer pool, Informix automatically creates a buffer pool using the default parameters defined in the `onconfig` configuration file.

   You cannot have multiple buffer pools with the same page size.
3. Define the page size of the dbspace when you create the dbspace. You can use the **onspaces** utility or ON-Monitor. For more information, see "Define the page size" on page 9-15.

For example, if you create a dbspace with a page size of 6 KB, you must create a buffer pool with a size of 6 KB. If you do not specify a page size for the new buffer pool, Informix uses the operating system default page size (4 KB on Windows and 2 KB on most UNIX platforms) as the default page size for the buffer pool.

Tip: If you use non-default page sizes, you might be required to increase the size of your physical log. If you perform many updates to non-default pages you might require a 150 to 200 percent increase of the physical log size. Some experimentation might be required to tune the physical log. You can adjust the size of the physical log as necessary according to how frequently the filling of the physical log triggers checkpoints.

Create a buffer pool for the non-default page size

When you create a buffer pool, you can use the BUFFERPOOL configuration parameter or the **onparams** utility to define information about the buffer pool including its size, the number of LRUS in the buffer pool, the number of buffers in the buffer pool, and **lru_min_dirty** and **lru_max_dirty** values.

You specify this information using the BUFFERPOOL configuration parameter.

The BUFFERPOOL configuration parameter consists of two lines in the `onconfig.std` file.

On UNIX systems, the lines are:
```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=2K,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

On Windows systems, the lines are:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=4K,buffers=10000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

The top line specifies the default values that are used if you create a dbspace with a page size that does not have a corresponding buffer pool that was created when the database server started. The following line after the default line specifies the database server default values for the buffer pool. These values are based on the default page size of the database server.

When you add a dbspace with a different page size with the **onspaces** utility or when you add a new buffer pool with the **onparams** utility, a new line is added to the BUFFERPOOL configuration parameter in the `onconfig` file. The page size for each buffer pool must be a multiple of the default page size for your operating system. The following example shows a third BUFFERPOOL line that was added to the `onconfig` file:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=2K,buffers=50000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
BUFFERPOOL size=6K,buffers=3000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

The fields in the BUFFERPOOL lines are not case sensitive, so you can specify **lrus** or **Lrus** or **LRUS**). These fields can be displayed in any order.

If you do not specify a page size for a new buffer pool, IBM Informix uses the operating system default page size (4 KB on Windows and 2 KB on most UNIX platforms) as the default page size for the buffer pool.

If the value of **buffers** is zero (0) or if the value of **buffers** is missing in the BUFFERPOOL configuration parameter, Informix does not create the buffer pool of the specified page size.

Important: Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters before Version 10.0 is now specified using the BUFFERPOOL configuration parameter. Information you enter using the BUFFERPOOL configuration parameter supersedes any information previously specified with the deprecated parameters. For more information about the deprecated parameters, see the appendix that contains information about discontinued configuration parameters that is in *IBM Informix Administrator's Reference*.

The following table provides an explanation of the values that you specify using the BUFFERPOOL configuration parameter or the **onparams** utility.

Field	Explanation	Range of values
size	Specifies the number of KB, followed by the suffix K, in a page.	The size can vary from 2K or 4K to 16 K. 2K is the default.

Field	Explanation	Range of values
buffers	Specifies the number of buffers for the page size. This is the maximum number of shared-memory buffers that the database server user threads have available for disk I/O on behalf of client applications. The number of buffers that the database server requires depends on the applications. For example, if the database server accesses 15 percent of the application data 90 percent of the time, you must allocate enough buffers to hold that 15 percent. Increasing the number of buffers can improve system performance. The percentage of physical memory that you require for buffer space depends on the amount of memory available on your computer and the amount of memory used for other applications. For systems with a large amount of available physical memory (4 gigabytes or more), buffer space can be as much as 90 percent of physical memory. For systems with smaller amounts of available physical memory, buffer space can range from 20 to 25 percent of physical memory. You must calculate all other shared-memory parameters after you set buffer space (`buffers` *system_page_size*).	For 32-bit platform on UNIX • with page size equal to 2048 bytes: 100 through 1,843,200 buffers (1843200 = 1800 * 1024 • with page size equal to 4096 bytes: 100 through 921,600 buffers (`921,600` = ((1800 * 1024)/4096) * 2048) For 32-bit platform on Windows: 100 through 524,288 buffers (`524,288` = 512 * 1024) For 64-bit platforms: 100 through 2^{31}-1 buffers (For the actual value for your 64-bit platform, see your release notes. The maximum number of buffers on Solaris is 536,870,912.)
lrus	Specifies the number of LRU (least-recently-used) queues in the shared-memory buffer pool for the page size. You can tune the value of LRUS, in combination with the **lru_min_dirty** and **lru_max_dirty** values, to control how frequently the shared-memory buffers are flushed to disk. Setting LRUS too high might result in excessive page-cleaner activity.	1 through 128
lru_min_dirty	Specifies the percentage of modified pages in the LRU queues at which page cleaning is no longer mandatory. Page cleaners might continue cleaning beyond this point under some circumstances. The LRU values for flushing the buffer pool between checkpoints are not critical for checkpoint performance. The **lru_min_dirty** value is usually only necessary for maintaining enough clean pages for page replacement. Start by setting the LRU flushing parameters to set **lru_min_dirty** to 70. For more information, see "LRU values for flushing a buffer pool between checkpoints" on page 15-6.	0 through 100 (fractional values are allowed) If a parameter is specified out of the range of values, then the default of `50.00` percent is set.

Field	Explanation	Range of values
lru_max_dirty	Specifies the percentage of modified pages in the LRU queues at which the queue is cleaned. The LRU values for flushing the buffer pool between checkpoints are not critical for checkpoint performance. The **lru_max_dirty** value is usually only necessary for maintaining enough clean pages for page replacement. Start by setting the LRU flushing parameters to set**lru_max_dirty** to 80. For more information, see "LRU values for flushing a buffer pool between checkpoints" on page 15-6.	0 through 100 (fractional values are allowed) If a parameter is specified out of the range of values, then the default of 60.00 percent is set.

If the database server is in online, quiescent, or administration mode, you can also use the **onparams** utility to add a new buffer pool of a different size. When you use the **onparams** utility, the information you specify is automatically transferred to the onconfig file and new values are specified using the BUFFERPOOL keyword. You cannot change the values by editing the onconfig.std file.

When you use the **onparams** utility, you specify information as follows:

```
onparams -b -g <size of buffer page in Kbytes> -n <number of buffers>
-r <number of LRUs> -x <max dirty (fractional value allowed)>
-m <minimum dirty (fractional value allowed)>
```

For example:

```
onparams -b -g 6 -n 3000 -r 2 -x 2.0 -m 1.0
```

This adds 3000 buffers of size 6K bytes each with 2 LRUS and **lru_max_dirty** set at 2 percent and **lru_min_dirty** set at 1 percent.

For more information about the **onparams** utility, see the *IBM Informix Administrator's Reference*.

Recommendation: Set the PHYSBUFF configuration parameter to at least 128 KB. If the database server is configured to use RTO_SERVER_RESTART, set the PHYSBUFF configuration parameter to at least 512 KB. Setting PHYSBUFF to a lower value might affect transaction performance and or result in a performance warning during server initialization.

The LG_ADDBPOOL log record and the **sysbufpool** system catalog table contain information about each buffer pool.

Buffer pools that are added while the database server is running go into virtual memory, not into resident memory. Only those buffer pool entries that are specified in the onconfig file at startup go into resident memory, depending on the availability of the memory you are using.

Resize an existing buffer pool:
If you must resize an existing buffer pool, you must bring down the database server. Then change the buffer pool size in the onconfig file.

Delete an existing buffer pool:
If you must delete an existing buffer pool, you must bring down the database server. Then, in the onconfig file, delete the buffer pool.

Define the page size

You can use the **onspaces -k** option or ON-Monitor to define the page size.

Set the page size in KB, as follows:

```
onspaces -c -d DBspace [-t] [-k pagesize] -p path -o offset -s size [-m path offset]
```

The root dbspace is the default page size.

If you specify a page size, the page size must be a multiple of the default page size, but not greater than 16 KB.

If you are using ON-Monitor to create a dbspace, be sure to enter the page size in KB in the **Page Size** field.

Improving the performance of cooked-file dbspaces by using direct I/O

On UNIX systems, you can improve the performance of cooked files used for dbspace chunks by using direct I/O.

Direct I/O must be available and the file system must support direct I/0 for the page size used for the dbspace chunk.

You can use IBM Informix to use either raw devices or cooked files for dbspace chunks. In general, cooked files are slower because of the additional overhead and buffering provided by the file system. Direct I/O bypasses the use of the file system buffers, and therefore is more efficient for reads and writes that go to disk. You specify direct I/O with the DIRECT_IO configuration parameter. If your file system supports direct I/O for the page size used for the dbspace chunk and you use direct I/O, performance for cooked files can approach the performance of raw devices used for dbspace chunks.

To improve the performance of cooked-file dbspaces by using direct I/O:
1. Verify that you have direct I/O and the file system supports direct I/O for the page size used for the dbspace chunk.
2. Enable direct I/O by setting the DIRECT_IO configuration parameter to 1.

If you have an AIX operating system, you can also enable concurrent I/O for Informix to use with direct IO when reading and writing to chunks that use cooked files.

For more information about using direct IO or concurrent IO, see the *IBM Informix Performance Guide*.

Storing multiple named fragments in a single dbspace

For fragmented tables that use expression-based, interval, list, or round-robin distribution schemes, you can create named fragments that can be located within a single dbspace.

Storing multiple table or index fragments in a single dbspace improves query performance over storing each fragment in a different dbspace and simplifies management of dbspaces.

Suppose you are creating a fragmented table using an expression-based distribution scheme in which each expression specifies the data sets that are placed

in particular fragments. You might decide to separate the data in the table with data from one month in one dbspace and data from the next 11 months in 11 other dbspaces. However, if you want to use only one dbspace for all the yearly data, you can create named fragments so the data for each month is stored in one dbspace.

If you create a fragmented table with named fragments, each row in the **sysfragments** system catalog contains a fragment name in the **Partition** column. If you create a fragmented table without named fragments, the name of the dbspace is in the **Partition** column. The **Flags** column in the **sysfragments** catalog tells you if the fragmentation scheme has named fragments.

You can create tables and indexes with named fragments, and you can create, drop, and alter named fragments using the PARTITION keyword and the fragment name.

To create a fragmented table with named fragments, use SQL syntax as shown in the following example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbspace1,
     PARTITION part2 (a >=5 AND a < 10) IN dbspace1
        ...
      ;
```

If you created a table or index fragment containing named fragments, you must use syntax containing the fragment name when you use the ALTER FRAGMENT statement, as shown in the following examples:

```
ALTER FRAGMENT ON TABLE tb1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;
```

You can use the PARTITION BY EXPRESSION clause instead of the FRAGMENT BY EXPRESSION clause in CREATE TABLE, CREATE INDEX, and ALTER FRAGMENT ON INDEX statements as shown in this example:

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
            PARTITION part1   (a <= 10) IN idxdbspc1,
            PARTITION part2   (a <= 20) IN idxdbspc1,
            PARTITION part3   (a <= 30) IN idxdbspc1;
```

Use ALTER FRAGMENT syntax to change fragmented tables and indexes that do not have named fragments into tables and indexes that have named fragments. The following syntax shows how you might convert a fragmented table with multiple dbspaces into a fragmented table with named fragments:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1,
     (c1=20) IN dbs2;
ALTER FRAGMENT ON TABLE t1 MODIFY dbs2 TO PARTITION part_3 (c1=20)
 IN dbs1
```

The following syntax shows how you might convert a fragmented index into an index that contains named fragments:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
    (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
    (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
    PARTITION part_3 (c1=30) IN dbs1,
```

See the *IBM Informix Performance Guide* for more information about fragmentation, including fragmentation guidelines, procedures for fragmenting indexes, procedures for creating attached and detached indexes with named fragments, and examples of SQL statements used to create attached and detached indexes containing named fragments.

See the *IBM Informix Guide to SQL: Syntax* for more syntax details, including information about named fragments in GRANT FRAGMENT, REVOKE FRAGMENT statements and details for using the DROP, DETACH and MODIFY clauses of the ALTER FRAGMENT statement.

Creating a temporary dbspace

To specify where to allocate the temporary files, create temporary dbspaces.

To define temporary dbspaces:

1. Use the **onspaces** utility with the **-c -d -t** options.

   For more information, see "Creating a dbspace that uses the default page size" on page 9-7.

2. Use the **DBSPACETEMP** environment variables or the DBSPACETEMP configuration parameter to specify the dbspaces that the database server can use for temporary storage.

   The DBSPACETEMP configuration parameter can contain dbspaces with a non-default page size. Although you can include dbspaces with different page sizes in the parameter list for DBSPACETEMP, the database server only uses dbspaces with the same page size as the first listed dbspace.

   For further information about DBSPACETEMP, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

3. If you create more than one temporary dbspace, the dbspaces must be located on separate disks to optimize the I/O.

If you are creating a temporary dbspace, you must make the database server aware of the existence of the newly created temporary dbspace by setting the **DBSPACETEMP** configuration parameter, the DBSPACETEMP environment variable, or both. The database server does not begin to use the temporary dbspace until you take both of the following steps:

- Set the **DBSPACETEMP** configuration parameter, the **DBSPACETEMP** environment variable, or both.
- Restart the database server.

The following example shows how to create 5-megabyte temporary dbspace named **temp_space** with an offset of 5000 KB:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5120
```

For more information, see "Temporary dbspaces" on page 8-11.

What to do if you run out of disk space

When the initial chunk of the dbspace that you are creating is a cooked file on UNIX or an NTFS file on Windows, the database server verifies that the disk space is sufficient for the initial chunk. If the size of the chunk is greater than the available space on the disk, a message is displayed and no dbspace is created. However, the cooked file that the database server created for the initial chunk is not removed. Its size represents the space left on your file system before you created the dbspace. Remove this file to reclaim the space.

Adding a chunk to a dbspace or blobspace

You add a chunk when a dbspace, blobspace, or sbspace is becoming full or requires more disk space.

Important: The newly added chunk (and its associated mirror, if one exists) is available immediately. If you are adding a chunk to a mirrored storage space, you must also add a mirror chunk.

To add a chunk using **onspaces**:

1. On UNIX, you must be logged in as user **informix** or **root** to add a chunk.

   On Windows, users in the **Informix-Admin** group can add a chunk.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.
3. Allocate disk space for the chunk, as described in "Allocate disk space" on page 9-1.
4. To add a chunk, use the **-a** option of **onspaces**.

   If the storage space is mirrored, you must specify the path name of both a primary chunk and mirror chunk.

   If you specify an incorrect path name, offset, or size, the database server does not create the chunk and displays an error message. Also see "What to do if you run out of disk space."
5. After you create the chunk, you must perform a level-0 backup of the root dbspace and the dbspace, blobspace, or sbspace that contains the chunk.

The following example adds a 10-megabyte mirror chunk to **blobsp3**. An offset of 200 KB for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option.

```
onspaces -a blobsp3 -p /dev/raw_dev1 -o 200 -s 10240  -m /dev/raw_dev2 200
```

The next example adds a 5-megabyte chunk of raw disk space, at an offset of 5200 KB, to dbspace **dbspc3**.

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5120
```

You can also define information that Informix can use to automatically extend the size of a chunk when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space (dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace) will run out of space and when it will run out of space.

Related concepts:

"Automatic space management" on page 9-26

Related tasks:

"Adding a chunk to an sbspace" on page 9-24

Related reference:

"The onstat -B utility" on page 7-11

➠ The onspaces utility (Administrator's Reference)

Adding a chunk with ON-Monitor (UNIX)

You can use ON-Monitor to add a chunk to a dbspace.

To add a chunk to a dbspace, follow these instructions:

1. Choose **Add Chunk** > **Dbspaces** option.
2. Use **RETURN** or the arrow keys to select the blobspace or dbspace that receives the new chunk and press **CTRL-B** or **F3**.
3. The next screen indicates whether the blobspace or dbspace is mirrored. If it is, enter Y in the **Mirror** field.
4. If the dbspace to which you are adding the chunk is a temporary dbspace, enter Y in the **Temp** field.
5. If you indicated that the dbspace or blobspace is mirrored, you must specify both a primary chunk and mirror chunk.

   Enter the complete path name for the new primary chunk in the **Full Pathname** field of the primary-chunk section.
6. Specify an offset in the **Offset** field.
7. Enter the size of the chunk, in KB, in the **Size** field.
8. If you are mirroring this chunk, enter the complete path name, size, and optional offset in the mirror-chunk section of the screen.

Rename dbspaces

You can use the **onspaces** utility to rename a dbspace if you are user **informix** or have DBA privileges and the database server is in quiescent mode (and not any other mode).

To rename a dbspace use the following **onspaces** utility command:

onspaces -ren *old_dbspace_name*-n *new_dbspace_name*

You can rename standard dbspaces and all other spaces, including blobspaces, smart blobspaces, temporary spaces, and external spaces. However, you cannot rename any critical dbspace, such as a root dbspace or a dbspace that contains physical logs.

You can rename a dbspace and an sbspace:

- When Enterprise Replication is enabled
- On a primary database server when data replication is enabled

You cannot rename a dbspace and an sbspace on a secondary database server or when the secondary database server is part of the Enterprise Replication configuration

The rename dbspace operation only changes the dbspace name; it does not reorganize data.

The rename dbspace command updates the dbspace name in all places where that name is stored. This includes reserved pages on disk, system catalogs, the ONCONFIG configuration file, and in-memory data structures.

Important: After renaming a dbspace, perform a level-0 archive of the renamed dbspace and the root dbspace. For information, see the *IBM Informix Backup and Restore Guide*.

Additional actions that may be required after you rename a dbspace

If you rename a dbspace, you must rewrite and recompile any stored procedure code that references the old dbspace name. For example, if you have a stored procedure that contains the ALTER FRAGMENT keywords and a reference to the dbspace name, you must rewrite and recompile that stored procedure.

If you rename dbspaces that are specified in the DATASKIP configuration parameter, you must manually update the DATASKIP configuration parameter after renaming the dbspace.

Manage blobspaces

This section explains how to create a blobspace and determine the blobpage size. The database server stores TEXT and BYTE data in dbspaces or blobspaces, but blobspaces are more efficient. For information about adding a chunk, see "Adding a chunk to a dbspace or blobspace" on page 9-18.

For information about monitoring blobspaces, see "Monitor storage spaces" on page 9-6

Creating a blobspace

You can use **onspaces** or ON-Monitor to create a blobspace.

Before you create a blobspace:
1. Allocate disk space for the blobspace, as described in "Allocate disk space" on page 9-1.
2. Determine what blobpage size is optimal for your environment.

   For instructions, see "Determine blobpage size" on page 9-22.

Specify a blobspace name of up to 128 bytes. The name must be unique and must begin with a letter or underscore. You can use letters, digits, underscores, and $ characters in the name.

Important: You can mirror the blobspace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.

To create a blobspace using **onspaces**:
1. To create a blobspace on UNIX, you must be logged in as user **informix** or **root**.

   To create a blobspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.
3. To add a blobspace, use the **onspaces -c -b** options.

a. Specify an explicit path name for the blobspace. If the blobspace is mirrored, you must specify the path name and size of both the primary chunk and mirror chunk.

b. Use the **-o** option to specify an offset for the blobspace.

c. Use the **-s** option to specify the size of the blobspace chunk, in KB.

d. Use the **-g** option to specify the blobpage size in terms of the number of disk pages per blobpages.

   See "Determine blobpage size" on page 9-22. For example, if your database server instance has a disk-page size of 2 KB, and you want your blobpages to have a size of 10 KB, enter 5 in this field.

   If you specify an incorrect path name, offset, or size, the database server does not create the blobspace and displays an error message. Also see "What to do if you run out of disk space" on page 9-18.

4. After you create the blobspace, you must perform a level-0 backup of the root dbspace and the new blobspace.

The following example shows how to create a 10-megabyte mirrored blobspace, **blobsp3**, with a blobpage size of 10 KB, where the database server page size is 2 KB. An offset of 200 KB for the primary and mirror chunks is specified. The blobspace is created from raw disk space on UNIX.

```
onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10240 -m /dev/raw_dev2 200
```

For reference information about creating a blobspace with onspaces, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

To create a blobspace with ON-Monitor (UNIX):

1. Select the **Dbspaces** > **BLOBSpace** option.

2. Enter the name of the new blobspace in the **BLOBSpace Name** field.

3. If you want to create a mirror for the initial blobspace chunk, enter Y in the **Mirror** field. Otherwise, enter N.

4. Specify the blobpage size in terms of the number of disk pages per blobpage in the **BLOBPage Size** field.

   See "Determine database server page size" on page 9-22. For example, if your database server instance has a disk-page size of 2 KB, and you want your blobpages to have a size of 10 KB, enter 5 in this field.

5. Enter the complete path name for the initial primary chunk of the blobspace in the **Full Pathname** field of the primary-chunk section.

6. Specify an offset in the **Offset** field.

7. Enter the size of the chunk, in KB, in the **Size** field.

8. If you are mirroring this blobspace, enter the full path name, size, and optional offset in the mirror-chunk section of the screen.

Prepare blobspaces to store TEXT and BYTE data

A newly created blobspace is not immediately available for storage of TEXT or BYTE data. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert TEXT and BYTE data into that blobspace be created in separate logical-log files. This requirement is true for all blobspaces, regardless of the logging status of the database. To accommodate this requirement, switch to the next logical-log file after you create a blobspace. (For instructions, see "Back up log files to free blobpages" on page 13-6.)

Determine blobpage size

When you create a blobspace, use the size of the most frequently occurring simple large object as the size of the blobpage. In other words, choose a blobpage size that wastes the least amount of space. For information about calculating an optimal blobpage size, see blobpage size considerations in the topics on the effect of configuration on I/O activity in the *IBM Informix Performance Guide*.

If a table has more than one TEXT or BYTE column, and the objects are not close in size, store each column in a different blobspace, each with an appropriately sized blobpage. See "Tables" on page 8-22.

Determine database server page size

When you specify the blobpage size, you specify it in terms of the database server base page size.

You can use one of the following methods to determine the database server page size for your system:

- Run the **onstat -b** utility to display the system page size, given as buffer size on the last line of the output.
- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.
- **UNIX only**: In ON-Monitor, select either the **Parameters** > **Shared-Memory** or **Parameters** > **Initialize** option to display the system page size.

Obtain blobspace storage statistics

To help you determine the optimal blobpage size for each blobspace, use the following database server utility commands:

- **oncheck -pe**
- **oncheck -pB**

The **oncheck -pe** command provides background information about the objects stored in a blobspace:

- Complete ownership information (displayed as *database:owner.table*) for each table that has data stored in the blobspace chunk
- The total number of pages used by each table to store its associated TEXT and BYTE data
- The total free and total overhead pages in the blobspace

The **oncheck -pB** command lists the following statistics for each table or database:

- The number of blobpages used by the table or database in each blobspace
- The average fullness of the blobpages used by each simple large object stored as part of the table or database

For more information, see "Monitor blobspace usage with oncheck -pe" on page 9-50, "Determine blobpage fullness with oncheck -pB" on page 9-50, and optimizing blobspace blobpage size in the topics about table performance considerations in the *IBM Informix Performance Guide*.

Manage sbspaces

This section describes how to create a standard or temporary sbspace, monitor the metadata and user-data areas, add a chunk to an sbspace, and alter storage characteristics of smart large objects.

For information about monitoring sbspaces, see "Monitor storage spaces" on page 9-6.

Creating an sbspace

Use the **onspaces** utility to create an sbspace.

Use the **onspaces** utility or IBM Informix Server Administrator (ISA) to create an sbspace.

To create an sbspace using **onspaces**:

1. To create an sbspace on UNIX, you must be logged in as user **informix** or **root**.

   To create an sbspace on Windows, you must be a member of the **Informix-Admin** group.

2. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.

3. Use the **onspaces -c -S** options to create the sbspace.

   a. Use the **-p** option to specify the path name, the **-o** option to specify the offset, and the **-s** option to specify the sbspace size.

   b. If you want to mirror the sbspace, use the **-m** option to specify the mirror path and offset.

   c. If you want to use the default storage characteristics for the sbspace, omit the **-Df** option.

      If you want to specify different storage characteristics, use the **-Df** option. For more information, see "Storage characteristics of sbspaces" on page 8-16.

   d. The first chunk in an sbspace must have a metadata area.

      You can specify a metadata area for an sbspace or let the database server calculate the size of the metadata area. For more information, see "Size sbspace metadata" on page 9-24.

4. After you create the sbspace, you must perform a level-0 backup of the root dbspace and the new sbspace.

5. To start storing smart large objects in this sbspace, specify the space name in the SBSPACENAME configuration parameter.

6. Use **onstat -d**, **onstat -g smb s**, and **oncheck -cs**, **-cS**, **-ps**, or **-pS** to display information about the sbspace.

   For more information, see "Monitor sbspaces" on page 9-51.

This shows how to create a 20-megabyte mirrored sbspace, **sbsp4**. Offsets of 500 KB for the primary and 500 KB for the mirror chunks are specified, and a metadata size of 150 KB with a 200 KB offset. The AVG_LO_SIZE **-Df** tag specifies an expected average smart-large-object size of 32 KB.

```
onspaces -c -S sbsp4 -p /dev/rawdev1 -o 500 -s 20480 -m /dev/rawdev2 500
-Ms 150 -Mo 200 -Df "AVG_LO_SIZE=32"
```

For information about creating an sbspace and default options for smart large objects, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*. For information about creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

To create an sbspace using ISA

1. Create the sbspace using ISA. For more information, see the ISA online help.

2. Back up the new sbspace and the root dbspace.

Size sbspace metadata

The first chunk of an sbspace must have a metadata area. When you add smart large objects and chunks to the sbspace, the metadata area grows. In addition, the database server reserves 40 percent of the user area to be used in case the metadata area runs out of space.

It is important to size the metadata area for an sbspace correctly to ensure that the sbspace does not run out of metadata space. You can either:
- Let the database server calculate the size of the metadata area for the new sbspace chunk.
- Specify the size of the metadata area explicitly.

For instructions on estimating the size of the sbspace and metadata area, see table performance considerations in the *IBM Informix Performance Guide*. Also see "Monitoring the metadata and user-data areas" on page 9-55.

Adding a chunk to an sbspace

You can add a chunk to an sbspace or temporary sbspace.

You can specify a metadata area for a chunk, let the database server calculate the metadata area, or use the chunk for user data only.

To add a chunk to an sbspace using **onspaces**:
1. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.
2. Use the **onspaces -a** option to create the sbspace chunk.
   a. Use the **-p** option to specify the path name, the **-o** option to specify the offset, and the **-s** option to specify the chunk size.
   b. If you want to mirror the chunk, use the **-m** option to specify the mirror path and offset.
   c. To specify the size and offset of the metadata space, use the **-Mo** and **-Ms** options.

      The database server allocates the specified amount of metadata area on the new chunk.
   d. To allow the database server to calculate the size of the metadata for the new chunk, omit the **-Mo** and **-Ms** options.

      The database server divides the estimated average size of the smart large objects by the size of the user data area.
   e. To use the chunk for user data only, specify the **-U** option.

      If you use the **-U** option, the database server does not allocate metadata space in this chunk. Instead, the sbspace uses the metadata area in one of the other chunks.
3. After you add a chunk to the sbspace, the database server writes the CHRESERV and CHKADJUP log records.
4. Perform a level-0 backup of the root dbspace and the sbspace.
5. Use **onstat -d** and **oncheck -pe** to monitor the amount of free space in the sbspace chunk.

This example adds a 10-megabyte mirror chunk to **sbsp4**. An offset of 200 KB for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option. The **-U** option specifies that the new chunk contains user data exclusively.

```
onspaces -a sbsp4 -p /dev/rawdev1 -o 200 -s 10240 -m /dev/rawdev2 200 -U
```

You can also define information that Informix can use to automatically expand the size of a chunk when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space (dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace) will run out of space and when it will run out of space.

Related concepts:

"Automatic space management" on page 9-26

Related tasks:

"Adding a chunk to a dbspace or blobspace" on page 9-18

Related reference:

"The onstat -B utility" on page 7-11

"Monitor sbspaces" on page 9-51

Alter storage characteristics of smart large objects

Use the **onspaces -ch** command to change the following default storage characteristics for the sbspace:

- Extent sizes
- Average smart-large-object size
- Buffering mode
- Last-access time
- Lock mode
- Logging

For more information, see "Storage characteristics of sbspaces" on page 8-16 and managing sbspaces in the topics about table performance considerations in your *IBM Informix Performance Guide*.

Creating a temporary sbspace

For background information and the rules for determining where temporary smart large objects are stored, see "Temporary sbspaces" on page 8-19. You can store temporary smart large objects in a standard or temporary sbspace. You can add or drop chunks in a temporary sbspace.

To create a temporary sbspace with a temporary smart large object:

1. Allocate space for the temporary sbspace. For details, see "Allocate disk space" on page 9-1.

   For information about SBSPACETEMP, see the configuration parameters topics in the *IBM Informix Administrator's Reference*.

2. Create the temporary sbspace as the following example shows:

   ```
   onspaces -c -S tempsbsp -t -p ./tempsbsp -o 0 -s 1000
   ```

3. You can specify any of the following **onspaces** options:

   a. Specify a metadata area and offset (**-Ms** and **-Mo**).

   b. Specify storage characteristics (**-Df**).

You cannot turn on logging for a temporary sbspace.

4. Set the SBSPACETEMP configuration parameter to the name of the default temporary sbspace storage area.

   Restart the database server.

5. Use **onstat -d** to display the temporary sbspace.

   For information and an example of **onstat -d** output, see the **onstat** utility in the *IBM Informix Administrator's Reference*.

6. Specify the LO_CREATE_TEMP flag when you create a temporary smart large object.

   Using DataBlade API:

   ```
   mi_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
   ```

   Using Informix ESQL/C:

   ```
   ifx_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
   ```

For information about creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Automatic space management

You can configure the server to add more storage space automatically when more space is required. This enables you to use space more effectively and ensure that space is allocated as necessary, while reducing out-of-space errors and reducing the time required to manually monitor your space and determine which storage space will run out of free space and when it will run out of space. If you configure the server to automatically add space, you can also manually expand a space or extend a chunk.

When the server expands a storage space (dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace), the server can add a chunk to the storage space. If the storage space is a non-mirrored dbspace or a temporary dbspace, the server can also extend a chunk in the storage space.

To configure for the automatic and manual space management, you run SQL administration API commands to perform these tasks:

1. Create, modify, and delete one or more entries in the storage pool. The storage pool contains entries for available raw devices, cooked files, and directories that Informix uses to expand a storage space.

2. Mark a chunk as extendable.

3. Modify the create and extend size of a storage space (optional).

4. Change the threshold and wait time for the automatic addition of more space (optional).

5. Configure the frequency of the monitor low storage task (optional).

If your storage pool contains entries, you can also run SQL administration API commands to:

- Manually expand the storage space or extend a chunk, when you do not want to wait for the task that automatically expands the space to run.

- Manually create storage spaces from storage pool entries and return space from empty storage spaces to the storage pool.

If you do not want to the server to automatically expand space, you can set the SP_AUTOEXPAND configuration parameter to 0 to disable the automatic creation or extension of chunks. You can also specify that a chunk is not extendable.

As an alternative to running SQL administration API commands, you can use the IBM OpenAdmin Tool (OAT) for Informix graphical interface to configure for the automatic and manual space management and to manage storage pool entries.

Related concepts:

"Size of the root dbspace" on page 8-32

"Extendable chunks" on page 8-4

"The storage pool" on page 8-34

Related tasks:

"Adding a chunk to a dbspace or blobspace" on page 9-18

"Adding a chunk to an sbspace" on page 9-24

Related reference:

SP_WAITTIME configuration parameter (Administrator's Reference)

Creating and managing storage pool entries

You can add, modify, and delete the entries in the *storage pool* (a collection of available raw devices, cooked files, or directories that Informix can use if necessary to automatically add space to an existing storage space).

Each entry in the storage pool contains information about a directory, cooked file, or raw device that an Informix instance can use if necessary to automatically expand an existing dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

Creating a storage pool entry

To create a storage pool entry, run the admin() or task() function with the **storagepool add** argument, as follows:

```
EXECUTE FUNCTION task("storagepool add", "path", "begin_offset",
"total_size", "chunk size", "priority");
```

Specify the following information:

- The path for the file, directory, or device that the server can use when additional storage space is required.
- The offset in KB into the device where Informix can begin allocating space.
- The total space available to Informix in this entry. The server can allocate multiple chunks from this amount of space.
- The minimum size in KB of a chunk that can be allocated from the device, file, or directory. The smallest chunk that you can create is 1000 K. Therefore, the minimum chunk size that you can specify is 1000 K.
- A number from 1 to 3 for the priority (1 = high; 2 = medium; 3 = low). The server attempts to allocate space from a high-priority entry before it allocates space from a lower priority entry.

The default units for storage pool sizes and offsets are KB. However, you can specify information in any of the ways shown in the following examples:

- "100000"
- "100000 K"

- "100 MB"
- "100 GB"
- "100 TB"

Modifying a storage pool entry

To modify a storage pool entry, run the admin() or task() function with the
storagepool modify argument, as follows:

```
EXECUTE FUNCTION task("storagepool modify", "storage_pool_entry_id",
"new_total_size", "new_chunk size", "new_priority");
```

Deleting storage pool entries

To delete a storage pool entry, run the admin() or task() function with the
storagepool delete argument, as follows:

```
EXECUTE FUNCTION task("storagepool delete", "storage_pool_entry_id");
```

To delete all storage pool entries, run the admin() or task() function with the
storagepool purge all argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge all");
```

To delete all storage pool entries that are full, run the admin() or task() function
with the **storagepool purge full** argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge full");
```

To delete storage pool entries that have errors, run the admin() or task() function
with the **storagepool purge errors** argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge errors");
```

Examples

The following command adds a directory named /region2/dbspaces with a
beginning offset of 0, a total size of 0, an initial chunk size of 20 megabytes, and a
high priority. In this example the offset of 0 and the total size of 0 are the only
acceptable entries for a directory.

```
EXECUTE FUNCTION task("storagepool add", "/region2/dbspaces", "0", "0",
"20000", "1");
```

The following command changes the total size, chunk size, and priority of storage
pool entry 8 to 10 gigabytes, 10 megabytes, and a medium priority.

```
EXECUTE FUNCTION task("storagepool modify", "8", "10 GB", "10000", "2");
```

The following command deletes the storage pool entry with an entry ID of 7:

```
EXECUTE FUNCTION task("storagepool delete", "7");
```

Related concepts:

"The storage pool" on page 8-34

Related reference:

➡ storagepool purge argument: Delete storage pool entries (SQL administration API) (Administrator's Reference)

➡ storagepool modify argument: Modify a storage pool entry (SQL administration API) (Administrator's Reference)

➡ storagepool delete argument: Delete one storage pool entry (SQL administration API) (Administrator's Reference)

Marking a chunk as extendable or not extendable

You mark a chunk as extendable to enable the automatic or manual extension of the chunk. You can change the mark to not extendable to prevent the automatic or manual extension of the chunk.

If a chunk is marked as not extendable:
- The server cannot automatically extend the chunk when there is little or no free space in the chunk. (However, if the storage pool contains entries, the server can expand a storage space by adding another chunk to the storage space.)
- You cannot manually extend the size of the chunk.

Prerequisite: An extendable chunk must be in an unmirrored dbspace or temporary dbspace.

To mark a chunk as extendable:
1. Run the admin() or task() function with the `modify chunk extendable` argument, as follows:
2. `EXECUTE FUNCTION task("modify chunk extendable", "chunk number");`

To mark a chunk as not extendable:
1. Run the admin() or task() function with the `modify chunk extendable off` argument, as follows:

    `EXECUTE FUNCTION task("modify chunk extendable off", "chunk number");`

The following command specifies that chunk 12 can be extended:

`EXECUTE FUNCTION task("modify chunk extendable", "12");`

Related concepts:

"Size of the root dbspace" on page 8-32

"Extendable chunks" on page 8-4

Related reference:

➡ modify chunk extendable argument: Mark a chunk as extendable (SQL administration API) (Administrator's Reference)

➡ modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API) (Administrator's Reference)

Modifying the create or extend size of a storage space

You can control how a storage pool entry is used by modifying two different dbspace sizes that are associated with expanding a storage space, the create size and the extend size.

To modify the create or extend size of a storage space:

Run the admin() or task() function with the **modify space sp_sizes** argument, as follows:
```
EXECUTE FUNCTION task("modify space sp_sizes", "space_name",
 "new_create_size", "new_extend_size");
```

For the:
- `new_create_size`, specify the minimum size that the server can use when creating a new chunk in the specified dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.
- `new_extend_size`, specify the minimum size that the server can use when extending a chunk in the specified unmirrored dbspace or temporary dbspace.

Specify the size with a number (for the number of KB) or a percentage (for a percentage of the total space).

The following command sets the create size and extend size to 60 megabytes and 10 megabytes, respectively, for a space named `dbspace3`:
```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace3", "60000", "10000");
```

The following command sets the create size and extend size to 20 percent and 1.5 percent, respectively, for a space named `logdbs`:
```
EXECUTE FUNCTION task("modify space sp_sizes", "logdbs", "20", "1.5");
```

Related reference:

⤷ modify space sp_sizes argument: Modify the create or extend size of a storage space (SQL administration API) (Administrator's Reference)

Changing the threshold and wait time for the automatic addition of more space

While Informix can react to out-of-space conditions by automatically extending or adding chunks when a storage space is full, you can also configure the server to extend or add chunks before a storage space is full.

You do this by specifying a threshold for the minimum amount of free KB in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

The threshold you define triggers a task that expands the space.

You can also use the SP_WAITTIME configuration parameter to specify the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.

To change the threshold and wait time:
1. Change the value of the threshold specified in the SP_THRESHOLD configuration parameter from 0 (disabled) to a non-zero value. Specify a value from either 1 to 50 for a percentage of a value from 1000 to the maximum size of a chunk in KB.
2. Change the value of the SP_WAITTIME configuration parameter, which specifies the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.

Related reference:

➡ SP_THRESHOLD configuration parameter (Administrator's Reference)

➡ SP_WAITTIME configuration parameter (Administrator's Reference)

Configuring the frequency of the monitor low storage task

You can change the frequency of the **mon_low_storage** task, which periodically scans the list of dbspaces to find spaces that fall below the threshold indicated by SP_THRESHOLD configuration parameter. If the task finds spaces that below the threshold, the task attempts to expand the space, by extending an extendable chunk or by using the storage pool to add a chunk.

The default frequency of the **mon_low_storage** task is once per hour, but you can configure the task to run more or less frequently

Prerequisite: Specify a value in the SP_THRESHOLD configuration parameter for the minimum amount of free KB in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

To configure the mon_low_storage task to run more or less frequently:

Run the following SQL statements, where *minutes* is the number of minutes between each run:

```
DATABASE sysadmin;
UPDATE ph_task set tk_frequency = INTERVAL (minutes)
 MINUTE TO MINUTE WHERE tk_name = "mon_low_storage";
```

For example, to configure the task to run every 10 minutes, run the following SQL statements:

```
DATABASE sysadmin;
UPDATE ph_task set tk_frequency = INTERVAL (10) MINUTE TO MINUTE
 WHERE tk_name = "mon_low_storage";
```

Manually expanding a space or extending an extendable chunk

You can manually expand a space or extend a chunk when necessary, instead of waiting for Informix to automatically expand the space or extend a chunk.

Prerequisites:
- You can extend a chunk only if it is in an unmirrored dbspace or temporary dbspace.
- The chunk must be marked as extendable before it can be extended. If not, you must run the admin() or task() function with the **modify chunk extendable** argument to specify that the chunk is extendable.
- If a space cannot be expanded by extending a chunk, the storage pool must contain active entries that the server can use to create new chunks.

To immediately increase your storage space:

Either:
- Manually expand a space by running the admin() or task() function with the **modify space expand** argument, as follows:
  ```
  EXECUTE FUNCTION task("modify space expand", "space_name", "size");
  ```

For example, the following command expands space number 8 by 1 gigabyte:

```
EXECUTE FUNCTION task("modify space expand", "8", "1000000");
```

The server expands the space either by extending a chunk in the space or adding a new chunk. The server might round the requested size up, depending on the page size of the storage space and the configured chunk size for any storage pool entry used during the expansion.

- Manually extend a chunk by running the admin() or task() function with the **modify chunk extend** argument, as follows:

```
EXECUTE FUNCTION task("modify chunk extend", "chunk_number", "extend_amount");
```

For example, the following command extends chunk number 12 by 5000 KB:

```
EXECUTE FUNCTION task("modify chunk extend", "12", "5000");
```

The server might round the requested size up, depending on the page size of the storage space.

Related concepts:

"Size of the root dbspace" on page 8-32

"Extendable chunks" on page 8-4

Related reference:

➡ modify space expand argument: Expand the size of a space (SQL administration API) (Administrator's Reference)

➡ modify chunk extend argument: Extend the size of a chunk (SQL administration API) (Administrator's Reference)

Example of minimally configuring for and testing the automatic addition of more space

This example shows how you can minimally configure and then test the automatic addition of more space. You can do this by creating a dbspace, filling the space, adding an entry to the Informix storage pool, and loading tables into the space. When the space fills, Informix automatically expands it.

To minimally configure for and test the automatic addition of more space:

1. Create a dbspace.

   For example, create a dbspace named expandable_dbs and allocate an initial chunk using the first 10000 KB of a cooked file named /my_directory/my_chunk, as follows:

   ```
   onspaces -c -d expandable_dbs -p /my_directory/my_chunk -o 0 -s 10000
   ```

2. Fill the dbspace.

   For example, fill the dbspace without loading a row of data. Instead, create a table and allocate a large set of contiguous free pages to the first extent, as follows:

   ```
   CREATE TABLE large_tab (col1 int) IN expandable_dbs EXTENT SIZE 10000000;
   ```

   You can monitor the free pages in your chunks by using the **onstat -d** command or the IBM OpenAdmin Tool (OAT) for Informix. If your dbspace is full, you receive out-of-space errors when attempting to create and load data into another new table.

3. Add an entry to the Informix storage pool.

   For example, add the $INFORMIXDIR/tmp directory to the storage pool, as follows:

   ```
   DATABASE sysadmin;
   EXECUTE FUNCTION task("storagepool add", "$INFORMIXDIR/tmp",
    "0", "0", "10000", "2");
   ```

4. In the SP_THRESHOLD configuration parameter, set a threshold for the minimum amount of free KB that can exist in a storage space before Informix automatically runs a task to expand the space.

5. Create and load new tables into your database.

   Now, if a storage space becomes full, instead of receiving an out-of-space error, Informix automatically creates a cooked file in the `$INFORMIXDIR/tmp` file and add a chunk to the `expandable_dbs` database using the new cooked file. As you continue to fill this chunk, the server automatically extends it. The server will always extend chunks if possible before adding new ones to a dbspace.

6. Reduce the free space in a storage space to test the value in the SP_THRESHOLD configuration parameter.

   Allocate enough pages in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace to reduce the free space so it is below the threshold indicated by SP_THRESHOLD. However, do not completely fill the space.

   You must see the space automatically expanded the next time that the **mon_low_storage** task runs.

7. Create an out-of-space condition.

   Allocate all pages in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace. Then try to allocate more pages. The allocation must be successful and you must not receive an out-of-space error.

   Informix writes messages to the log whenever it extends or adds a chunk and marks new chunks as extendable.

   Run the **onstat -d** command to display all chunks in the instance. Look for extendable chunks, which are marked with an E flag. The command output shows that the server automatically expanded the space, either through the addition of a new chunk or by extending the size of an existing chunk.

Example of configuring for the automatic addition of more space

This example shows how you can fully configure for the automatic addition of more space by changing some configuration parameter settings, changing the frequency of a task that monitors low storage, and specifying information for extendable spaces and chunks.

To configure for the automatic addition of more storage space:

1. Add entries to the storage pool.

   For example, add the `$INFORMIXDIR/tmp` directory to the storage pool, as follows:

   ```
   DATABASE sysadmin;
   EXECUTE FUNCTION task("storagepool add", "$INFORMIXDIR/tmp",
    "0", "0", "10000", "2");
   ```

2. Mark some chunks in unmirrored dbspaces and temporary dbspaces as extendable so that the server can extend the chunks if necessary in the future.

   For example, specify that chunk 12 can be extended:

   ```
   EXECUTE FUNCTION task("modify chunk extendable", "12");
   ```

   You can also change the mark to of an extendable chunk to not extendable. For example, specify that chunk number 10 cannot be extended:

   ```
   EXECUTE FUNCTION task("modify chunk extendable off", "10");
   ```

3. In the SP_THRESHOLD configuration parameter, set a threshold for the minimum amount of free KB that can exist in a storage space before Informix automatically runs a task to expand the space. Specify either:

- A value from 1 to 50 for a percentage,
- A value from 1000 to the maximum size of the chunk in KB

If an individual storage space fills beyond this threshold that you define and remains that full until the space-monitoring task (**mon_low_storage**) next runs, the server attempts to expand the space by extending an extendable chunk or by using the storage pool to add a chunk.

For example, suppose the SP_THRESHOLD value is 5.5, which the server treats as 5.5 percent. If a space runs low on free pages, and the free space percentage falls below 5.5 percent and remains below that level until the **mon_low_storage** task runs next, that task attempts to expand the space. If SP_THRESHOLD is set to 50000 and a space has fewer than free 50000 KB, that space is expanded the next time **mon_low_storage** runs.

4. Optionally, change how often the **mon_low_storage** task runs. This task periodically scans the list of dbspaces to find spaces that fall below the threshold indicated by SP_THRESHOLD configuration parameter.

   For example, to configure the task to run every 10 minutes, run the following SQL statements:

   ```
   DATABASE sysadmin;
   UPDATE ph_task set tk_frequency = INTERVAL (10) MINUTE TO MINUTE
    WHERE tk_name = "mon_low_storage";
   ```

5. Optionally, change the value of the SP_WAITTIME configuration parameter, which specifies the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.

6. Optionally, change two sizes associated with expanding a storage space:
   - The extend size, which is the minimum size used when extending a chunk in a dbspace or temporary dbspace
   - The create size, which is the minimum size used when creating a new chunk in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace that is not a mirror space

   For example, the following command sets the create size and extend size to 60 megabytes and 10 megabytes, respectively, for space number 3:

   ```
   EXECUTE FUNCTION task("modify dbspace sp_sizes",
    "3", "60000", "10000");
   ```

After you configure for the automatic expansion of a storage space, you can also manually expand the space or extend a chunk in the space, as necessary.

Drop a chunk

Use **onspaces** or IBM Informix Server Administrator to drop a chunk from a dbspace.

Before you drop a chunk, ensure that the database server is in the correct mode, using the following table as a guideline.

Chunk type	Database server in online mode	Database server in administration or quiescent mode	Database server in offline mode
Dbspace chunk	Yes	Yes	No
Temporary dbspace chunk	Yes	Yes	No
Blobspace chunk	No	Yes	No

Chunk type	Database server in online mode	Database server in administration or quiescent mode	Database server in offline mode
Sbspace or temporary sbspace chunk	Yes	Yes	No

Verify whether a chunk is empty

To drop a chunk successfully from a dbspace with either of these utilities, the chunk must not contain any data. All pages other than overhead pages must be freed.

If any pages remain allocated to nonoverhead entities, the utility returns the following error: Chunk is not empty.

In addition, when a dbspace consists of two or more chunks and the additional chunks do not contain user data, the additional chunks cannot be deleted if the chunks contain a tblspace **tblspace**.

If you receive the Chunk is not empty message, you must determine which table or other entity still occupies space in the chunk by running **oncheck -pe** to list contents of the extent.

Usually, the pages can be removed when you drop the table that owns them. Then reenter the utility command.

Drop a chunk from a dbspace with onspaces

The following example drops a chunk from **dbsp3** on UNIX. An offset of 300 KB is specified.

```
onspaces -d dbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of a dbspace with the syntax in the previous example. Instead, you must drop the dbspace. Use the **fchunk** column of **onstat -d** to determine which is the initial chunk of a dbspace. For more information about **onstat**, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

For information about dropping a chunk from a dbspace with **onspaces**, see the *IBM Informix Administrator's Reference*.

Drop a chunk from a blobspace

The procedure for dropping a chunk from a blobspace is identical to the procedure for dropping a chunk from a dbspace described in "Drop a chunk from a dbspace with onspaces" except that the database server must be in quiescent or administration mode. Other than this condition, you must substitute the name of your blobspace wherever a reference to a dbspace occurs.

Drop a chunk from an sbspace with onspaces

The following example drops a chunk from **sbsp3** on UNIX. An offset of 300 KB is specified. The database server must be in online administration, or quiescent mode when you drop a chunk from an sbspace or temporary sbspace.

```
onspaces -d sbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of an sbspace with the syntax in the previous example. Instead, you must drop the sbspace. Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of an sbspace.

The -f (force) option

You can use the **-f** option of **onspaces** to drop an sbspace chunk without metadata allocated in it. If the chunk contains metadata for the sbspace, you must drop the entire sbspace. Use the **Chunks** section of **onstat -d** to determine which sbspace chunks contain metadata.

```
onspaces -d sbsp3 -f
```

Warning: If you force the drop of an sbspace, you might introduce consistency problems between tables and sbspaces.

Delete smart large objects without any pointers

Each smart large object has a reference count, the number of pointers to the smart large object. When the reference count is greater than 0, the database server assumes that the smart large object is in use and does not delete it.

Rarely, a smart large object with a reference count of 0 remains. You can use the **onspaces -cl** command to delete all smart large objects that have a reference count of 0, if it is not open by any application.

For information about using **onspaces -cl**, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Drop a storage space

Use **onspaces** or ON-Monitor to drop a dbspace, temporary dbspace, blobspace, sbspace, temporary sbspace, or extspace.

On UNIX, you must be logged in as root or **informix** to drop a storage space. On Windows, you must be a member of the **Informix-Admin** group to drop a storage space.

You can drop a storage space only when the database server is in online, administration, or quiescent mode.

Preparation for dropping a storage space

Before you drop a dbspace, you must first drop all databases and tables that you previously created in that dbspace. You cannot drop the root dbspace.

Before you drop a blobspace, you must drop all tables that have a TEXT or BYTE column that references the blobspace.

Run **oncheck -pe** to verify that no tables or log files are located in the dbspace or blobspace.

Before you drop an sbspace, you must drop all tables that have a CLOB or BLOB column that reference objects that are stored in the sbspace. For sbspaces, you are not required to delete columns that point to an sbspace, but these columns must be null; that is, all smart large objects must be deallocated from the sbspace.

Tip: If you drop tables on dbspaces where light appends are occurring, the light appends might be slower than you expect. The symptom of this problem is

physical logging activity. If light appends are slower than you expect, make sure that no tables are dropped in the dbspace either before or during the light appends. If you have dropped tables, force a checkpoint with **onmode -c** before you perform the light append.

Important: Dropping a chunk or a dbspace triggers a blocking checkpoint, which forces all database updates to wait while all the buffer pools are flushed to disk. This update blocking can be significantly longer during a blocking checkpoint than during a non-blocking checkpoint, especially if the buffer pool is large.

Drop a mirrored storage space

If you drop a storage space that is mirrored, the mirror spaces are also dropped.

If you want to drop only a storage-space mirror, turn off mirroring. (See "End mirroring" on page 18-6.) This action drops the dbspace, blobspace, or sbspace mirrors and frees the chunks for other uses.

Drop a storage space with onspaces

To drop a storage space with **onspaces**, use the **-d** option as illustrated in the following examples.

This example drops a dbspace called **dbspce5** and its mirrors.
```
onspaces -d dbspce5
```

This example drops a dbspace called **blobsp3** and its mirrors.
```
onspaces -d blobsp3
```

Use the **-d** option with the **-f** option if you want to drop an sbspace that contains data. If you omit the **-f** option, you cannot drop an sbspace that contains data. This example drops an sbspace called **sbspc4** and its mirrors.
```
onspaces -d sbspc4 -f
```

Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the deleted smart large objects.

For information about dropping a storage space with **onspaces**, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Dropping a dbspace or blobspace with ON-Monitor (UNIX)

You can drop a dbspace or blobspace with ON-Monitor.

To drop a dbspace or blobspace with ON-Monitor:
1. Select the **Dbspaces** > **Drop** option.
2. Use **RETURN** or arrow keys to scroll to the dbspace or blobspace that you want to drop.
3. Press **CTRL-B** or **F3**.

You are asked to confirm that you want to drop the dbspace or blobspace.

Back up after dropping a storage space

If you create a storage space with the same name as the deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see the *IBM Informix Backup and Restore Guide*.

Important: After you drop a dbspace, blobspace, or sbspace, the newly freed chunks are available for reassignment to other dbspaces, blobspaces, or sbspaces. However, before you reassign the newly freed chunks, you must perform a level-0 backup of the root dbspace and the modified storage space. If you do not perform this backup, and you subsequently must perform a restore, the restore might fail because the backup reserved pages are not up-to-date.

Creating a space or chunk from the storage pool

If your storage pool contains entries, you can create storage spaces or chunks from free space in the storage pool.

Prerequisite: The storage pool must contain entries (a directory, cooked file, or raw device).

To create a storage space or chunk from the storage pool:

Run the admin() or task() function with one of the following arguments for creating a space from the storage pool. The elements you use in the command vary, depending on the type of space that you are creating.

- `EXECUTE FUNCTION task("create dbspace from storagepool", "space_name", "size", "page_size", "mirroring_flag", "first_extent", "next_extent");`
- `EXECUTE FUNCTION task("create tempdbspace from storagepool", "space_name", "size", "page_size");`
- `EXECUTE FUNCTION task("create blobspace from storagepool", "space_name", "size", "page_size", "mirroring_flag",);`
- `EXECUTE FUNCTION task("create sbspace from storagepool", "space_name", "size", "log_flag", "mirroring_flag",);`
- `EXECUTE FUNCTION task("create tempsbspace from storagepool", "space_name", "size",);`
- `EXECUTE FUNCTION task("create chunk from storagepool", "space_name", "size",);`

Examples

The following command creates a mirrored blobspace named `blobspace1`. The new blobspace has a size of 100 gigabytes and a blobpage size of 100 pages.

```
EXECUTE FUNCTION task("create blobspace from storagepool", "blobspace1", "100 GB",
 "100", "1");
```

The following command adds a chunk to the dbspace named `logdbs`. The new chunk has a size of 200 megabytes.

```
EXECUTE FUNCTION task("create chunk from storagepool", "logdbs", "200 MB");
```

Related reference:

📤 create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API) (Administrator's Reference)

📤 create tempdbspace from storagepool argument: Create a temporary dbspace from the storage pool (SQL administration API) (Administrator's Reference)

📤 create blobspace from storagepool argument: Create a blobspace from the storage pool (SQL administration API) (Administrator's Reference)

📤 create sbspace from storagepool argument: Create an sbspace from the storage pool (SQL administration API) (Administrator's Reference)

📤 create tempsbspace from storagepool argument: Create a temporary sbspace from the storage pool (SQL administration API) (Administrator's Reference)

📤 create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API) (Administrator's Reference)

Returning empty space to the storage pool

You can return the space from an empty chunk or storage space to the storage pool.

To return storage space from an empty chunk, dbspace, temporary dbspace, blobspace, sbspace, or temporary sbspace to the storage pool:

Run the admin() or task() function with one of the following arguments for returning space to the storage pool. The elements you use in the command vary, depending on the type of object that you are dropping.
- EXECUTE FUNCTION task("drop chunk to storagepool", "space_name", "chunk_path", "chunk_offset")
- EXECUTE FUNCTION task("drop dbspace to storagepool", "space_name");
- EXECUTE FUNCTION task("drop tempdbspace to storagepool", "space_name");
- EXECUTE FUNCTION task("drop blobspace to storagepool", "space_name");
- EXECUTE FUNCTION task("drop sbspace to storagepool", "space_name");
- EXECUTE FUNCTION task("drop tempsbspace to storagepool", "space_name");

Examples

The following command drops an empty blobspace named blob4 and adds all of the freed space to the storage pool.
```
EXECUTE FUNCTION task("drop blobspace to storagepool", "blob4");
```

The following command drops an empty chunk in a dbspace named health and adds all of the freed space to the storage pool.
```
EXECUTE FUNCTION task("drop chunk to storagepool", "health",
 "/health/rawdisk23", "100 KB");
```

Related reference:

➡ drop chunk to storagepool argument: Return space from an empty chunk to the storage pool (SQL administration API) (Administrator's Reference)

➡ drop dbspace to storagepool argument: Return space from an empty dbspace to the storage pool (SQL administration API) (Administrator's Reference)

➡ drop tempdbspace to storagepool argument: Return space from an empty temporary dbspace to the storage pool (SQL administration API) (Administrator's Reference)

➡ drop blobspace to storagepool argument: Return space from an empty blobspace to the storage pool (SQL administration API) (Administrator's Reference)

➡ drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API) (Administrator's Reference)

➡ drop tempsbspace to storagepool argument: Return space from an empty temporary sbspace to the storage pool (SQL administration API) (Administrator's Reference)

Manage extspaces

An extspace does not require allocation of disk space. You create and drop extspaces using the **onspaces** utility. For more information about extspaces, see "Extspaces" on page 8-21.

Create an extspace

You create an extspace with the **onspaces** utility. But you must first have a valid data source and a valid access method with which to access that data source. Although you can create an extspace without a valid access method or a valid data source, any attempts to retrieve data from the extspace generate an error. For information about access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.

To create an extspace with **onspaces**, use the -c option as illustrated in the following example. The following example shows how to create an extspace, **pass_space**, that is associated with the UNIX password file.

```
onspaces -c -x pass_space -l /etc/passwd
```

Specify an extspace name of up to 128 bytes. The name must be unique and begin with a letter or underscore. You can use letters, digits, underscores, and **$** characters in the name.

Important: The preceding example assumes that you have coded a routine that provides functions for correctly accessing the file passwd and that the file itself exists. After you have created the extspace, you must use the appropriate commands to allow access to the data in the file passwd. For more information about user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.

For reference information about creating an extspace with **onspaces**, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Drop an extspace

To drop an extspace with **onspaces**, use the **-d** option as illustrated in the following examples. An extspace cannot be dropped if it is associated with an existing table or index.

This example drops an extspace called **pass_space**.

```
onspaces -d pass_space
```

Skip inaccessible fragments

One benefit that fragmentation provides is the ability to skip table fragments that are unavailable during an I/O operation. For example, a query can proceed even when a fragment is located on a chunk that is currently down as a result of a disk failure. When this situation occurs, a disk failure affects only a portion of the data in the fragmented table. By contrast, tables that are not fragmented can become completely inaccessible if they are located on a disk that fails.

This function is controlled as follows:
- By the database server administrator with the DATASKIP configuration parameter
- By individual applications with the SET DATASKIP statement

The DATASKIP configuration parameter

You can set the DATASKIP parameter to OFF, ALL, or ON *dbspace_list.* OFF means that the database server does not skip any fragments. If a fragment is unavailable, the query returns an error. ALL indicates that any unavailable fragment is skipped. ON *dbspace_list* instructs the database server to skip any fragments that are located in the specified dbspaces.

The dataskip feature of onspaces

Use the dataskip feature of the onspaces utility to specify the dbspaces that are to be skipped when they are unavailable. For example, the following command sets the DATASKIP parameter so that the database server skips the fragments in **dbspace1** and **dbspace3**, but not in **dbspace2**:

```
onspaces -f ON dbspace1 dbspace3
```

For the complete syntax of this **onspaces** option, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Use onstat to check dataskip status

Use the **onstat** utility to list the dbspaces currently affected by the dataskip feature. The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of the **onspaces** utility.

When you run **onstat -f**, you receive a message that tells you whether the DATASKIP configuration parameter is set to on for all dbspaces, off for all dbspaces, or on for specific dbspaces.

The SQL statement SET DATASKIP

An application can use the SQL statement SET DATASKIP to control whether a fragment is skipped if it is unavailable. Applications must include this statement only in limited circumstances, because it causes queries to return different results, depending on the availability of the underlying fragments. Like the configuration

parameter DATASKIP, the SET DATASKIP statement accepts a list of dbspaces that indicate to the database server which fragments to skip. For example, suppose that an application programmer included the following statement at the beginning of an application:

```
SET DATASKIP ON dbspace1, dbspace5
```

This statement causes the database server to skip **dbspace1** or **dbspace5** whenever both of these conditions are met:

• The application attempts to access one of the dbspaces.
• The database server finds that one of the dbspaces is unavailable.

If the database server finds that both **dbspace1** and **dbspace5** are unavailable, it skips both dbspaces.

A database server administrator can use the DEFAULT setting for the SET DATASKIP statement to control the dataskip feature. Suppose that an application developer includes the following statement in an application:

```
SET DATASKIP DEFAULT
```

When a query is run subsequent to this SQL statement, the database server checks the value of the configuration parameter DATASKIP. A database server administrator can encourage users to use this setting to specify which dbspaces are to be skipped as soon as the database server administrator becomes aware that one or more dbspaces are unavailable.

Effect of the dataskip feature on transactions

If you turn the dataskip feature on, a SELECT statement always executes. In addition, an INSERT statement always succeeds if the table is fragmented by round-robin and at least one fragment is online. However, the database server does not complete operations that write to the database if a possibility exists that such operations might compromise the integrity of the database. The following operations fail:

• All UPDATE and DELETE operations where the database server cannot eliminate the down fragments

   If the database server can eliminate the down fragments, the update or delete is successful, but this outcome is independent of the DATASKIP setting.

• An INSERT operation for a table fragmented according to an expression-based distribution scheme where the appropriate fragment is down

• Any operation that involves referential constraint checking if the constraint involves data in a down fragment

   For example, if an application deletes a row that has child rows, the child rows must also be available for deletion.

• Any operation that affects an index value (for example, updates to a column that is indexed) where the index in question is located in a down chunk

Determine when to use dataskip

Use this feature sparingly and with caution because the results are always suspect. Consider using it in the following situations:

• You can accept the compromised integrity of transactions.
• You can determine that the integrity of the transaction is not compromised.

The latter task can be difficult and time consuming.

Determine when to skip selected fragments

In certain circumstances, you might want the database server to skip some fragments, but not others. This usually occurs in the following situations:

- Fragments can be skipped because they do not contribute significantly to a query result.
- Certain fragments are down, and you decide that skipping these fragments and returning a limited amount of data is preferable to canceling a query.

When you want to skip fragments, use the ON *dbspace-list* setting to specify a list of dbspaces with the fragments that the database server must skip.

Determine when to skip all fragments

Setting the DATASKIP configuration parameter to ALL causes the database server to skip all unavailable fragments. Use this option with caution. If a dbspace becomes unavailable, all queries initiated by applications that do not issue a SET DATASKIP OFF statement before they execute can be subject to errors.

Monitor fragmentation use

The database administrator might find the following aspects of fragmentation useful to monitor:

- Data distribution over fragments
- I/O request balancing over fragments
- The status of chunks that contain fragments

The administrator can monitor the distribution of data over table fragments. If the goal of fragmentation is improved administration response time, it is important for data to be distributed evenly over the fragments. To monitor fragmentation disk use, you must monitor database server tblspaces, because the unit of disk storage for a fragment is a tblspace. (For information about how to monitor the data distribution for a fragmented table, see "Monitor tblspaces and extents" on page 9-48.)

The administrator must monitor I/O request queues for data that is contained in fragments. When I/O queues become unbalanced, the administrator must work with the DBA to tune the fragmentation strategy. (For an explanation of how to monitor chunk use, including the I/O queues for each chunk, see "Monitor chunks" on page 9-44.)

The administrator must monitor fragments for availability and take appropriate steps when a dbspace that contains one or more fragments fails. For how to determine if a chunk is down, see "Monitor chunks" on page 9-44.

Display databases

You can display the databases that you create with SMI tables or ON-Monitor.

SMI tables

Query the **sysdatabases** table to display a row for each database managed by the database server. For a description of the columns in this table, see the **sysdatabases** information in the topics about the **sysmaster** database in the *IBM Informix Administrator's Reference*.

Using IBM Informix Server Administrator

To query **sysdatabases** using IBM Informix Server Administrator (ISA), follow these steps:

1. Choose **SQL** > **Query**.
2. Select the **sysmaster** data in the **Database** list.
3. Enter the following command and click **Submit**: **select * from sysdatabases;**

Find database status with ON-Monitor (UNIX)

To use ON-Monitor to find the current status of each database, select the **Status** > **Databases** option.

ON-Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in the previous section.

Monitor disk usage

These topics describe methods of tracking the disk space used by various database server storage units.

For background information about internal database server storage units mentioned in this section, see the chapter about disk structures and storage in the *IBM Informix Administrator's Reference*.

Monitor chunks

You can monitor chunks for the following information:

- Chunk size
- Number of free pages
- Tables within the chunk

You can use this information to track the disk space used by chunks, monitor chunk I/O activity, and check for fragmentation.

The onstat -d utility

The **onstat -d** utility lists all dbspaces, blobspaces, and sbspaces and the following information for the chunks within those spaces.

- The address of the chunk
- The chunk number and associated dbspace number
- The offset into the device (in pages)
- The size of the chunk (in pages)
- The number of free pages in the chunk
- The path name of the physical device

If you issue the **onstat -d** command on an instance with blobspace chunks, the number of free pages shown is out of date. The tilde (~) that precedes the `free` value indicates that this number is approximate. The **onstat -d** command does not register a blobpage as available until the logical login which a deletion occurred is backed up and the blobpage is freed. Therefore, if you delete 25 simple large objects and immediately run **onstat -d**, the newly freed space is not in the **onstat** output.

To obtain an accurate number of free blobpages in a blobspace chunk, issue the **onstat -d update** command. For details, see "The onstat -d update option."

In **onstat -d update** output, the **flags** column in the **chunk** section provides the following information:
- Whether the chunk is the primary chunk or the mirror chunk
- Whether the chunk is online, is down, is being recovered, or is a new chunk

For an example of **onstat -d** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Important: You must perform a level-0 backup of the root dbspace and the modified dbspace before mirroring can become active and after turning off mirroring.

The onstat -d update option

The **onstat -d update** option displays the same information as **onstat -d** and an accurate number of free blobpages for each blobspace chunk.

The onstat -D option

The **onstat -D** option displays the same information as **onstat -d**, plus the number of pages read from the chunk (in the **page Rd** field).

Monitor chunk I/O activity with the onstat -g iof command

Use the **onstat -g iof** command to monitor chunk I/O activity, including the distribution of I/O requests against the different fragments of a fragmented table.

The **onstat -g iof** command displays:
- The number of reads from each chunk and the number of writes to each chunk
- I/0 by service level, broken down by individual operation
- The type of operation
- The number of times the operation occurred
- The average time the operation took to complete

If one chunk has a disproportionate amount of I/O activity against it, this chunk might be a system bottleneck.

For an example of **onstat -g iof** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The oncheck -pr command

The database server stores chunk information in the reserved pages PAGE_1PCHUNK and PAGE_2PCHUNK.

To list the contents of the reserve pages, run **oncheck -pr**. The following example shows sample output for **oncheck -pr**. This output is essentially the same as the **onstat -d** output; however, if the chunk information has changed since the last checkpoint, these changes are not in the **oncheck -pr** output.

```
Validating PAGE_1DBSP & PAGE_2DBSP...
        Using dbspace page PAGE_2DBSP.

    DBspace number            1
    DBspace name              rootdbs
    Flags                     0x20001         No mirror chunks
```

```
        Number of chunks              2
        First chunk                   1
        Date/Time created             07/28/2008 14:46:55
        Partition table page number   14
        Logical Log Unique Id         0
        Logical Log Position          0
        Oldest Logical Log Unique Id  0
        Last Logical Log Unique Id    0
        Dbspace archive status        No archives have occurred
.
.
.
Validating PAGE_1PCHUNK & PAGE_2PCHUNK...
        Using primary chunk page PAGE_2PCHUNK.

        Chunk number                  1
        Flags                         0x40      Chunk is online
        Chunk path                    /home/server/root_chunk
        Chunk offset                  0 (p)
        Chunk size                    75000 (p)
        Number of free pages          40502
        DBSpace number                1
.
.
.
```

The oncheck -pe command

To obtain the physical layout of information in the chunk, run **oncheck -pe**. The dbspaces, blobspaces, and sbspaces are listed. The following example shows sample output for **oncheck -pe**.

The following information is displayed:

- The name, owner, and creation date of the dbspace
- The size in pages of the chunk, the number of pages used, and the number of pages free
- A listing of all the tables in the chunk, with the initial page number and the length of the table in pages

The tables within a chunk are listed sequentially. This output is useful for determining chunk fragmentation. If the database server is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

```
DBSpace Usage Report:  rootdbs        Owner:  informix  Created: 08/08/2006

Chunk  Pathname                                 Size      Used      Free
    1  /home/server/root_chunk                 75000     19420     55580

Description                                           Offset      Size
----------------------------------------           --------------------
RESERVED PAGES                                             0        12
CHUNK FREELIST PAGE                                       12         1
rootdbs:'informix'.TBLSpace                              13       250
PHYSICAL LOG                                             263      1000
FREE                                                    1263      1500
LOGICAL LOG: Log file 2                                 2763      1500
LOGICAL LOG: Log file 3                                 4263      1500
...
sysmaster:'informix'.sysdatabases                     10263         4
sysmaster:'informix'.systables                        10267         8
...

Chunk  Pathname                                 Size      Used      Free
    2  /home/server/dbspace1                    5000        53      4947
```

```
Description                                    Offset    Size
----------------------------------------    --------------------
RESERVED PAGES                                      0       2
CHUNK FREELIST PAGE                                 2       1
dbspace1:'informix'.TBLSpace                        3      50
FREE                                               53    4947
```

IBM Informix Server Administrator

You can perform the following tasks using IBM Informix Server Administrator (ISA) commands:

- Check reserved pages.
- Check storage spaces.
- Add dbspaces, temporary dbspaces, blobspaces, temporary sbspaces, and sbspaces.
- Display and add chunks to a storage space.
- Check the dataskip status.
- Display and add external spaces.
- Display the number of pages in your database, percentage of allocated space, and used space.
- Override ONDBSPACEDOWN.

Monitor spaces and chunks with ON-Monitor (UNIX)

You can create and monitor spaces and chunks with ON-Monitor.

You can perform the following tasks using ON-Monitor commands.

Status > Spaces
> Displays status information about storage spaces and chunks

Dbspaces > Create
> Creates a dbspace

Dbspaces > BLOBSpace
> Creates a blobspace

Dbspaces > Mirror
> Adds or drops mirroring for a storage space

Dbspaces > Info
> Displays information about storage spaces

Dbspaces > Add Chunk
> Adds a chunk to a storage space

Dbspaces > dataSkip
> Starts or stops dataskip

Dbspaces > Chunk
> Adds a chunk to a dbspace or blobspace

Dbspaces > Drop
> Drops a dbspace or blobspace

Dbspaces > Status
> Changes the mirror status of a chunk

SMI tables

Query the **syschunks** table to obtain the status of a chunk. The following columns are relevant.

chknum

> Number of the chunk within the dbspace

dbsnum

> Number of the dbspace

chksize

> Total size of the chunk in pages

nfree Number of pages that are free

is_offline

> Whether the chunk is down

is_recovering

> Whether the chunk is recovering

mis_offline

> Whether the mirror chunk is down

mis_recovering

> Whether the mirror chunk is being recovered

The **syschkio** table contains the following columns.

pagesread

> Number of pages read from the chunk

pageswritten

> Number of pages written to the chunk

Monitor tblspaces and extents

Monitor tblspaces and extents to determine disk usage by database, table, or table fragment. Monitoring disk usage by table is particularly important when you are using table fragmentation, and you want to ensure that table data and table index data are distributed appropriately over the fragments.

Run **oncheck -pt** to obtain extent information. The **oncheck -pT** option returns all the information from the **oncheck -pt** option and the additional information about page and index usage.

SMI tables

Query the **systabnames** table to obtain information about each tblspace. The **systabnames** table has columns that indicate the corresponding table, database, and table owner for each tblspace.

Query the **sysextents** table to obtain information about each extent. The **sysextents** table has columns that indicate the database and the table that the extent belongs to, and the physical address and size of the extent.

Monitor simple large objects in a blobspace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

The onstat -O option

The **onstat -O** option displays information about the staging-area blobspace and the Optical Subsystem memory cache. The totals shown in the display accumulate from session to session. The database server resets the totals to 0 only when you run **onstat -z**.

For an example of **onstat -O** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

The first section of the display describes the following system-cache totals information.

Column
> **Description**

size   Size specified in the OPCACHEMAX configuration parameter

alloc   Number of 1 KB pieces that the database server allocated to the cache

avail   Portion of alloc (in KB) not used

number
> Number of simple large objects that the database server successfully put into the cache without overflowing

kbytes
> Number of KB of the simple large objects that the database server put into the cache without overflowing

number
> Number of simple large objects that the database server wrote to the staging-area blobspace

kbytes
> Number of KB of simple large objects that the database server wrote to the staging-area blobspace

Although the **size** output indicates the amount of memory that is specified in the configuration parameter OPCACHEMAX, the database server does not allocate memory to OPCACHEMAX until necessary. Therefore, the **alloc** output reflects only the number of 1 KB pieces of the largest simple large object that has been processed. When the values in the **alloc** and **avail** output are equal, the cache is empty.

The second section of the display describes the following user-cache totals information.

Column
> **Description**

SID   Session ID for the user

user   User ID of the client

size   Size specified in the `INFORMIXOPCACHE` environment variable, if set

> If you do not set the `INFORMIXOPCACHE` environment variable, the database server uses the size that you specify in the configuration parameter OPCACHEMAX.

number
> Number of simple large objects that the database server put into cache without overflowing

kbytes
> Number of KB of simple large objects that the database server put into the cache without overflowing

number

Number of simple large objects that the database server wrote to the staging-area blobspace

kbytes

Size of the simple large objects (in KB) that the database server wrote to the staging-area blobspace

Determine blobpage fullness with oncheck -pB

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

Run **oncheck -pB** with either a database name or a table name as a parameter. The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores_demo** database:

```
oncheck -pB stores_demo:sriram.catalog
```

For detailed information about interpreting the **oncheck -pB** output, see optimizing blobspace blobpage size in the chapter on table performance considerations in the *IBM Informix Performance Guide*.

Monitor blobspace usage with oncheck -pe

The **oncheck -pe** command provides information about blobspace usage:

- Names of the tables that store TEXT and BYTE data, by chunk
- Number of disk pages (not blobpages) used, by table
- Number of free disk pages remaining, by chunk
- Number of overhead pages used, by chunk

The following example shows sample **oncheck -pe** output.

```
BLOBSpace Usage Report:  fstblob          Owner:  informix  Created: 03/01/08
    Chunk: 3   /home/server/blob_chunk                 Size     Used     Free
                                                       4000      304     3696

        Disk usage for Chunk 3                    Total Pages
        -------------------------------------------------------
        OVERHEAD                                           8
        stores_demo:chrisw.catalog                       296
        FREE                                            3696
```

Monitor simple large objects in a dbspace with oncheck -pT

Use **oncheck -pT** to monitor dbspaces to determine the number of dbspace pages that TEXT and BYTE data use.

This command takes a database name or a table name as a parameter. For each table in the database, or for the specified table, the database server displays a general tblspace report.

Following the general report is a detailed breakdown of page use in the extent, by page type. See the **Type** column for information about TEXT and BYTE data.

The database server can store more than one simple large object on the same blobpage. Therefore, you can count the number of pages that store TEXT or BYTE data in the tblspace, but you cannot estimate the number of simple large objects in the table.

The following example shows sample output.

```
TBLSpace Usage Report for mydemo:chrisw.catalog

   Type                 Pages     Empty  Semi-Full      Full  Very-Full
   ----------------  ----------  ----------  ----------  ----------  ----------
   Free                 7
   Bit-Map              1
   Index                2
   Data (Home)          9
   Data (Remainder)     0           0           0           0           0
   Tblspace BLOBs       5           0           0           1           4
                     ----------
   Total Pages          24

   Unused Space Summary

       Unused data bytes in Home pages             3564
       Unused data bytes in Remainder pages           0
       Unused bytes in Tblspace Blob pages         1430

Index Usage Report for index  111_16 on mydemo:chrisw.catalog

                 Average    Average
   Level    Total No. Keys Free Bytes
   -----  -------- -------- ----------
       1         1       74       1058
   -----  -------- -------- ----------
   Total         1       74       1058

Index Usage Report for index  111_18 on mydemo:chrisw.catalog

                 Average    Average
   Level    Total No. Keys Free Bytes
   -----  -------- -------- ----------
       1         1       74        984
   -----  -------- -------- ----------
   Total         1       74        984
```

Monitor sbspaces

One of the most important areas to monitor in an sbspace is the metadata page use. When you create an sbspace, you specify the size of the metadata area. Also, any time that you add a chunk to the sbspace, you can specify that metadata space be added to the chunk.

If you attempt to insert a new smart large object, but no metadata space is available, you receive an error. The administrator must monitor metadata space availability to prevent this situation from occurring.

Use the following commands to monitor sbspaces.

Command	Description
onstat -g smb s	Displays the storage attributes for all sbspaces in the system:
	• sbspace name, flags, owner
	• Logging status
	• Average smart-large-object size
	• First extent size, next extent size, and minimum extent size
	• Maximum I/O access time
	• Lock mode

Command	Description
onstat -g smb c	Displays the following information for each sbspace chunk: • Chunk number and sbspace name • Chunk size and path name • Total user data pages and free user data pages • Location and number of pages in each user-data and metadata areas
oncheck -ce **oncheck -pe**	Displays the following information about sbspace use: • Names of the tables that store smart-large-object data, by chunk • Number of disk pages (not sbpages) used, by table • Number of free user-data pages that remain, by chunk • Number of reserved user-data pages that remain, by chunk • Number of metadata pages used, by chunk The output provides the following totals: • Total number of used pages for all user-data areas and metadata area. The system adds 53 pages for the reserved area to the totals for the user-data area and metadata area. • Number of free pages that remain in the metadata area • Number of free pages that remain in all user-data areas
onstat -d	Displays the following information about the chunks in each sbspace: • Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data areas • Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data areas
oncheck -cs **oncheck -ps**	Validates and displays information about the metadata areas for sbspaces..
oncheck -cS	Displays information about smart-large-object extents and user-data areas for sbspaces.
oncheck -pS	Displays information about smart-large-object extents, user-data areas, and metadata areas for sbspaces. For more information about **oncheck -cS** and **-pS**, see managing sbspaces in the topics on table performance considerations in your *IBM Informix Performance Guide*.

Related tasks:

"Adding a chunk to an sbspace" on page 9-24

"Monitoring the metadata and user-data areas" on page 9-55

Related reference:

⮕ onstat -g smb command: Print sbspaces information (Administrator's Reference)

"The oncheck -ce and oncheck -pe options" on page 9-53

"The onstat -d option"

"The oncheck -ps option" on page 9-54

"The oncheck -cs option" on page 9-54

The onstat -d option

Use the **onstat -d** option to display the following information about the chunks in each sbspace:

• Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data area

• Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data area

For an example of **onstat -d** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

To find out the total amount of used space, run the **oncheck -pe** command. For more information, see "The oncheck -ce and oncheck -pe options."

The **onstat -d** option does not register an sbpage as available until the logical login which a deletion occurred is backed up and the sbpage is freed. Therefore, if you delete 25 smart large objects and immediately run **onstat -d**, the newly freed space is not in the **onstat** output.

Related reference:

"Monitor sbspaces" on page 9-51

The oncheck -ce and oncheck -pe options

Run **oncheck -ce** to display the size of each sbspace chunk, the total amount of used space, and the amount of free space in the user-data area. The **oncheck -pe** option displays the same information as **oncheck -ce** plus a detailed listing of chunk use. First the dbspaces are listed and then the sbspaces. The **-pe** output provides the following information about sbspace use:

- Names of the tables that store smart-large-object data, by chunk
- Number of disk pages (not sbpages) used, by table
- Number of free user-data pages that remain, by chunk
- Number of metadata pages used, by chunk

The output provides the following totals:

- Total number of used pages for the user-data area, metadata area, and reserved area

  The system adds 53 extra pages for the reserved area to the totals for the user-data area and metadata area.
- Number of free pages that remain in the metadata area
- Number of free pages that remain in the user-data area

Tip: The **oncheck -pe** option provides information about sbspace use in terms of database server pages, not sbpages.

The following example shows sample output. In this example, the sbspace **s9_sbspc** has a total of 214 used pages, 60 free pages in the metadata area, and 726 free pages in the user-data area.

```
Chunk Pathname                                       Size      Used      Free
     2 /ix/ids9.2/./s9_sbspc                         1000       940        60

   Description                                       Offset    Size
   ------------------------------------------------- -------- --------
   RESERVED PAGES                                         0         2
   CHUNK FREELIST PAGE                                    2         1
   s9_sbspc:'informix'.TBLSpace                           3        50
   SBLOBSpace LO [2,2,1]                                 53         8
   SBLOBSpace LO [2,2,2]                                 61         1
...
SBLOBSpace LO [2,2,79]                               168         1
   SBLOBSpace FREE USER DATA                            169       305
   s9_sbspc:'informix'.sbspace_desc                     474         4
   s9_sbspc:'informix'.chunk_adjunc                     478         4
   s9_sbspc:'informix'.LO_hdr_partn                     482         8
   s9_sbspc:'informix'.LO_ud_free                       490         5
   s9_sbspc:'informix'.LO_hdr_partn                     495        24
```

```
FREE                                                    519     60
SBLOBSpace FREE USER DATA                               579    421

Total Used:                                    214
Total SBLOBSpace FREE META DATA:                60
Total SBLOBSpace FREE USER DATA:               726
```

You can use CHECK EXTENTS as the SQL administration API *command* equivalent to **oncheck -ce**. For information about using SQL API commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Administrator's Reference*.

Related reference:

"Monitor sbspaces" on page 9-51

The oncheck -cs option

The **oncheck -cs** and the **oncheck -Cs** options validate the metadata area of an sbspace. The following example shows an example of the **-cs** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Use the **oncheck -cs** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area. To find the number of used pages in the metadata area, total the numbers in the **Used** column. To find the number of free pages in the metadata area, total the numbers in the **Free** column.

For example, based on the field values displayed in the following figure, the total number of used pages in the metadata area for **s9_sbspc** is 33 2 KB pages (or 66 KB). The metadata area contains a total of 62 free pages (or 124 KB).

```
Validating space 's9_sbspc' ...

SBLOBspace Metadata Partition                   Partnum    Used    Free
s9_sbspc:'informix'.TBLSpace                    0x200001      6      44
s9_sbspc:'informix'.sbspace_desc                0x200002      2       2
s9_sbspc:'informix'.chunk_adjunc                0x200003      2       2
s9_sbspc:'informix'.LO_hdr_partn                0x200004     21      11
s9_sbspc:'informix'.LO_ud_free                  0x200005      2       3
```

Related reference:

"Monitor sbspaces" on page 9-51

The oncheck -ps option

The **oncheck -ps** option validates and displays information about the metadata areas in sbspace partitions. The following example shows an example of the **-ps** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** validates and displays tblspace information for all storage spaces.

To monitor the amount of free metadata space, run the following command:

```
oncheck -ps spacename
```

The **-ps** output includes information about the locking granularity, **partnum**, number of pages allocated and used, extent size, and number of rows in the metadata area. Use the **oncheck -ps** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area.

If you run **oncheck -ps** for the dbspace that contains the tables where the smart large objects are stored, you can find the number of rows in the table.

```
Validating space 's9_sbspc' ...

TBLSpace Report for
    TBLspace Flags              2801         Page Locking
                                             TBLspace use 4 bit bit-maps
                                             Permanent System TBLspace

    Partition partnum           0x200001
    Number of rows              92
    Number of special columns   0
    Number of keys              0
    Number of extents           1
    Current serial value        1
    First extent size           50
    Next extent size            50
    Number of pages allocated   50
    Number of pages used        6
    Number of data pages        0
    Number of rows              0
    Partition lockid            2097153
    Optical Cluster Partnum     -1
    Current SERIAL8 value       1
    Current REFID value         1
    Created                     Thu May 24 14:14:33 2007
```

Related reference:

"Monitor sbspaces" on page 9-51

Monitoring the metadata and user-data areas

The database server reserves 40 percent of the user-data area as a *reserved area*. The database server uses this reserved space for either the metadata or user data. The metadata area gets used up as smart large objects are added to that sbspace. When the database server runs out of metadata or user-data space, it moves a block of the reserved space to the corresponding area.

When all of the reserve area is used up, the database server cannot move space to the metadata area, even if the user-data area contains free space.

1. As you add smart large objects to the sbspace, use **oncheck -pe** or **onstat -g smb c** to monitor the space in the metadata area, user-data area, and reserved area. For an example, see "The oncheck -ce and oncheck -pe options" on page 9-53.
2. Use the message log to monitor metadata stealing.

    The database server prints messages about the number of pages allocated from the reserved area to the metadata area.
3. Add another chunk to the sbspace before the sbspace runs out of space in the metadata and reserved areas.

    For more information, see "Adding a chunk to an sbspace" on page 9-24.
4. The database server writes the FREE_RE and CHKADJUP log records when it moves space from the reserve area to the metadata or user-data area.

For more information, see "Size sbspace metadata" on page 9-24.

Related reference:

"Monitor sbspaces" on page 9-51

Storage optimization

Data compression and consolidation methods can minimize the disk space used by your data.

The following table describes the you can use to reduce the amount of disk space used by data.

Table 9-1. Storage optimization methods

Storage optimization method	Purpose	When to use
Compressing data	Compresses data in table or fragment rows, reducing the amount of required disk space After you compress data, you can also consolidate the free space that remains in the table or fragment, and return the free space to the dbspace.	When you want to reduce the size of data in a table
Repacking data	Consolidates free space in tables and fragments	After you compress data or separately when you want to consolidate free space in the table or fragment
Shrinking data	Returns free space to the dbspace	After you compress or repack data or separately when you want to return free space to the dbspace
Defragmenting table extents	Brings data rows closer together in contiguous, merged extents	When frequently updated tables become scattered among multiple non-contiguous extents

You can automate any or all of these methods.

You can perform these methods in the IBM OpenAdmin Tool (OAT) for Informix or programmatically with SQL administration API functions.

Automatically optimizing data storage

You can configure the automatic compressing, shrinking, repacking, and defragmenting of tables and extents by updating the **auto_crsd** Scheduler task.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To enable and configure automatic storage optimization:

1. Enable the **auto_crsd** Scheduler task by using an UPDATE statement on the **ph_task** table to set the value of the **tk_enable** column to T. For example, the following statement enables the **auto_crsd** task:

```
UPDATE ph_task
   SET tk_enable = 'T'
   WHERE tk_name = 'auto_crsd';
```

2. Optional: Disable individual operations by using an UPDATE statement on the **ph_threshold** table to set the **value** column for a threshold to F:
   - AUTOCOMPRESS_ENABLED: controls compression
   - AUTOREPACK_ENABLED: controls repacking
   - AUTOSHRINK_ENABLED: controls shrinking
   - AUTODEFRAG_ENABLED: controls defragmenting

For example, the following statement disables just the defragmentation operation of the **auto_crsd** task:

```
UPDATE ph_threshold
  SET value = 'F'
  WHERE name = 'AUTODEFRAG_ENABLED';
```

3. Optional: Change the thresholds of individual operations by using and UPDATE statement on the **ph_threshold** table to change the value of the **value** column for a threshold:

- AUTOCOMPRESS_ROWS: The threshold for compression is the number of uncompressed rows. The default threshold is 10 000 rows.
- AUTOREPACK_SPACE: The threshold for repacking a table is the percentage of noncontiguous space. The default is 90%.
- AUTOSHRINK_UNUSED: The threshold for shrinking a table or fragment is the percentage of unused, allocated space. The default is 50%.
- AUTODEFRAG_EXTENTS: The threshold for defragmenting table or fragment extents is the number of extents. The default is 100.

For example, the following statement changes the compression threshold to 5000 rows:

```
UPDATE ph_threshold
  SET value = '5000'
  WHERE name = 'AUTOCOMPRESS_ROWS';
```

Related concepts:

"Data compression" on page 9-58

"Defragment partitions"

Chapter 27, "The Scheduler," on page 27-1

Related tasks:

"Compressing and uncompressing data" on page 9-62

Related reference:

➥ table or fragment arguments: Compress data and optimize storage (SQL administration API) (Administrator's Reference)

Defragment partitions

You can improve performance by defragmenting partitions to merge non-contiguous extents.

A frequently updated table can become fragmented over time which degrades performance every time the table is accessed by the server. Defragmenting a table brings data rows closer together and avoids partition header page overflow problems. Defragmenting an index brings the entries closer together which improves the speed at which the table information is accessed.

To determine how many extents a table, index, or partition has, you can run the oncheck -pt and pT command.

To defragment a table, index, or partition, run the SQL administration API **task()** or **admin()** function with the `defragment` argument or the `defragment partnum` argument and specify the table name, index, or partition number that you want to defragment.

Limitations and considerations

Before you defragment a partition, review these important considerations:
- You cannot stop a defragment request after the request has been submitted.
- You cannot be defragment the following objects:
  - Pseudo tables, such as Virtual-Table Interface (VTI) tables
  - Tables with Virtual-Index Interface (VII) indexes
  - Tables with B-tree functional indexes
  - Temporary tables
  - Sort files
  - Optical BLOB files
- You must not issue conflicting operations on a table or partition that you want to defragment. The first operation must complete before the second operation is started. If the first operation is still running, the request for the second operation returns an error. The following list contains examples of conflicting operations:
  - Only one defragment request can operate on a partition at a time.
  - Only one defragment request can operate on a dbspace at a time.
  - A table cannot be defragmented when DDL statements, such as DROP TABLE or ALTER FRAGMENT, are being run on the table or partition.
  - A table cannot be defragmented when the table is being truncated.
  - A table cannot be defragmented when the table is being compressed or uncompressed.
  - A table cannot be defragmented when an online index build is being run.
  - A table cannot be defragmented that has exclusive access set on the table.

If there are problems completing a defragment request, error messages are sent to the online log file.

Related tasks:

"Automatically optimizing data storage" on page 9-56

Data compression

You can compress data in tables and table fragments to reduce the amount of disk space. You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Before you compress data, you can estimate the amount of disk space that you can save.

Compressing data, consolidating data, and returning free space have the following benefits:
- Significant savings in disk storage space
- Reduced disk usage for compressed fragments
- Significant saving of logical log usage, which saves additional space and can prevent bottlenecks for high-throughput OLTP after the compression operation is completed.
- Fewer page reads, because more rows can fit on a page
- Smaller buffer pools, because more data fits in the same size pool
- Reduced I/O activity, because:
  - More compressed rows than uncompressed rows fit on a page
  - Log records for insert, update, and delete operations of compressed rows are smaller

- Ability to compress older fragments of time-fragmented data that are not often accessed, while leaving more recent data that is frequently accessed in uncompressed form
- Ability to free space no longer required for a table
- Faster backup and restore

I/O-bound tables, for example, tables that have bad cache hit ratios, are good candidates for compression. In OLTP environments, compressing I/O-bound tables can improve performance.

If your applications run with high buffer cache hit ratios and high performance is more important than space usage, you might not want to compress your data, because compression might slightly decrease performance.

Queries can access data in a compressed table.

Because compressed data covers fewer pages and has more rows per page than uncompressed data, the query optimizer might choose different plans after compression.

If you use Enterprise Replication (ER), compressing data on one replication server does not affect the data on any other replication server.

If you use High-Availability Data Replication (HDR), data that is compressed in the source table is compressed in the target table. You cannot perform compression operations on an HDR secondary, RS secondary, or SD secondary server, because the HDR target server must have the same data and physical layout as the source server.

You cannot use the **onload** and **onunload** utilities to move compressed data from one database to another. You must uncompress data in compressed tables and fragments before using the **onload** and **onunload** utilities.

You can perform compression-related operations with IBM OpenAdmin Tool (OAT) for Informix or by running SQL statements that contain SQL administration API commands with compression parameters.

The main alternative to compression is to buy more physical storage. The main alternative for reducing bottlenecks in IO-bound workloads is to buy more physical memory to enable the expansion of the buffer pools.

Related tasks:
"Automatically optimizing data storage" on page 9-56
"Compressing and uncompressing data" on page 9-62

Data that you can compress
You can compress data in rows and simple large objects in dbspaces. However, you might not want to compress all the types of data that you can compress.

Table or table-fragment data with frequently repeating long patterns is very compressible. Certain types of data, such as text, might be more compressible than other types of data, such as numeric data, because data types like text might contain longer and more frequently repeating patterns.

However, you cannot predict a compression ratio based only on the type of data. For example:

- Text in different languages or character sets might have different compression ratios, even though the text is stored in CHAR or VARCHAR columns.
- Numeric data that consists mostly of zeros might compress well, while more variable numeric data might not compress well.
- Data with long runs of blank spaces compresses well
- Data that has already been compressed using some other algorithm and data that has been encrypted might not compress well. For example, images and sound samples in rows might already be compressed, so compressing the data again would not achieve any additional saving of space.

IBM Informix can compress any combination of data types, because it treats all data to be compressed as unstructured sequences of bytes. Thus, the server can compress patterns that span columns, for example, in city, state, and zip code combinations. (The server uncompresses a sequence of bytes in the same sequence that existed before the data was compressed.)

Compression is applied only to the contents of data rows, including any remainder pieces for rows that span pages, and the images of those rows that are contained in logical log records.

Data that you cannot compress

You cannot compress data in indexes and you cannot compress data in some types of tables and fragments.

You cannot compress data in rows in:
- Tables or fragments in the **sysmaster**, **sysutils**, **sysuser**, **syscdr**, and **syscdcv1** databases
- Catalogs
- Temporary tables
- Virtual-table interface tables
- A tblspace **tblspace** (These are hidden fragments, one per dbspace. Each one holds metadata about all of the fragments in the dbspace.)
- Internal partition tables
- Dictionary tables, one per dbspace (These tables hold compression dictionaries for the fragments or tables that are compressed in that dbspace and metadata about the dictionaries.)
- Indexes
- A table if an online index build is occurring on the table

Compression is not applied to index data, LOB data that is stored outside of the row, or any other form of non-row data.

Encrypted data, data that is already compressed by another algorithm, and data without long repeating patterns compresses poorly or does not compress. Try to avoid placing columns with data that compresses poorly between columns that have frequent patterns to prevent the potential disruption of column-spanning patterns.

If XML data is stored with the first portion in a row and the remainder outside of the row, compression is only applied to the portion that is stored in the row.

IBM Informix compresses images of the rows only if the images of the compressed rows are smaller than the uncompressed images. Even if compressed rows are only

slightly smaller than their uncompressed images, a small saving of space can enable the server to put more rows onto pages.

Compression estimates

Before you compress a table or table fragment, you can estimate the amount of space you can save if data is compressed. The ratios you display are estimates based on samples of row data. The actual ratio of saved space might vary slightly.

IBM Informix estimates the compression ratios by random sampling of row data (using the same sampling algorithm as dictionary building) and then summing up the sizes of the following items:

- Uncompressed row images
- Compressed row images using a new compression dictionary (which is temporarily created by the estimate compression command)
- Compressed row images using the existing dictionary if there is one (If there is no existing dictionary, this value is the same as the sum of sizes of uncompressed row images.)

The actual space saving ratios achieved might vary for these reasons:

- A small sampling error can occur.
- The estimates are based on raw compressibility of the rows.

  For example, the server generally tries to put the entire row onto a single page. So, if each uncompressed row nearly fills a complete page and the compression ratio is less than 50 percent, the compressed rows still fill more than half a page each and the server tends to put each row on a separate page even after compression. In this case, although the estimated compression ratio might be something like 45 percent, the actual space savings might turn out to be 0 percent.

  Uncompressed rows fill slightly more than half a page each. Each uncompressed row consumes a full page because two full rows do not fit. For example, the estimated compression ratio might be something like 5 percent, but this might be just enough to shrink the rows to be less than half a page each. Thus, after compression, two rows would fit on a page and the true space savings might be 50 percent.

The actual compression achieved might also vary from the estimate because Informix can never store more than 255 rows on a single page. Thus, small rows or large pages can reduce the total savings that compression can achieve. For example, if 200 rows fit onto a page before compression, no matter how small the rows are when compressed, the maximum effective compression ratio is approximately 20 percent, because only 255 rows can fit on a page after compression.

If you are using a page size that is larger than the minimum page size, one way to increase the realized compression space savings is to switch to smaller pages, so that:

- The 255 row limit can no longer be reached.
- If this limit is still reached, there is less unused space on the pages.

More (or less) space can be saved, compared to the estimate, if the compress operation is combined with a repack operation, shrink operation, or repack and shrink operation. The repack operation can save additional space only if more compressed rows fit on a page than uncompressed rows. The shrink operation can save space at the dbspace level if the repack operation frees space.

Illustration of compressed data and storage optimization

The illustration in this topic shows uncompressed data that uses most of the space in a fragment, free space that is created when the data is compressed, free space that is moved to the end of the fragment after a repack operation, and data that remains in the fragment after a shrink operation.

Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

1) Uncompressed data uses most of the space in this fragment.

Data	Data	Free space	Data
Free space	Data	Data	Data
Data	Data	Free space	Data
Free space	Data	Data	Data

2) A compress operation reduces the size of rows, creating free space.

Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data
Free space	Free space	Free space	Free space

3) A repack operation moves compressed rows to the front of the fragment, leaving the free space at the end of the fragment.

Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

4) A shrink operation returns free space to the dbspace.

Figure 9-1. Data in a fragment during the compression and storage optimization process

Compressing and uncompressing data

This scenario shows how you can run SQL administration API commands to administer compression and storage optimization.

In this scenario, there is a table named **rock** in a database named **music** owned by user **mario**.

Prerequisites:
- There must be at least 2,000 rows in each fragment of the table, not just a total of 2,000 rows in the table as a whole.
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- You must configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to a compress operation, consume log files faster than one every 30 seconds.
- Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to these compression operations, consume log files faster than one every 30 seconds. Compress, repack, repack_offline, uncompress, and uncompress_offline operations can consume large amounts of logs.

To compress and uncompress row data:

1. You are not sure if you want to compress the **rock** table, so you run the following command to check how much space you might save by compressing the table:

   ```
   EXECUTE FUNCTION task("table estimate_compression", "rock", "music", "mario");
   ```

   You review the resulting report, which indicates you can save 75 percent of the space currently used by the **rock** table. You decide to compress the table.

2. Before you compress data, you want to create a compression dictionary, which contains information that IBM Informix uses to compress data in the **rock** table. You run the following command

   ```
   EXECUTE FUNCTION task("table create_dictionary", "rock", "music", "mario");
   ```

   If you do not create the compression dictionary as a separate step, Informix creates the dictionary automatically when you compress data.

3. You decide that you want to compress data in the **rock** table, consolidate the data, and then return the free space to the dbspace. You run the following command:

   ```
   EXECUTE FUNCTION task("table compress repack shrink", "rock", "music", "mario");
   ```

   After the existing rows are compressed, Informix consolidates the free space that is left at the end of the table, and then removes the free space from the table, returning that space to the dbspace.

4. Now suppose that you must uncompress the data. You run the following command:

   ```
   EXECUTE FUNCTION task("table uncompress", "rock", "music", "mario");
   ```

5. You want to remove the compression dictionary.

   a. Verify Enterprise Replication does not require the dictionary.

      Do not remove compression dictionaries for uncompressed or dropped tables and fragments, if you do require the dictionaries for Enterprise Replication.

   b. Archive the dbspace that contains the table or fragment with a compression dictionary.

   c. Run this command:

      ```
      EXECUTE FUNCTION task("table purge_dictionary", "rock", "music", "mario");
      ```

You compress and uncompress data in table fragments the same way you compress and uncompress data in rows, except that the commands you run have the following format:

```
EXECUTE FUNCTION task("fragment compression_arguments", "partnum_list");
```

If you want to consolidate free space or return free space without compressing or recompressing, you can run a command that tells the server to repack, shrink, or repack and shrink.

Related concepts:

"Data compression" on page 9-58

Related tasks:

"Automatically optimizing data storage" on page 9-56

Related reference:

➡ table or fragment arguments: Compress data and optimize storage (SQL administration API) (Administrator's Reference)

Estimating compression ratios:

You can estimate the percentage of space that you can save if you compress tables or specific or all fragments of fragmented tables. The command displays information that you can use to determine if you want to compress or recompress row data.

Prerequisite: You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.

The command that estimates compression ratios always estimates both a new compression ratio and a current ratio.

For general information about compression ratios and estimates, see "Compression ratios" on page 9-70 and "Compression estimates" on page 9-61.

To estimate the compression benefit:

Run the **admin()** or **task()** function with the `estimate_compression` argument. For example, for a table use this syntax:

```
EXECUTE FUNCTION task("table estimate_compression",
"table_name", "database_name", "owner_name");
```

For a fragment, use this syntax:

```
EXECUTE FUNCTION task("fragment estimate_compression",
"partnum_list");
```

The following example shows a command that tells IBM Informix to estimate the benefit of compressing a table named "cash_transaction" in a "store123" database in which "wong" is the owner.

```
EXECUTE FUNCTION task("table estimate_compression", "cash_transaction",
 "store123", "wong");
```

The estimate_compression operation displays the estimated compression ratio that can be achieved, the current compression ratio, an estimate of the percentage gain or loss, the partition number of each fragment, and the full name of the table, including the database, owner, and table names. The current ratio is 0.0 percent if the table is not compressed.

In the following example, the first fragment is already compressed. The second fragment is not compressed. If you recompress the first fragment, a .4 percent increase in saved space can occur. If you compress the second fragment, a 75.7 percent increase can occur.

```
  est    curr  change     partnum      table
  -----  ----- ------  -------------  -----------------------------------
  75.7% 75.3%   +0.4    0x00200003   store123:wong.cash_transaction
  75.7%  0.0%  +75.7    0x00300002   store123:wong.cash_transaction
```

Output from compression estimates for tables and fragments looks the same, except that the output for a table always shows all fragments in the table, while the output for a fragment only shows information for the specified fragments.

Related reference:

⬛➡ Output of the estimate compression operation (SQL administration API) (Administrator's Reference)

Creating a compression dictionary:

You can create a compression dictionary based on existing rows for IBM Informix to use when compressing data in tables or table fragments. After you create the dictionary, Informix uses the dictionary to compress newly inserted or updated rows.

If a compression dictionary does not exist, you can also create one when you run the compress command. The only difference between the two commands is that the compress command also compresses existing data in the table or fragment.

For general information about compression dictionaries, see "Compression dictionaries" on page 9-71.

Prerequisites:
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- If you want to create a compression dictionary for a fragment, the fragment must contain at least 2,000 rows. If you want to create a compression dictionary for a table, each fragment of the table must contain at least 2,000 rows.

To create a compression dictionary:

Run the **admin()** or **task()** function with the **table create_dictionary** or **fragment create_dictionary** arguments.
For example:
- For a table, specify information as follows:
  ```
  EXECUTE FUNCTION task("table create_dictionary", "table_name",
  "database_name", "owner_name");
  ```
  The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Informix uses the current database and owner name.
- For a fragment, specify information as follows:
  ```
  EXECUTE FUNCTION task("fragment create_dictionary", "partnum_list");
  ```
  The *partnum_list* is a space-separated list of partition numbers.

The following example shows a command that tells Informix to create a compression dictionary for a table named "classical" in a "music" database in which "shakar" is the owner.
```
EXECUTE FUNCTION task("table create_dictionary","classical","music","shakar");
```

To compress data in existing table or fragment rows after you create the compression dictionary, you must run a compress command.

You can delete a compression dictionary only after you uncompress the table or fragment.

Consolidating free space in tables:

You can consolidate (repack) free space in tables and fragments when you compress the tables or fragments or separately without compressing.

You can perform a repack operation online or offline, using the **repack** or **repack_offline** argument. The repack_offline operation is the same as the repack

operation, except that IBM Informix performs the operation while holding an exclusive lock on the table or fragment. This operation prevents all other access to data until the operation is completed.

If light appends occur in a table or fragment while a repack operation is occurring, the repack operation does not complete the consolidation of space at the end of a table or fragment. The repack operation does not complete because the new extents are added in the location where the repack operation has already occurred, so space cannot be returned to the dbspace. To complete the repack process, you must run a second repack operation after light append activity has completed. This second repack operation builds on the work of the first repack operation.

Dropping or disabling indexes before you compete a repack_offline operation can decrease the amount of time that it takes the server to complete the operation. afterward, you can recreate or reenable the indexes, preferably taking advantage of PDQ. Dropping or disabling the indexes and then creating or enabling them again can be faster than completing a repack_offline operation without doing this.

Prerequisites:
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to a repack or repack_offline operation, consume log files faster than one every 30 seconds.

To consolidate free space in a table:

1. Run the **admin()** or **task()** function with the **table repack**, **table repack_offline**, **fragment repack** or **fragment repack_offline** command arguments.

   For example, for a table specify:

   ```
   EXECUTE FUNCTION task("table repack", "table_name",
   "database_name", "owner_name");
   ```

   The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Informix uses the current database and owner name.

   For example, for a fragment specify:

   ```
   EXECUTE FUNCTION task("fragment repack_offline", "partnum_list");
   ```

   The *partnum_list* is a space-separated list of partition numbers that belong to the same table.

2. Optionally expand the arguments to include **compress** and **shrink** in any of the following combinations:
   - **compress repack**
   - **compress repack shrink**
   - **repack shrink**

The following example shows a command that tells Informix to consolidate free space in a table named "opera" in a "music" database in which "bob" is the owner.

```
EXECUTE FUNCTION task("table repack","opera","music","bob");
```

The following example shows a command that tells Informix to consolidate free space offline in a table named "folk" in a "music" database in which "janna" is the owner.

```
EXECUTE FUNCTION task("table repack_offline","folk","music","janna");
```

The following example shows a command that tells the Informix to consolidate free space and return the space to the dbspace in a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment repack shrink", "14680071");
```

You can cancel a command with a **compress** argument, for example, by typing CTRL-C in DB-Access. You can reissue commands with **repack** or **repack_offline** arguments after a prior interrupted command. (Compress and repack operations are logged, but run in small portions.)

Returning free space to the dbspace:

You can return free space to the dbspace (shrink the space) when you compress, repack, or compress and repack tables or fragments; or you can return free space separately without compressing or repacking. Returning free space reduces the total size of the fragment or table.

You can safely shrink the entire table without compromising the allocation strategy of the table. For example, if you have a fragmented table with one fragment for each day of the week and many fragments pre-allocated for future use, you can shrink the table without compromising this allocation strategy. If the table is empty, IBM Informix shrinks the table to the initial extent size specified when the table was created.

When you initiate a shrink operation, Informix shortens extents as follows:
- It shortens all extents except the first extent to as small a size as possible.
- If the table is entirely in the first extent (for example, because the table is an empty table), Informix does not shrink the first extent to a size that was smaller than the extent size that was specified when the table was created with the CREATE TABLE statement.

You can use the MODIFY EXTENT SIZE clause of the ALTER TABLE statement to reduce the current extent size. After you do this, you can rerun the shrink operation to shrink the first extent to the new extent size.

Prerequisite: You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.

To return free space to the dbspace:
1. Run the **admin()** or **task()** function with the **table shrink** or **fragment shrink** arguments.

    For example, for a table specify:
    ```
    EXECUTE FUNCTION admin("table shrink", "table_name",
    "database_name", "owner_name");
    ```
    The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, Informix uses the current database and owner name.

    For example, for a fragment specify:
    ```
    EXECUTE FUNCTION task("fragment shrink", "partnum_list");
    ```
    The *partnum_list* is a space-separated list of partition numbers that belong to the same table.
2. Optionally expand the arguments to include **compress** and **repack** in any of the following combinations:

- **compress repack shrink**
- **compress shrink**
- **repack shrink**

The following example shows a command that tells Informix to shrink a table named "opera" in a "music" database in which "bob" is the owner.

```
EXECUTE FUNCTION task("table shrink","opera","music","bob");
```

The following example shows a command that tells the Informix to repack and shrink a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment repack shrink," "14680071");
```

Uncompressing data:

You can uncompress previously compressed tables and fragments. Uncompressing a table or fragment deactivates compression for new insert and update operations, uncompresses all compressed rows, deactivates the compression dictionary, and allocates new pages for rows that no longer fit on their original pages.

You can uncompress online or offline, using the **uncompress** or **uncompress_offline** argument. An uncompress_offline operation is the same as the uncompress operation, except that this operation is performed while holding an exclusive lock on the fragment, preventing all other access to the fragment data until the operation is completed

Dropping or disabling indexes before you compete an uncompress_offline operation can decrease the amount of time that it takes the server to complete the operation. afterward, you can recreate or reenable the indexes, preferably taking advantage of PDQ. Dropping or disabling the indexes and then creating or enabling them again can be faster than completing a repack_offline or uncompress_offline operation without doing this.

Prerequisites:
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- The table or fragments must be compressed.
- Configure your logs to be larger than they currently are, if any workload that you expect to run, including but not limited to an uncompress or uncompress_offline operation, consume log files faster than one every 30 seconds.

To uncompress data in a table or fragment:

Run the **admin()** or **task()** function with the **table uncompress**, **table uncompress_offline**, **fragment uncompress**, or **fragment uncompress_offline** command arguments.
For a table, specify information as follows:

```
EXECUTE FUNCTION task("table uncompress", "table_name",
"database_name", "owner_name");
```

or

```
EXECUTE FUNCTION admin("table uncompress_offline",
"table_name", "database_name", "owner_name");
```

The table name is mandatory. The database and owner names are optional. If you do not specify a database or owner name, IBM Informix uses the current database and owner name.

For a fragment, specify information as follows:

```
EXECUTE FUNCTION task("fragment uncompress", "partnum_list");
```

or

```
EXECUTE FUNCTION task("fragment uncompress_offline", "partnum_list");
```

Examples

The following example shows a command that tells Informix to uncompress a table named "rock" in a "music" database in which "mario" is the owner.

```
EXECUTE FUNCTION task("table uncompress","rock","music","mario");
```

The following example shows a command that tells the Informix to uncompress offline a fragment with a partition number of 14680071.

```
EXECUTE FUNCTION task("fragment uncompress_offline," "14680071");
```

If a table is uncompressed, Informix marks the dictionary for that table as inactive. Informix does not delete dictionaries, because Enterprise Replication functions use the dictionaries for older logs. You can delete the dictionaries that you no longer require.

You can cancel a command with an **uncompress** argument, for example, by typing CTRL-C in DB-Access.

You can reissue commands with **uncompress** and **uncompress_offline** arguments after a prior interrupted command. (Compress, repack, and uncompress operations are logged, but run in small portions.)

Deleting compression dictionaries:

You can delete an inactive compression dictionary for a specific table or fragment, you can delete all inactive compression dictionaries, or you can delete all inactive compression dictionaries up to a specified date. You must uncompress tables and fragments, which makes the dictionaries inactive, before you delete any compression dictionaries that were created for the tables and fragments.

Do not remove compression dictionaries that Enterprise Replication requires.

Prerequisites:
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Uncompress or drop the table or fragment before deleting the associated dictionary. You only can delete a compression dictionary that is no longer used by a compressed table or fragment.
- Be sure that Enterprise Replication functions are not using the compression dictionary for older logs.
- Archive any dbspace that contains a table or fragment with a compression dictionary, even if you uncompressed data in the table or fragment and the dictionary is no longer active.

To delete one or more compression dictionaries that you no longer require:

Run the **admin()** or **task()** function with:

- The `table purge_dictionary` or `fragment purge_dictionary` command arguments to delete a specific, inactive dictionary.
- The `compression purge_dictionary` command arguments to delete all dictionaries.
- The `compression purge_dictionary` command arguments and a date to delete all dictionaries that were created before and on the date.

  You can use any date in a format that can be converted to a DATE data type based on your locale and environment. For example, you can specify `01/31/2012`, `01/31/12`, or `Jan 31, 2012`.

Example 1: Run the following command to remove the inactive dictionary for a table named "latin" in a "music" database in which "arlette" is the owner.

```
EXECUTE FUNCTION task("table purge_dictionary",
"latin", "music", "arlette");
```

Example 2: Run the following command to remove all dictionaries that were created before and on March 8, 2011:

```
EXECUTE FUNCTION task("compression purge_dictionary", "03/08/11");
```

When you compress a table, the database server creates an internal partition that holds the compression dictionary. When you uncompress a table and purge the dictionary, the database server removes the dictionary. However, the internal partition is not removed. If the internal partition that was created for holding the dictionary resides in the second chunk, you might have a chunk that contains only the internal partition. If this occurs, you will not be able to drop the chunk. You can back up the dbspace and then drop the dbspace.

Move compressed data:

You can use the High-Performance Loader (HPL) or any of the other IBM Informix utilities, except the **onunload** and **onload** utilities, to move compressed data.

You cannot use the **onunload** and **onload** utilities to move compressed data from one database to another. You must uncompress data in compressed tables and fragments before using the **onunload** and **onload** utilities.

The **dbexport** utility uncompresses compressed data. Therefore, if your database contains tables or fragments with compressed data, you must recompress after you use the **dbimport** utility to import the data.

For details on using HPL or the Informix utilities, see the *IBM Informix High-Performance Loader User's Guide* or the *IBM Informix Migration Guide*.

Compression ratios

The compression ratio depends on the data being compressed. The compression algorithm that IBM Informix uses is a dictionary-based algorithm that performs operations on the patterns of the data that were found to be the most frequent, weighted by length, in the data that was sampled at the time the dictionary was built.

If the typical data distribution skews away from the data that was sampled when the dictionary was created, compression ratios can decrease.

The maximum possible compression ratio is 90 percent. This is because the maximum possible compression of any sequence of bytes occurs by replacing each group of 15 bytes with a single 12-bit symbol number, yielding a compressed image that is ten percent of the size of the original image. However, the 90 percent ratio is never quite achieved, because Informix adds a single byte of metadata to each compressed image.

Compression dictionaries

A separate compression dictionary exists for each compressed fragment and each compressed non-fragmented table. Each compression dictionary is a library of frequently occurring patterns in the fragment or table data and the symbol numbers that replace the patterns.

A compression dictionary is built using data that is sampled randomly from a fragment or non-fragmented table that contains at least 2,000 rows. If the fragment or table does not contain 2,000 rows, IBM Informix does not build a compression dictionary.

The compression dictionary can store a maximum of 3,840 patterns, each of which can be from two to 15 bytes in length. (Patterns that are longer than seven bytes reduce the total number of patterns that the dictionary can hold.) Each of these patterns is represented by a 12-bit symbol number in a compressed row. To be compressed, a sequence of bytes in the input row image must exactly match a complete pattern in the dictionary. A row that does not have enough pattern matches against the dictionary might not be compressible, because each byte of an input row that did not completely match is replaced in the compressed image by 12 bits (1.5 bytes).

Informix attempts to capture the best compressible patterns (the frequency of the pattern multiplied by the length). Data is compressed by replacing occurrences of the patterns with the corresponding symbol numbers from the dictionary, and replacing occurrences of bytes that do not match any pattern with special reserved symbol numbers.

All dictionaries for the tables or fragments in a dbspace are stored in a hidden dictionary table in that dbspace. The **syscompdicts_full** table and the **syscompdicts** view in the **sysmaster** database provide information about the compression dictionaries.

Typically, approximately 100 KB of space is required for storing the compression dictionary for a compressed fragment or table. Thus, very small tables are not good candidates for compression, because you might not be able to gain back enough space from compressing the rows to offset the storage cost of the compression dictionary.

Additionally, Informix cannot compress an individual row to be smaller than four bytes long. This is because the server must leave room in case the row image later grows beyond what the page can hold. Therefore, you must not try to compress fragments or non-fragmented tables with rows that contain four bytes or are shorter than four bytes.

Compression information that you can view

You can use Informix utilities, a **sysmaster** database table, and a **sysmaster** view to display compression statistics, information about compression dictionaries, and the compression dictionary.

Table 9-2. Utilities and the **sysmaster** *Table and View that Show Compression Information*

Utilities, Table, or View	Description
oncheck -pT option	Displays statistics on any compressed items in the "Compressed Data Summary" section of the output. If no items are compressed, the "Compressed Data Summary" section does not appear in the output. For example, for row data, **oncheck -pT** displays the number of any compressed rows in a table or table fragment and the percentage of table or table-fragment rows that are compressed.
onlog -c option	Uses the compression dictionary to expand compressed data and display the uncompressed contents of compressed log records.
onstat –g dsk option	Displays information that shows the progress of currently running compression operations.
onstat -g ppd option	Displays information on the active compression dictionaries that exist for currently open compressed fragments (also referred to as partitions). This option shows the same information as the **syscompdicts** view in the **sysmaster** database.
syscompdicts_full table in the **sysmaster** database	Displays metadata about the compression dictionary and the compression dictionary binary object. Only user **informix** can access this table.
syscompdicts view in the **sysmaster** database	Displays the same information as the **syscompdicts_full** table, except that for security reasons, it excludes the **dict_dictionary** column, which contains the compression dictionary binary object.

You can use an UNLOAD statement to unload the compression dictionary from the **syscompdicts_full** table to the compression dictionary file, as follows:

```
UNLOAD TO 'compression_dictionary_file'
   SELECT * FROM sysmaster:syscompdicts_full;
```

For more information about the **onlog** Utility, the **onstat -g dsk** option, the **onstat -g ppd** option, the **oncheck -pT** option, the **syscompdicts_full** table, and the **syscompdicts** view, see the *IBM Informix Administrator's Reference*.

Related concepts:

The onlog Utility (Administrator's Reference)

Related reference:

onstat -g dsk command: Print the progress of the currently running compression operation (Administrator's Reference)

onstat -g ppd command: Print partition compression dictionary information (Administrator's Reference)

oncheck -pt and -pT: Display tblspaces for a Table or Fragment (Administrator's Reference)

syscompdicts_full (Administrator's Reference)

Load data into a table

You can load data into an existing table in the following ways.

Method to load data	TEXT or BYTE data	CLOB or BLOB data	Reference
DB-Access LOAD statement	Yes	Yes	LOAD statement in the *IBM Informix Guide to SQL: Syntax*
dbload utility	Yes	Yes	*IBM Informix Migration Guide*
dbimport utility	Yes	Yes	*IBM Informix Migration Guide*
Informix ESQL/C programs	Yes	Yes	*IBM Informix ESQL/C Programmer's Manual*
Insert MERGE, using an EXTERNAL source table	Yes	Yes	*IBM Informix Guide to SQL: Syntax*
onload utility	No	No	*IBM Informix Migration Guide*
onpladm utility	Yes, deluxe mode	Yes, deluxe mode	IBM Informix Server Administrator
High-Performance Loader (HPL)	Yes, deluxe mode	Yes, deluxe mode	*IBM Informix High-Performance Loader User's Guide*

Important: The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been loaded into a database.

Chapter 10. Moving data with external tables

You can use external tables to load and unload database data.

You issue a series of SQL statements that perform the following functions:
- Transfer operational data efficiently to or from other systems
- Transfer data files across platforms in IBM Informix internal data format
- Use the database server to convert data between delimited ASCII, fixed-ASCII, and IBM Informix internal (raw) representation
- Use SQL INSERT and SELECT statements to specify the mapping of data to new columns in a database table
- Provide parallel standard INSERT operations so that data can be loaded without dropping indexes
- Use named pipes to support loading data to and unloading data from storage devices, including tape drives and direct network connections
- Maintain a record of load and unload statistics during the run
- Perform express (high-speed) and deluxe (data-checking) transfers

You can issue the SQL statements with DB-Access or embed them in an ESQL/C program.

External tables

An *external table* is a data file that is not managed by an IBM Informix database server. The definition of the external table includes data-formatting type, external data description fields, and global parameters.

To map external data to internal data, the database server views the external data as an external table. Treating the external data as a table provides a powerful method for moving data into or out of the database and for specifying transformations of the data.

When the database server runs a load task, it reads data from the external source and performs the conversion required to create the row and then inserts the row into the table. The database server writes errors to a reject file.

If the data in the external table cannot be converted, you can specify that the database server write the record to a *reject file*, along with the reason for the failure. To do this, you specify the REJECTFILE keyword in the CREATE EXTERNAL TABLE statement.

The database server provides a number of different conversion mechanisms, which are performed within the database server and therefore provide maximum performance during the conversion task. The database server optimizes data conversion between ASCII and IBM Informix data representations, in both fixed and delimited formats.

To perform customized conversions, you can create a filter program that writes converted data to a named pipe. The database server then reads its input from the named pipe in one of the common formats.

Defining external tables

To define an external table, you use SQL statements to describe the data file, define the table, and then specify the data to load or unload.

To set up loading and unloading tasks, you issue a series of SQL statements:
- CREATE EXTERNAL TABLE to describe the data file to load or unload
- CREATE TABLE to define the table to load
- INSERT...SELECT to load and unload

The following steps outline the load process:

1. The CREATE EXTERNAL TABLE statement describes the location of the various external files, which can be on disk or come from a pipe (tape drive or direct network connection), and the format of the external data. The following example is a CREATE EXTERNAL TABLE statement:

```
CREATE EXTERNAL TABLE emp_ext
   ( name CHAR(18) EXTERNAL CHAR(18),
     hiredate DATE EXTERNAL CHAR(10),
     address VARCHAR(40) EXTERNAL CHAR(40),
     empno INTEGER EXTERNAL CHAR(6) )
USING (
   FORMAT 'FIXED',
    DATAFILES
      ("DISK:/work2/mydir/emp.fix")
    );
```

2. The CREATE TABLE statement defines the table to load. The following sample CREATE TABLE statement defines the **employee** table:

```
CREATE TABLE employee
   FRAGMENT BY ROUND ROBIN IN dbspaces;
```

3. The INSERT...SELECT statement maps the movement of the external data from or to the database table. The following sample INSERT statement loads the **employee** table from the external table:

```
INSERT INTO employee SELECT * FROM emp_ext
```

Important: If you specify more than one *INSERT...SELECT statement* to unload data, each subsequent INSERT statement overwrites the data file. Use absolute paths for data files.

When you load data into the database, the FROM *table* portion of the SELECT clause is the external table that the CREATE EXTERNAL statement defined. When you unload data to an external file, the SELECT clause controls the retrieval of the data from the database.

Unlike a TEMP table, the external table has a definition that remains in the catalog until it is dropped. When you create an external table you can save the external description of the data for reuse. This action is particularly helpful when you unload a table into the IBM Informix internal data representation because you can later use the same external table description to reload that data.

On Windows systems, if you use the **DB-Access** utility or the **dbexport** utility to unload a database table into a file and then plan to use the file as an external table datafile, you must define RECORDEND as '\012' in the CREATE EXTERNAL TABLE statement.

The external table definition contains all the information required to define the data in the external data file as follows:

- The description of the fields in the external data.
- The DATAFILES clause.

  This clause specifies:
  - Whether the data file is located on disk or a named pipe.
  - The path name of the file.
- The FORMAT clause.

  This clause specifies the type of data formatting in the external data file. The database server converts external data from several data formats, including delimited and fixed ASCII, and IBM Informix internal.
- Any global parameters that affect the format of the data.

If you map the external table directly into the internal database table in delimited format, you can use the CREATE EXTERNAL TABLE statement to define the columns and add the clause SAMEAS *internal-table* instead of enumerating the columns explicitly.

Map columns to other columns

If the data file is to have fields in a different order (for example, **empno**, **name**, **address**, **hiredate**), you can use the INSERT statement to map the columns. First, create the table with the columns in the order in which they are found in the external file.

```
CREATE EXTERNAL TABLE emp_ext
   (
   f01 INTEGER,
   f02 CHAR(18),
   f03 VARCHAR(40),
   f04 DATE
   )
USING (
   DATAFILES ("DISK:/work2/mydir/emp.dat"),
   REJECTFILE "/work2/mydir/emp.rej"
   );
INSERT INTO employee (empno, name, address, hiredate)
   SELECT * FROM emp_ext;
```

With this method, the insert columns are mapped to match the field order of the external table.

Another way to reorder columns is to use the SELECT clause to match the order of the database table.

```
INSERT INTO employee
   SELECT f02, f04, f03, f01 FROM emp_ext;
```

Load data from and unload to a named pipe

You can use a named pipe, also called a first-in-first-out (FIFO) data file, to load from and unload to a nonstandard device, such as a tape drive.

Unlike ordinary operating-system files, named pipes do not have a 2-gigabyte size limitation. The operating system opens and checks for the end of file differently for named pipes than for ordinary files.

Loading data with named pipes

You can use a named pipe to load data from external tables.

To use a named pipe to load data from an external table, follow these steps:

1. Specify the named pipes in the DATAFILES clause of the CREATE EXTERNAL TABLE statement in SQL.
2. Create the named pipes that you specified in the DATAFILES clause. Use operating-system commands to create the named pipes.

   Use the **mknod** UNIX command with the **-p** option to create a named pipe. To avoid blocking open problems for pipes on UNIX, start separate UNIX processes for pipe-readers and pipe-writers or open the pipes with the O_NDELAY flag set.
3. Open the named pipes with a program that reads the named pipe.
4. Execute the INSERT statement in SQL.

   ```
   INSERT INTO employee SELECT * FROM emp_ext;
   ```

Important: If you do not create and open the named pipes before you execute the INSERT statement, the INSERT succeeds, but no rows are loaded.

FIFO virtual processors

The database server uses FIFO virtual processors (VPs) to read and write to external tables on named pipes.

The default number of FIFO virtual processors is 1.

The database server uses one FIFO VP for each named pipe that you specify in the DATAFILES clause of the CREATE EXTERNAL TABLE statement. For example, suppose you define an external table with the following SQL statement:

```
CREATE EXTERNAL TABLE ext_items
   SAMEAS items
   USING (
      DATAFILES("PIPE:/tmp/pipe1",
                "PIPE:/tmp/pipe2",
                "PIPE:/tmp/pipe3"
      ));
```

If you use the default value of 1 for FIFO VPs, the database server does not read from **pipe2** until it finishes reading all the data from **pipe1**, and does not read from **pipe3** until it finishes reading all the data from **pipe2**.

Unloading data with named pipes

You can use a named pipe to unload data from the database to external tables.

To use named pipes to unload data to external tables, follow these steps:

1. Specify the named pipe in the DATAFILES clause of either the CREATE EXTERNAL TABLE statement or the SELECT INTO EXTERNAL statement of SQL.

   ```
   DATAFILES ("PIPE:/usr/local/TAPE")
   ```
2. Create the named pipes that you specified in the DATAFILES clause. Use operating-system commands to create the named pipes.
3. Open the named pipes with a program that writes to the named pipe.
4. Unload data to the named pipe.

```
CREATE EXTERNAL TABLE emp_ext
   ( name CHAR(18) EXTERNAL CHAR(20),
     hiredate DATE EXTERNAL CHAR(10),
     address VARCHAR(40) EXTERNAL CHAR(40),
     empno INTEGER EXTERNAL CHAR(6) )
   USING (
      FORMAT 'FIXED',
      DATAFILES
      ("PIPE:/usr/local/TAPE")
       );

INSERT INTO emp_ext SELECT * FROM employee;
```

Important: If you do not create and open the named pipes before you execute the SELECT or INSERT statement, the unload fails with the *ENXIO* error message (no such device or address).

Copying data from one instance to another using the PIPE option

You can use a named pipe to copy data from one Informix instance to another without writing the data to an intermediate file.

You can use a named pipe to unload data from one Informix instance and load it into another instance without writing data to an intermediate file. You can also use a named pipe to copy data from one table to another on the same Informix instance. In the following example, data is copied from a source table on one instance to a destination table on a second instance.

Depending on the hardware platform you are using, you must first create a named pipe using one of the following commands. For this example, the named pipe is called pipe1.

```
% mkfifo /work/pipe1
% mknod /work/pipe1
```

Follow these steps to copy data from a table on a source instance to a table on a destination instance on the same computer.

1. Create the source table on the source instance. In this example, the source table is called source_data_table:

```
CREATE TABLE source_data_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
);
```

2. Create the external table on the source instance. In this example, the external table is named ext_table:

```
CREATE EXTERNAL TABLE ext_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
)
USING
(DATAFILES
  (
    'PIPE:/work/pipe1'
  )
);
```

3. Create the destination table on the destination instance. In this example, the destination table is called destin_data_table:

```
CREATE TABLE destin_data_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
);
```

4. Create the external table on the destination instance. In this example, the external table is named ext_table:

```
CREATE EXTERNAL TABLE ext_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
)
USING
(DATAFILES
  (
    'PIPE:/work/pipe1_1'
  )
);
```

5. Run the following command from a UNIX shell. The command redirects data from /work/pipe1 to /work/pipe1_1

```
cat /work/pipe1 > /work/pipe1_1
```

6. Run the following command on the destination instance to direct data from the named pipe to the destination table:

```
INSERT INTO destin_data_table SELECT * FROM ext_table;
```

7. Run the following command on the source instance to spool data to the named pipe:

```
INSERT INTO ext_table SELECT * FROM source_data_table;
```

You can use more than one pipe by inserting multiple PIPE statements in the DATAFILES clause and creating a named pipe for each.

Monitor the load or unload operations

You can monitor the status of an external table load or unload operation.

You might want to monitor the load or unload operations for the following situations:

- If you expect to load and unload the same table often to build a data mart or data warehouse, monitor the progress of the job to estimate the time of similar jobs for future use.
- If you load or unload from named pipes, monitor the I/O queues to determine if you have a sufficient number of FIFO virtual processors.

Monitor frequent load and unload operations

Use the **onstat -g iof** command to find the global file descriptor (gfd) in the file that you want to examine. Then the **onstat -g sql** command to monitor load and unload operations.

The following example shows sample **onstat -g iof** command output.

```
AIO global files:
gfd path name       bytes read   page reads   bytes write   page writes  io/s
3   rootdbs         1918976      937          145061888     70831        36.5
```

```
op type      count        avg. time
seeks        0            N/A
reads        937          0.0010
writes       4088         0.0335
kaio_reads   0            N/A
kaio_writes  0            N/A
```

To determine if a load or unload operation can use parallel execution, execute the SET EXPLAIN ON statement before the INSERT statement. The SET EXPLAIN output shows the following counts:

- Number of parallel SQL operators that the optimizer chooses for the INSERT statement
- Number of rows to be processed by each SQL operator

To monitor a load operation, run **onstat -g sql** to obtain the session ID.

Monitor FIFO virtual processors

You can monitor the effective usage of FIFO VPs with **onstat** commands.

Use the **onstat -g ioq** option to display the length of each FIFO queue that is waiting to perform I/O requests. The following example shows sample output.

```
AIO I/O queues:
q name/id   len maxlen totalops  dskread dskwrite  dskcopy
 fifo  0     0    0       0         0        0         0
  adt  0     0    0       0         0        0         0
  msc  0     0    1      153        0        0         0
  aio  0     0    9     3499      1013       77         0
  pio  0     0    2       3         0        2         0
  lio  0     0    2     2159        0      2158         0
  gfd  3     0   16    39860       38     39822         0
  gfd  4     0   16    39854       32     39822         0
  gfd  5     0    1       2         2        0         0
  gfd  6     0    1       2         2        0         0
...
  gfd 19     0    1       2         2        0         0
```

The **q name** field in the sample output in the previous example shows the type of the queue, such as **fifo** for a FIFO VP or **aio** for an AIO VP. If the **q name** field shows **gfd** or **gfdwq**, it is a queue for a file whose global file descriptor matches the **id** field of the output. Disk files have both read and write requests in one queue. One line per disk file displays in the **onstat -g ioq** output. Pipes have separate read and write queues. Two lines per pipe display in the output: **gfd** for read requests and **gfdwq** for write requests.

The **len** or **maxlen** field has a value of up to 4 for a load or 4 * number_of_writer_threads for an unload. The **xuwrite** operator controls the number of writer threads.

Use the values in the **totalops** field rather than the **len** or **maxlen** field to monitor the number of read or write requests done on the file or pipe. The **totalops** field represents 34 KB of data read from or written to the file. If **totalops** is not increasing, it means the read or write operation on a file or pipe is stalled (because the FIFO VPs are busy).

To improve performance, use the **onmode -p** command to add more FIFO VPs. The default number of FIFO VPs is 1. In this sample output, the FIFO queue does

not contain any data. For example, if you usually define more than two pipes to load or unload, increase the number of FIFO VPs with the following sample **onmode** command:

```
onmode -p +2 FIFO
```

You can also use the **onmode -p** command to remove FIFO VPs. However, the number of FIFO VPs cannot be set to a value less than one.

For more information, see *IBM Informix Administrator's Reference*.

External tables in high-availability cluster environments

You use external tables on secondary servers in much the same way they are used on the primary server.

You can perform the following operations on the primary and on secondary servers:
- Unload data from a database table to an external table:
  ```
  INSERT INTO external_table SELECT * FROM base_table WHERE ...
  ```
- Load data from an external table into a database table:
  ```
  INSERT INTO base_table SELECT * FROM external_table WHERE ...
  ```

Loading data on SDS, RSS, or HDR secondary servers is slower than loading data on the primary server.

The CREATE EXTERNAL TABLE statement and the SELECT ... INTO EXTERNAL ... statement are not supported on secondary servers.

When unloading data from a database table to an external table, data files are created on the secondary server but not on the primary server. External table data files created on secondary servers are not automatically transferred to the primary server, nor are external table data files that are created on the primary server automatically transferred to secondary servers.

When creating an external table on a primary server, only the schema of the external table is replicated to the secondary servers, not the data file.

To synchronize external tables between the primary server and a secondary server, you can either copy the external table file from the primary server to the secondary servers, or use the following steps:
1. On the primary server:
   a. Create a temporary table with the same schema as the external table.
   b. Populate the temporary table:
      ```
      INSERT INTO dummy_table SELECT * FROM external_table
      ```
2. On the secondary server:
   Use the following command to populate the external table:
   ```
   INSERT INTO external_table SELECT * FROM dummy_table
   ```

System catalog entries for external tables

You can query system catalog tables to determine the status of external tables.

IBM Informix updates the **sysexternal** and **sysextdfiles** system catalog tables each time an external table is created. The **sysextcols** system catalog table is updated when the external format type (**fmttype**) FIXED is specified.

Table 10-1. External table system catalog entries

Table name	Description
sysexternal	Stores information about each external table.
sysextdfiles	Stores information about external table data files
sysextcols	Stores information about external tables of type FIXED

See the *IBM Informix Guide to SQL: Reference* for more information.

A row is inserted into the **systables** system catalog when an external table is created; however, the **nrows** (number of rows) and the **npused** (number of data pages used) columns might not accurately reflect the number of rows and the number of data pages used by the external table unless the NUMROWS clause was specified when the external table was created.

When an external table is created without specifying a value for the NUMROWS clause, Informix is unable to determine the number of rows in the external table because the data exists outside the database in data files. Informix updates the **nrows** column in the **systables** system catalog by inserting a large value (MAXINT – 1), and computes the number of data pages used based on the **nrows** value. The values stored in **npused** and **nrows** are later used by the optimizer to determine the most efficient execution plan. While the NUMROWS clause is not required to be specified precisely, the more accurately it is specified, the more accurate the values for **nrows** and **npused** are.

Performance considerations when using external tables

Use external tables when you want to manipulate data in an ASCII file using SQL commands, or when loading data from an external data file to a RAW database table.

There are several ways to load information into a database, including:
- LOAD FROM ... INSERT INTO... DB-Access command
- **dbimport** utility
- High-Performance Loader utility
- External tables

The High Performance Loader utility provides best performance for loading external data into a database table with indexes.

External tables provide the best performance for loading data into a RAW table with no indexes.

Note: Locking an external table prior to loading data increases the load performance

Manage errors from external table load and unload operations

You can manage errors that occur during external table load and unload operations.

These topics describe how to use the reject file and error messages to manage errors, and how to recover data loaded into the database.

Reject files

Rows that have conversion errors during a load are written to a reject file on the server that performs the conversion.

The REJECTFILE keyword in the CREATE EXTERNAL TABLE statement determines the name given to the reject file.

Instead of using a reject file, you can use the MAXERRORS keyword in the CREATE EXTERNAL TABLE statement to specify the number of errors that are allowed before the database server stops loading data. (If you do not set the MAXERRORS keyword, the database server processes all data regardless of the number of errors.)

The database server removes the reject files, if any, at the beginning of a load. The reject files are recreated and written only if errors occur during the load.

Reject file entries are single lines with the following comma-separated fields:

file name, record, reason-code, field-name: bad-line

file name
        Name of the input file

record   Record number in the input file where the error was detected

reason-code
        Description of the error

field-name
        The external field name where the first error in the line occurred or <none> if the rejection is not specific to a particular column

bad-line
        For delimited or fixed-ASCII files only, the bad line itself

The load operation writes *file name*, *record*, *field-name*, and *reason-code* in ASCII.

The *bad-line* information varies with the type of input file:

- For delimited files or fixed text files, the entire bad line is copied directly into the reject file. However, if the delimited format table has TEXT or BYTE columns, the reject file does not include any bad data. The load operation generates only a header for each rejected row.
- For IBM Informix internal data files, the bad line is not placed in the reject file because you cannot edit the binary representation in a file. However, the *file name*, *record*, *reason-code*, and *field-name* are still reported in the reject file so that you can isolate the problem.

The following types of errors can cause a row to be rejected.

CONSTRAINT *constraint name*
        This constraint was violated.

CONVERT_ERR
        Any field encounters a conversion error.

MISSING_DELIMITER
        No delimiter was found.

MISSING_RECORDEND
> No record end was found.

NOT NULL
> A null was found in *field-name*.

ROW_TOO_LONG
> The input record is longer than 2 GB.

External table error messages

Most of the error messages related to external tables are in the -26151 to -26199 range.

Additional messages are -615, -999, -23852, and -23855. In the messages, *n macro* and *r macro* refer to the values generated from the substitution character %r(*first..last*). For a list of error messages, see *IBM Informix Error Messages* or use the **finderr** utility. For information about the violations table error messages, see your *IBM Informix Administrator's Reference*.

Recoverability of table types for external tables

The database server checks the recoverability level of the table when loading of data.

- If the table type is RAW, the database server can use light-append (or express) mode to load data and process check constraints. If the database server crashes during the load, the data loaded is not rolled back, and the table might be left in an unknown state.
- If the table type is STATIC, the database server cannot load the data at all.
- Only deluxe mode supports data recoverability. Deluxe mode uses logged, regular inserts. To recover data after a failed express-mode load, revert to the most recent level-0 backup. The table type must be STANDARD for this level of recoverability.

For information about restoring table types, see the *IBM Informix Backup and Restore Guide*.

Part 3. Logging and log administration

Chapter 11. Logging

These topics describe logging of IBM Informix databases and addresses the following questions:

- Which database server processes require logging?
- What is transaction logging?
- What database server activity is logged?
- What is the database-logging status?
- Who can set or change the database logging status?

All the databases managed by a single database server instance store their log records in the same logical log, regardless of whether they use transaction logging. Most database users might be concerned with whether transaction logging is buffered or whether a table uses logging.

If you want to change the database-logging status, see "Settings or changes for logging status or mode" on page 11-7.

Database server processes that require logging

As IBM Informix operates, processing transactions, tracking data storage, and ensuring data consistency, Informix automatically generates *logical-log records* for some of the actions that it takes. Most of the time the database server makes no further use of the logical-log records. However, when the database server is required to roll back a transaction, to run a fast recovery after a system failure, for example, the logical-log records are critical. The logical-log records are at the heart of the data-recovery mechanisms.

The database server stores the logical-log records in a *logical log*. The logical log is made up of *logical-log files* that the database server manages on disk until they have been safely transferred offline (*backed up*). The database server administrator keeps the backed up logical-log files until they are required during a data restore, or until the administrator decides that the records are no longer required for a restore. See Chapter 13, "Logical log," on page 13-1 for more information about logical logs.

The logical-log records themselves are variable length. This arrangement increases the number of logical-log records that can be written to a page in the logical-log buffer. However, the database server often flushes the logical-log buffer before the page is full. For more information about the format of logical-log records, see the topics about interpreting logical-log records in the *IBM Informix Administrator's Reference*.

The database server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

Transaction rollback
> If a database is using transaction logging and a transaction must be rolled back, the database server uses the logical-log records to reverse the changes made during the transaction. For more information, see "Transaction logging" on page 11-2.

11-1

Fast recovery

If the database server shuts down in an uncontrolled manner, the database server uses the logical-log records to recover all transactions that occurred since the oldest update not yet flushed to disk and to roll back any uncommitted transactions. (When all the data in shared memory and on disk are the same, they are *physically consistent*.) The database server uses the logical-log records in fast recovery when it returns the entire database server to a state of logical consistency up to the point of the most recent logical-log record. (For more information, see "Fast recovery after a checkpoint" on page 15-7.)

Data restoration

The database server uses the most recent storage-space and logical-log backups to recreate the database server system up to the point of the most recently backed-up logical-log record. The logical restore applies all the log records since the last storage-space backup.

Deferred checking

If a transaction uses the SET CONSTRAINTS statement to set checking to DEFERRED, the database server does not check the constraints until the transaction is committed. If a constraint error occurs while the transaction is being committed, the database server uses logical-log records to roll back the transaction. For more information, see SET Database Object Mode in the *IBM Informix Guide to SQL: Syntax*.

Cascading deletes

Cascading deletes on referential constraints use logical-log records to ensure that a transaction can be rolled back if a parent row is deleted and the system fails before the children rows are deleted. For information about table inheritance, see the *IBM Informix Database Design and Implementation Guide*. For information about primary key and foreign key constraints, see the *IBM Informix Guide to SQL: Tutorial*.

Distributed transactions

Each database server involved in a distributed transaction keeps logical-log records of the transaction. This process ensures data integrity and consistency, even if a failure occurs on one of the database servers that is performing the transaction. For more information, see "Two-phase commit and logical-log records" on page 25-18.

Data Replication

Data Replication environments that use HDR secondary, SD secondary, and RS secondary servers use logical-log records to maintain consistent data on the primary and secondary database servers so that one of the database servers can be used quickly as a backup database server if the other fails. For more details, see "How data replication works" on page 22-1.

Enterprise Replication

You must use database logging with Enterprise Replication because it replicates the data from the logical-log records. For more information, see the *IBM Informix Enterprise Replication Guide*.

Transaction logging

A database or table is said to have or use transaction logging when SQL data manipulation statements in a database generate logical-log records.

The database-logging *status* indicates whether a database uses transaction logging. The *log-buffering mode* indicates whether a database uses buffered or unbuffered

logging, or ANSI-compliant logging. For more information, see "Database-logging status" on page 11-5 and Chapter 12, "Manage the database-logging mode," on page 12-1.

When you create a database, you specify whether it uses *transaction logging* and, if it does, what log-buffering mechanism it uses. After the database is created, you can turn off database logging or change to buffered logging, for example. Even if you turn off transaction logging for all databases, the database server always logs some events. For more information, see "Activity that is always logged" and "Database logging in an X/Open DTP environment" on page 11-7.

You can use logging or nonlogging tables within a database. The user who creates the table specifies the type of table. Even if you use nonlogging tables, the database server always logs some events. For more information, see "Table types for Informix" on page 8-24.

Logging of SQL statements and database server activity

Three types of logged activity are possible in the database server:

Activity that is always logged

Some database operations always generate logical-log records, even if you turn off transaction logging or use nonlogging tables.

The following operations are always logged for permanent tables:
- Certain SQL statements, including SQL data definition statements
- Storage-space backups
- Checkpoints
- Administrative changes to the database server configuration such as adding a chunk or dbspace
- Allocation of new extents to tables
- A change to the logging status of a database
- Smart-large-object operations:
  – Creating
  – Deleting
  – Allocating and deallocating extents
  – Truncating
  – Combining and splitting chunk free list pages
  – Changing the LO header and the LO reference count
- Sbspace metadata
- Blobspaces

The following table lists statements that generate operations that are logged even if transaction logging is turned off.

Table 11-1. SQL statements that generate operations that are always logged.

ALTER ACCESS_METHOD	CREATE SECURITY LABEL	DROP TRUSTED CONTEXT
ALTER FRAGMENT	CREATE SECURITY LABEL	DROP TYPE
ALTER FUNCTION	COMPONENT	DROP USERDROP VIEW
ALTER INDEX	CREATE SECURITY POLICY	DROP XADATASOURCE
ALTER PROCEDURE	CREATE SEQUENCE	DROP XADATASOURCE
ALTER ROUTINE	CREATE SYNONYM	TYPE
ALTER SECURITY LABEL	CREATE TABLE	GRANT
COMPONENT	CREATE TEMP TABLE	GRANT FRAGMENT
ALTER SEQUENCE	CREATE TRIGGER	RENAME COLUMN
ALTER TABLE	CREATE TRUSTED	RENAME DATABASE
ALTER TRUSTED CONTEXT	CONTEXT	RENAME INDEX
ALTER USERCLOSE DATABASE	CREATE USERCREATE VIEW	RENAME SECURITY
CREATE ACCESS_METHOD	CREATE XADATASOURCE	RENAME SEQUENCE
CREATE AGGREGATE	CREATE XADATASOURCE	RENAME TABLE
CREATE CAST	TYPE	RENAME TRUSTED
CREATE DATABASE	DROP ACCESS_METHOD	CONTEXT
CREATE DISTINCT TYPE	DROP AGGREGATE	RENAME USERREVOKE
CREATE EXTERNAL	DROP CAST	REVOKE FRAGMENT
TABLE	DROP DATABASE	TRUNCATE
CREATE FUNCTION	DROP FUNCTION	UPDATE STATISTICS
CREATE FUNCTION FROM	DROP INDEX	SAVE EXTERNAL
CREATE INDEX	DROP OPCLASS	DIRECTIVES
CREATE OPAQUE TYPE	DROP PROCEDURE	SET CONSTRAINTS
CREATE OPCLASS	DROP ROLE	SET Database Object Mode
CREATE PROCEDURE	DROP ROUTINE	SET INDEXES
CREATE PROCEDURE	DROP ROW TYPE	SET TRIGGERS
FROM	DROP SECURITY	START VIOLATIONS
CREATE ROLE	DROP SEQUENCE	TABLE
CREATE ROUTINE FROM	DROP SYNONYM	STOP VIOLATIONS
CREATE ROW TYPE	DROP TABLE	TABLE
CREATE SCHEMA	DROP TRIGGER	

Activity logged for databases with transaction logging

If a database uses transaction logging, the following SQL statements generate one or more log records. If these statements are rolled back, the rollback also generates log records.

Table 11-2. SQL statements that generate one or more logs only if transaction logging is turned on.

DELETE	LOAD	SELECT INTO TEMP
FLUSH	MERGE	UNLOAD
INSERT	PUT	UPDATE

The following SQL statements generate logs in special situations.

Table 11-3. SQL statements that generate logs in special situations.

SQL statement	Log record that the statement generates
BEGIN WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
COMMIT WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
ROLLBACK WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
EXECUTE	Whether this statement generates a log record depends on the command being run.

Table 11-3. SQL statements that generate logs in special situations. (continued)

SQL statement	Log record that the statement generates
EXECUTE FUNCTION	Whether this statement generates a log record depends on the function being executed.
EXECUTE IMMEDIATE	Whether this statement generates a log record depends on the command being run.
EXECUTE PROCEDURE	Whether this statement generates a log record depends on the procedure being executed.

Activity that is not logged

The following SQL statements do not produce log records, regardless of the database logging mode.

Table 11-4. SQL statements that generate operations that are never logged.

ALLOCATE COLLECTION	INFO	SET ISOLATION
ALLOCATE DESCRIPTOR	LOCK TABLE	SET LOCK MODE
ALLOCATE ROW	OPEN	SET LOG
CLOSE	OUTPUT	SET OPTIMIZATION
CONNECT	PREPARE	SET PDQPRIORITY
DATABASE	RELEASE SAVEPOINT	SET ROLE
DEALLOCATE COLLECTION	SAVEPOINT	SET SESSION AUTHORIZATION
DEALLOCATE DESCRIPTOR	SELECT	SET STATEMENT CACHE
DEALLOCATE ROW	SET AUTOFREE	SET TRANSACTION
DECLARE	SET COLLATION	SET Transaction Mode
DESCRIBE	SET CONNECTION	SET USER PASSWORD
DISCONNECT	SET DATASKIP	UNLOCK TABLE
FETCH	SET DEBUG FILE	WHENEVER
FREE	SET DEFERRED_PREPARE	SET ENVIRONMENT
GET DESCRIPTOR	SET DESCRIPTOR	SET EXPLAIN
GET DIAGNOSTICS	SET ENCRYPTION PASSWORD	

For temporary tables in temporary dbspaces, nothing is logged, not even the SQL statements listed in "Activity that is always logged" on page 11-3. If you include temporary (nonlogging) dbspaces in DBSPACETEMP, the database server places nonlogging tables in these temporary dbspaces first. For more information, see "Temporary tables" on page 8-34.

Database-logging status

You must use transaction logging with a database to take advantage of any of the features listed in "Database server processes that require logging" on page 11-1.

Every database that the database server manages has a logging status. The logging status indicates whether the database uses transaction logging and, if so, which log-buffering mechanism the database employs. To find out the transaction-logging status of a database, use the database server utilities, as explained in "Monitor the logging mode of a database" on page 12-6. The database-logging status indicates any of the following types of logging:

- Unbuffered transaction logging
- Buffered transaction logging
- ANSI-compliant transaction logging
- No logging

All logical-log records pass through the logical-log buffer in shared memory before the database server writes them to the logical log on disk. However, the point at

which the database server flushes the logical-log buffer is different for buffered transaction logging and unbuffered transaction logging. For more information, see Figure 6-1 on page 6-2 and "Flush the logical-log buffer" on page 6-28.

Unbuffered transaction logging

If transactions are made against a database that uses unbuffered logging, the records in the logical-log buffer are guaranteed to be written to disk during commit processing. When control returns to the application after the COMMIT statement (and before the PREPARE statement for distributed transactions), the logical-log records are on the disk. The database server flushes the records as soon as any transaction in the buffer is committed (that is, a commit record is written to the logical-log buffer).

When the database server flushes the buffer, only the used pages are written to disk. Used pages include pages that are only partially full, however, so some space is wasted. For this reason, the logical-log files on disk fill up faster than if all the databases on the same database server use buffered logging.

Unbuffered logging is the best choice for most databases because it guarantees that all committed transactions can be recovered. In the event of a failure, only uncommitted transactions at the time of the failure are lost. However, with unbuffered logging, the database server flushes the logical-log buffer to disk more frequently, and the buffer contains many more partially full pages, so it fills the logical log faster than buffered logging does.

Buffered transaction logging

If transactions are made against a database that uses buffered logging, the records are held (*buffered*) in the logical-log buffer for as long as possible. They are not flushed from the logical-log buffer in shared memory to the logical log on disk until one of the following situations occurs:

- The buffer is full.
- A commit on a database with unbuffered logging flushes the buffer.
- A checkpoint occurs.
- The connection is closed.

If you use buffered logging and a failure occurs, you cannot expect the database server to recover the transactions that were in the logical-log buffer when the failure occurred. Thus, you might lose some committed transactions. In return for this risk, performance during alterations improves slightly. Buffered logging is best for databases that are updated frequently (when the speed of updating is important), as long as you can recreate the updates in the event of failure. You can tune the size of the logical-log buffer to find an acceptable balance for your system between performance and the risk of losing transactions to system failure.

ANSI-compliant transaction logging

The ANSI-compliant database logging status indicates that the database owner created this database using the MODE ANSI keywords. ANSI-compliant databases always use unbuffered transaction logging, enforcing the ANSI rules for transaction processing. You cannot change the buffering status of ANSI-compliant databases.

No database logging

If you turn off logging for a database, transactions are not logged, but other operations are logged. For more information, see "Activity that is always logged" on page 11-3. Usually, you would turn off logging for a database when you are loading data, or just running queries.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of database server processing. For example, if you are loading many rows into a database from a recoverable source such as tape or an ASCII file, you might not require transaction logging, and the loading would proceed faster without it. However, if other users are active in the database, you would not have logical-log records of their transactions until you reinitiate logging, which must wait for a level-0 backup.

Databases with different log-buffering status

All databases on a database server use the same logical log and the same logical-log buffers. Therefore, transactions against databases with different log-buffering statuses can write to the same logical-log buffer. In that case, if transactions exist against databases with buffered logging and against databases with unbuffered logging, the database server flushes the buffer either when it is full or when transactions against the databases with unbuffered logging complete.

Database logging in an X/Open DTP environment

Databases in the X/Open distributed transaction processing (DTP) environment must use *unbuffered logging*. Unbuffered logging ensures that the database server logical logs are always in a consistent state and can be synchronized with the transaction manager. If a database created with buffered logging is opened in an X/Open DTP environment, the database status automatically changes to unbuffered logging. The database server supports both ANSI-compliant and non-ANSI databases. For more information, see "Transaction managers" on page 25-1.

Settings or changes for logging status or mode

The user who creates a database with the CREATE DATABASE statement establishes the logging status or buffering mode for that database. For more information about the CREATE DATABASE statement, see the *IBM Informix Guide to SQL: Syntax*.

If the CREATE DATABASE statement does not specify a logging status, the database is created without logging.

Only the database server administrator can change logging status. Chapter 12, "Manage the database-logging mode," on page 12-1, describes this topic. Ordinary users cannot change database-logging status.

If a database does not use logging, you are not required to consider whether buffered or unbuffered logging is more appropriate. If you specify logging but do not specify the buffering mode for a database, the default is unbuffered logging.

Users *can* switch from unbuffered to buffered (but not ANSI-compliant) logging and from buffered to unbuffered logging for the *duration of a session*. The SET LOG statement performs this change within an application. For more information about the SET LOG statement, see the *IBM Informix Guide to SQL: Syntax*.

Chapter 12. Manage the database-logging mode

You can monitor and modify the database-logging mode.

The topics in this section provide information about:
- Understanding database-logging mode
- Modifying database-logging mode with **ondblog**
- Modifying database-logging mode with **ontape**
- Modifying database-logging mode with ON-Monitor
- Monitoring transaction logging

As a database server administrator, you can alter the logging mode of a database as follows:
- Change transaction logging from buffered to unbuffered.
- Change transaction logging from unbuffered to buffered.
- Make a database ANSI compliant.
- Add transaction logging (buffered or unbuffered) to a database.
- End transaction logging for a database.

For information about database-logging mode, when to use transaction logging, and when to buffer transaction logging, see Chapter 11, "Logging," on page 11-1. To find out the current logging mode of a database, see "Monitor the logging mode of a database" on page 12-6.

For information about using SQL administration API commands instead of some **ondblog** and **ontape** commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Administrator's Reference*.

Change the database-logging mode

You can use **ondblog** or **ontape** to add or change logging. Then use ON-Bar, or **ontape** to back up the data. When you use ON-Bar or **ontape**, the database server must be in online, administration, or quiescent mode.

You can use **ondblog**, **ontape**, or IBM Informix Server Administrator (ISA) to add or change logging. Then use ON-Bar, or **ontape** to back up the data. When you use ON-Bar or **ontape**, the database server must be in online, administration, or quiescent mode.

For information about ON-Bar and **ontape**, see the *IBM Informix Backup and Restore Guide*.

The following table shows how the database server administrator can change the database-logging mode. Certain logging mode changes take place immediately, while other changes require a level-0 backup.

Table 12-1. Logging mode transitions

Converting from:	Converting to no logging	Converting to unbuffered logging	Converting to buffered logging	Converting to ANSI compliant
No logging	Not applicable	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)
Unbuffered logging	Yes	Not applicable	Yes	Yes
Buffered logging	Yes	Yes	Not applicable	Yes
ANSI compliant	Illegal	Illegal	Illegal	Not applicable

Changing the database-logging mode has the following effects:

- While the logging status is being changed, the database server places an exclusive lock on the database to prevent other users from accessing the database, and frees the lock when the change is complete.

- If a failure occurs during a logging-mode change, check the logging mode in the flags in the **sysdatabases** table in the **sysmaster** database, after you restore the database server data. For more information, see "Monitor the logging mode of a database" on page 12-6. Then try the logging-mode change again.

- If a failure occurs during a logging-mode change, check the logging mode in ISA or the flags in the **sysdatabases** table in the **sysmaster** database, after you restore the database server data. For more information, see "Monitor the logging mode of a database" on page 12-6. Then try the logging-mode change again.

- After you choose either buffered or unbuffered logging, an application can use the SQL statement SET LOG to change from one logging mode to the other. This change lasts for the duration of the session. For information about SET LOG, see the *IBM Informix Guide to SQL: Syntax*.

- If you add logging to a database, the change is not complete until the next level-0 backup of all the storage spaces for the database.

Modify the database-logging mode with ondblog

You can use the **ondblog** utility to change the logging mode for one or more databases. If you add logging to a database, you must create a level-0 backup of the dbspace(s) that contains the database before the change takes effect. For more information, see the topics on using **ondblog** in the *IBM Informix Administrator's Reference*.

Change the buffering mode with ondblog

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, run the following command:

```
ondblog unbuf stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, run the following command:

```
ondblog buf stores_demo
```

Cancel a logging mode change with ondblog

To cancel the logging mode change request before the next level-0 backup occurs, run the following command:

```
ondblog cancel stores_demo
```

You cannot cancel the logging changes that are executed immediately.

End logging with ondblog

To end logging for two databases that are listed in a file called `dbfile`, run the following command:

```
ondblog nolog -f dbfile
```

Make a database ANSI compliant with ondblog

To make a database called **stores_demo** into an ANSI-compliant database with **ondblog**, run the following command:

```
ondblog ansi stores_demo
```

Changing the logging mode of an ANSI-compliant database

After you create or convert a database to ANSI mode, you cannot easily change it to any other logging mode. If you accidentally convert a database to ANSI mode, follow these steps to change the logging mode:

To change the logging mode:

1. To unload the data, use **dbexport** or any other migration utility. The **dbexport** utility creates the `schema` file.

   For information about how to load and unload data, see the *IBM Informix Migration Guide*.

2. To recreate a database with buffered logging and load the data, use the **dbimport -l buffered** command.

   To recreate a database with unbuffered logging and load the data, use the **dbimport -l** command.

Modify the database logging mode with ontape

If you use **ontape** as your backup tool, you can use **ontape** to change the logging mode of a database.

Turn on transaction logging with ontape

Before you modify the database-logging mode, read "Change the database-logging mode" on page 12-1.

You add logging to a database with **ontape** at the same time that you create a level-0 backup.

For example, to add buffered logging to a database called **stores_demo** with **ontape**, run the following command:

```
ontape -s -B stores_demo
```

To add unbuffered logging to a database called **stores_demo** with **ontape**, run the following command:

```
ontape -s -U stores_demo
```

In addition to turning on transaction logging, these commands create full-system storage-space backups. When **ontape** prompts you for a backup level, specify a level-0 backup.

Tip: With **ontape**, you must perform a level-0 backup of all storage spaces.

End logging with ontape

To end logging for a database called **stores_demo** with **ontape**, run the following command:

```
ontape -N stores_demo
```

Change buffering mode with ontape

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, run the following command:

```
ontape -U stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, run the following command:

```
ontape -B stores_demo
```

Make a database ANSI compliant with ontape

To make a database called **stores_demo**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database with **ontape**, run the following command:

```
ontape -A stores_demo
```

To make a database called **stores_demo**, which does not already use transaction logging, into an ANSI-compliant database with **ontape**, run the following command:

```
ontape -s -A stores_demo
```

In addition to making a database ANSI compliant, this command also creates a storage-space backup at the same time. Specify a level-0 backup when you are prompted for a level.

Tip: After you change the logging mode to ANSI compliant, you cannot easily change it again. To change the logging mode of ANSI-compliant databases, unload the data, recreate the database with the new logging mode, and reload the data. For details, see "Changing the logging mode of an ANSI-compliant database" on page 12-3.

Modify database logging mode with ISA

IBM Informix Server Administrator (ISA) uses the **ondblog** utility to modify the database-logging mode. If you turn on logging, perform a level-0 backup. For more information, see the ISA online help and "Modify the database-logging mode with ondblog" on page 12-2.

Modify database logging mode with ON-Monitor (UNIX)

You can use ON-Monitor to change the log-buffering mode between unbuffered and buffered. If you want to add logging to a database or make it ANSI compliant, you cannot use ON-Monitor. You must use **ontape**.

To change the log-buffering mode for a database, select the **Logical-Logs** > **Databases** option.

Modify the table-logging mode

The database server creates standard tables that use logging by default. To create a nonlogging table, use the CREATE TABLE statement with the WITH LOG clause. For information about the CREATE TABLE and ALTER TABLE statements, see the *IBM Informix Guide to SQL: Syntax*. For more information, see "Table types for Informix" on page 8-24.

Alter a table to turn off logging

To switch a table from logging to nonlogging, use the SQL statement ALTER TABLE with the TYPE option of RAW. For example, the following statement changes table **tablog** to a RAW table:

```
ALTER TABLE tablog TYPE (RAW)
```

Alter a table to turn on logging

To switch from a nonlogging table to a logging table, use the SQL statement ALTER TABLE with the TYPE option of STANDARD. For example, the following statement changes table **tabnolog** to a STANDARD table:

```
ALTER TABLE tabnolog TYPE (STANDARD)
```

Important: When you alter a table to STANDARD, you turn logging on for that table. After you alter the table, perform a level-0 backup if you must be able to restore the table.

Disable logging on temporary tables

You can disable logging on temporary tables to improve performance and to prevent IBM Informix from transferring temporary tables when using a primary server in a data replication environment such as with HDR secondary, RS secondary, and SD secondary servers.

To disable logging on temporary tables, set the TEMPTAB_NOLOG configuration parameter to 1.

For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1.

You can use the **onmode -wf** command to change the value of TEMPTAB_NOLOG.

Monitor transactions

This topic contains references for information about ways to monitor transactions.

Command	Description	Reference
onstat -x	Monitor transactions.	"Monitor a global transaction" on page 25-15
onstat -g sql	Monitor SQL statements, listed by session ID and database.	Performance monitoring in the *IBM Informix Performance Guide*
onstat -g stm	Monitor memory usage of prepared SQL statements.	Memory utilization in the *IBM Informix Performance Guide*

Monitor the logging mode of a database

These topics explain ways to monitor the logging mode of your database and tables.

Monitor the logging mode with SMI tables

Query the **sysdatabases** table in the **sysmaster** database to determine the logging mode. This table contains a row for each database that the database server manages. The **flags** field indicates the logging mode of the database. The **is_logging**, **is_buff_log**, and **is_ansi** fields indicate whether logging is active, and whether buffered logging or ANSI-compliant logging is used. For a description of the columns in this table, see the **sysdatabases** section in the chapter about the **sysmaster** database in the *IBM Informix Administrator's Reference*.

Monitor the logging mode with ON-Monitor (UNIX)

To use ON-Monitor to find out the logging mode of a database, select the **Status** > **Databases** option.

When database logging is on, ON-Monitor shows the buffering mode. ON-Monitor can only display up to 100 databases. If you have more than 100 databases on your database server, use the SMI tables to display the full list, as described in "Monitor the logging mode with SMI tables."

Monitor the logging mode with ISA

To use IBM Informix Server Administrator (ISA) to display the logging status and buffering mode for your databases, select **SQL** > **Schema**.

Chapter 13. Logical log

The information in Chapter 11, "Logging," on page 11-1, and these topics explains how the database server uses the logical log. For information about how to perform logical-log tasks, see Chapter 14, "Manage logical-log files," on page 14-1, and Chapter 12, "Manage the database-logging mode," on page 12-1.

What is the logical log?

To keep a history of transactions and database server changes since the time of the last storage-space backup, the database server generates log records. The database server stores the log records in the *logical log*, a circular file that is composed of three or more logical-log files. The log is called *logical* because the log records represent logical operations of the database server, as opposed to physical operations. At any time, the combination of a storage-space backup plus logical-log backup contains a complete copy of your database server data.

As the database server administrator, you must configure and manage the logical log. For example, if you do not back up the log files regularly, the logical log fills and the database server suspends processing.

These responsibilities include the following tasks:
- Choosing an appropriate location for the logical log

  See "Location of logical-log files."
- Monitoring the logical-log file status

  See "Identification of logical-log files" on page 13-2.
- Allocating an appropriate amount of disk space for the logical log

  See "Size of the logical-log file" on page 13-3.
- Allocating additional log files whenever necessary

  See "Allocate log files" on page 14-8.
- Backing up the logical-log files to media

  See "Back up logical-log files" on page 14-3 and "Freeing of logical-log files" on page 13-5.
- Managing logging of blobspaces and sbspaces

  See "Log blobspaces and simple large objects" on page 13-6 and "Log sbspaces and smart large objects" on page 13-7.

Location of logical-log files

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no control over this action. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize contention), move the logical-log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log. See "Moving a logical-log file to another dbspace" on page 14-12.

To improve performance further, separate the logical-log files into two groups and store them on two separate disks (neither of which contains data). For example, if you have six logical-log files, you might locate files 1, 3, and 5 on disk 1, and files

2, 4, and 6 on disk 2. This arrangement improves performance because the same disk drive never is required to handle writes to the current logical-log file and backups to tape at the same time.

The logical-log files contain critical information and must be mirrored for maximum data protection. If you move logical-log files to a different dbspace, plan to start mirroring on that dbspace.

Identification of logical-log files

Each logical-log file, whether backed up to media or not, has a unique ID number. The sequence begins with 1 for the first logical-log file filled after you initialize the database server disk space. When the current logical-log file becomes full, the database server switches to the next logical-log file and increments the unique ID number for the new log file by one. Log files that are newly added or marked for deletion have unique ID numbers of 0.

The actual disk space allocated for each logical-log file has an identification number known as the *log file number*. For example, if you configure six logical-log files, these files have log numbers one through six. The log numbers might be out of sequence. As logical-log files are backed up and freed, the database server reuses the disk space for the logical-log files.

The following table illustrates the relationship between the log numbers and the unique ID numbers. Log 7 is inserted after log 5 and used for the first time in the second rotation.

Table 13-1. Logical-log file-numbering sequence

Log file number	First rotation unique ID number	Second rotation unique ID number	Third rotation unique ID number
1	1	7	14
2	2	8	15
3	3	9	16
4	4	10	17
5	5	11	18
7	0	12	19
6	6	13	20

Status flags of logical-log files

All logical-log files have one of the following status flags in the first position: Added (**A**), Deleted (**D**), Free (**F**), or Used (**U**). The following table shows the possible log-status flag combinations.

Table 13-2. Logical-log status flags

Status flag	Status of logical-log file
A------	Log file has been added, and is available, but has not yet been used.
D------	If you drop a log file with a status of U-B, it is marked as deleted. This log file is dropped and its space is freed for reuse when you take a level-0 backup of all storage spaces.

Table 13-2. Logical-log status flags (continued)

Status flag	Status of logical-log file
F------	Log file is free and available for use. A logical-log file is freed after it is backed up, all transactions within the logical-log file are closed, and the oldest update stored in this file is flushed to disk.
U	Log file has been used but not backed up.
U-B----	Log file is backed up but still required for recovery. (The log file is freed when it is no longer required for recovery.)
U-B---L	Log is backed up but still required for recovery. Contains the last checkpoint record.
U---C	The database server is currently filling the log file.
U---C-L	This current log file contains the last checkpoint record.

Use the **onstat -l** command to list the log files by number and monitor the status flags and percentage of log space used. For more details, see "The onstat -l command" on page 14-7.

Size of the logical-log file

The minimum size for a logical-log file is 200 KB.

The maximum size for a logical-log file is 524288 pages (equivalent to 0x7ffff + 1), with a 2 KB or 4 KB base-page size, depending on the operating system. To determine the database server's base-page size on your operating system, run **onstat -d** and then check the `pgsize` value for the root dbspace.

Determine the size and number of log files to use. If you allocate more disk space than necessary, space is wasted. If you do not allocate enough disk space, however, performance might be adversely affected. Use larger log files when many users are writing to the logs at the same time.

Note: Smaller log files mean that you can recover to a later time if the disk that contains the log files goes down. If continuous log backup is set, log files are automatically backed up as they fill. Smaller logs result in slightly longer logical recovery.

Number of logical-log files

When you think about the number of logical-log files, consider these points:
- You must always have at least three logical-log files and a maximum of 32,767 log files. The number of log files depends on the size of the log files.
- The number of log files affects the frequency of logical-log backups.
- The number of logical-log files affects the rate at which blobspace blobpages can be reclaimed. See "Back up log files to free blobpages" on page 13-6.

Performance considerations

For a given level of system activity, the less logical-log disk space that you allocate, the sooner that logical-log space fills up, and the greater the likelihood that user activity is blocked due to backups and checkpoints. Tune the logical-log size to find the optimum value for your system.

- Logical-log backups

  When the logical-log files fill, you must back them up. The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. Put the physical log, logical logs, and user data on separate disks. (See the *IBM Informix Backup and Restore Guide*.)

- Size of the logical log

  A smaller logical log fills faster than a larger logical log. You can add a larger logical-log file, as explained in "Adding logical-log files manually" on page 14-10.

- Size of individual logical-log records

  The sizes of the logical-log records vary, depending on both the processing operation and the database server environment. In general, the longer the data rows, the larger the logical-log records. The logical log contains images of rows that have been inserted, updated, or deleted. Updates can use up to twice as much space as inserts and deletes because they might contain both before-images and after-images. (Inserts store only the after-image and deletes store only the before-image.)

- Number of logical-log records

  The more logical-log records written to the logical log, the faster it fills. Databases with transaction logging fill the logical log faster than transactions against databases without transaction logging.

- Type of log buffering

  Databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.

- Enterprise Replication on a table

  Because Enterprise Replication generates before-images and after-images of the replicated tables, it might cause the logical log to fill.

- Frequency of rollbacks

  More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small.

- Number of smart large objects

  Smart large objects that have user data logging enabled and have a large volume of user data updates can fill logical logs at a prodigious rate. Use temporary smart large objects if you do not want to log the metadata.

Dynamic log allocation

Dynamic log allocation prevents log files from filling and hanging the system during long transaction rollbacks. The only time that this feature becomes active is when the next log file contains an open transaction. (A *transaction* is *long* if it is not committed or rolled back when it reaches the long-transaction high-watermark.)

The database server automatically (dynamically) allocates a log file after the current log file when the next log file contains an open transaction. You can use dynamic log allocation for the following actions:

- Add a log file while the system is active
- Insert a log file after the current log file
- Immediately access new log files even if the root dbspace is not backed up

The best way to test dynamic log allocation is to produce a transaction that spans all the log files and then use **onstat -l** to check for newly added log files. For more information, see "Allocate log files" on page 14-8.

Important: You still must back up log files to prevent them from filling. If the log files fill, the system hangs until you perform a backup.

Freeing of logical-log files

Each time the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began. The following criteria must be satisfied before the database server frees a logical-log file for reuse:

- The log file is backed up.
- No records within the logical-log file are associated with open transactions.
- The logical-log file does not contain the oldest update not yet flushed to disk.

Action if the next logical-log file is not free

If the database server attempts to switch to the next logical-log file but finds that the next log file in sequence is still in use, the database server immediately suspends all processing. Even if other logical-log files are free, the database server cannot skip a file in use and write to a free file out of sequence. Processing stops to protect the data within the logical-log file.

The logical-log file might be in use for any of the following reasons:

- The file contains the latest checkpoint or the oldest update not yet flushed to disk.

  Issue the **onmode -c** command to perform a checkpoint and free the logical-log file. For more information, see "Force a checkpoint" on page 16-4.
- The file contains an open transaction.

  The open transaction is the long transaction explained in "Set high-watermarks for rolling back long transactions" on page 14-15.
- The file is not backed up.

  If the logical-log file is not backed up, processing resumes when you use ON-Bar or **ontape** to back up the logical-log files.

Action if the next log file contains the last checkpoint

The database server does not suspend processing when the next log file contains the last checkpoint or the oldest update. The database server always forces a checkpoint when it switches to the last available log, if the previous checkpoint record or oldest update that is not yet flushed to disk is located in the log that follows the last available log. For example, if four logical-log files have the status shown in the following list, the database server forces a checkpoint when it switches to logical-log file 3.

Log file number
        Logical-log file status

1   U-B----

2   U---C--

3   F

4   U-B---L

Log blobspaces and simple large objects

Simple-large-object data (TEXT and BYTE data types) is potentially too voluminous to include in a logical-log record. If simple large objects are always logged, they might be so large that they slow down the logical log.

The database server assumes that you designed your databases so that smaller simple large objects are stored in dbspaces and larger simple large objects are stored in blobspaces:

- The database server includes simple-large-object data in log records for simple large objects stored in dbspaces.
- The database server does not include simple-large-object data in log records for simple large objects stored in blobspaces. The logical log records blobspace data only when you back up the logical logs.

To obtain better overall performance for applications that perform frequent updates of simple large objects in blobspaces, reduce the size of the logical log. Smaller logs can improve access to simple large objects that must be reused. For more information, see the chapter on configuration effects on I/O utilization in your *IBM Informix Performance Guide*.

Switch log files to activate blobspaces

You must switch to the next logical-log file in these situations:

- After you create a blobspace, if you intend to insert simple large objects in the blobspace right away
- After you add a new chunk to an existing blobspace, if you intend to insert simple large objects in the blobspace that uses the new chunk

The database server requires that the statement that creates a blobspace, the statement that creates a chunk in the blobspace, and the statements that insert simple large objects into that blobspace are created in separate logical-log files. This requirement is independent of the database-logging status.

For instructions on switching to the next log file, see "Switch to the next logical-log file" on page 14-4.

Back up log files to free blobpages

When you delete data stored in blobspace pages, those pages are not necessarily freed for reuse. The blobspace pages are free only when both of the following actions have occurred:

- The TEXT or BYTE data has been deleted, either through an UPDATE to the column or by deleting the row
- The logical log that stores the INSERT of the row that has TEXT or BYTE data is backed up

Back up blobspaces after inserting or deleting TEXT and BYTE data

Be sure to back up all blobspaces and logical logs containing transactions on simple large objects stored in a blobspace. During log backup, the database server uses the data pointer in the logical log to copy the changed text and byte data from the blobspace into the logical log.

Log sbspaces and smart large objects

Sbspaces, described in "Sbspaces" on page 8-13, contain two components: metadata and user data. By default, sbspaces are not logged.

The metadata component of the sbspace describes critical characteristics of smart large objects stored in a particular sbspace. The metadata contains pointers to the smart large objects. If the metadata were to be damaged or become inaccessible, the sbspace would be corrupted and the smart large objects within that sbspace would be unrecoverable.

Metadata in a standard sbspace is always logged, even if logging is turned off for a database. Logging sbspace metadata ensures that the metadata can always be recovered to a consistent transaction state. However, metadata in a temporary sbspace is not logged.

Sbspace logging

When an sbspace is logged, the database server slows down, and the logical logs fill up quickly. If you use logging for sbspaces, you must ensure that the logical logs are large enough to hold the logging data. For more information, see "Estimate the log size when logging smart large objects" on page 14-3.

When you turn on logging for a database, the database server does not begin logging until you perform a level-0 backup. However, when you turn on logging for a smart large object, the database server begins logging changes to it immediately. To reduce the volume of log entries, load smart large objects with logging turned off and then turn logging back on to capture updates to the smart large objects.

Important: When you turn logging on for a smart large object, you must immediately perform a level-0 backup to be able to recover and restore the smart large object.

For more information, see "Back up sbspaces" on page 14-4 and the *IBM Informix Backup and Restore Guide*.

Logging for smart large objects

Use logging for smart large objects if users are updating the data frequently or if the ability to recover any updated data is critical. The database server writes a record of the operation (insert, update, delete, read, or write) to the logical-log buffer. The modified portion of the CLOB or BLOB data is included in the log record.

To increase performance, turn off logging for smart large objects. Also turn off logging if users are primarily analyzing the data and updating it infrequently, or if the data is not critical to recover.

Logging for updated smart large objects

When you update a smart large object, the database server does not log the entire object. Assume that the user is writing X bytes of data at offset Y with logging enabled for smart large objects. The database server logs the following information:

- If Y is set to the end of the large object, the database server logs X bytes (the updated byte range).
- If Y is at the beginning or in the middle of the large object, the database server logs the smallest of these choices:

- Difference between the old and new image
- Before-image and after-image
- Nothing is logged if the before- and after-images are the same

Turn logging on or off for an sbspace

You can control whether logging is on of off for an sbspace with several different methods.

If you want to use logging in an sbspace, specify the **-Df "LOGGING=ON"** option of the **onspaces** command when you create the sbspace. If logging is turned off in the sbspace, you can turn on logging for smart large objects in specific columns. One column that contains smart large objects can have logging turned on while another column has logging turned off.

To verify that smart large objects in an sbspace are logged, use the **oncheck -pS** *sbspace_name* **|** **grep "Create Flags"** command.

If you create smart large objects in the sbspace with the default logging option and you see the LO_NOLOG flag in the output, the smart large objects in this sbspace are not logged. If you see the LO_LOG flag in the output, all smart large objects in this sbspace are logged.

You can modify the logging status of an sbspace in any of the following ways.

Function or statement to specify	Logging action	References
onspaces -ch -Df "LOGGING=ON" **onspaces -ch -Df "LOGGING=OFF"**	Turns logging on or off for an existing sbspace	"Alter storage characteristics of smart large objects" on page 9-25 onspaces -ch: Change sbspace default specifications
The SQL administration API **task()** or **admin()** function with the **set sbspace logging on** or **set sbspace logging off** argument	Turns logging on or off for an existing sbspace	set sbspace logging argument: Change the logging of an sbspace (SQL administration API)
LOG option in the PUT clause of the CREATE TABLE or alter table statement	Turns on logging for all smart large objects that you load into the column	"Logging" on page 8-17 PUT Clause
mi_lo_create DataBlade API function	Turns off logging for a smart large object when it is initially loaded	*IBM Informix DataBlade API Function Reference*
mi_lo_alter DataBlade API function	Turns on logging after the load is complete	*IBM Informix DataBlade API Function Reference*
ifx_lo_create Informix ESQL/C function	Turns off logging for a smart large object when it is initially loaded	*IBM Informix ESQL/C Programmer's Manual*
ifx_lo_alter Informix ESQL/C function	Turns on logging after the load is complete	*IBM Informix ESQL/C Programmer's Manual*

Smart-large-object log records

When you create a smart large object with the LOG option, the logical log creates a *smart-blob log record*. Smart-blob log records track changes to user data or metadata. When smart large objects are updated, only the modified portion of the sbpage is in the log record. User-data log records are created in the logical log only when logging is enabled for the smart large object.

Warning: Be careful about enabling logging for smart large objects that are updated frequently. This logging overhead might significantly slow down the database server.

For information about the log records for smart large objects, see the chapter on interpreting logical-log records in the *IBM Informix Administrator's Reference*.

Prevent long transactions when logging smart-large-object data

You can use smart large objects in situations where the data collection process for a single smart large object lasts for long periods of time. Consider, for example, an application that records many hours of low-quality audio information. Although the amount of data collected might be modest, the recording session might be long, resulting in a long-transaction condition.

Tip: To prevent long transactions from occurring, periodically commit writes to smart large objects.

Logging process

These topics describe in detail the logging process for dbspaces, blobspaces, and sbspaces. This information is not required for performing normal database server administration tasks.

Dbspace logging

The database server uses the following logging process for operations that involve data stored in dbspaces:

1. Reads the data page from disk to the shared-memory page buffer
2. Copies the unchanged page to the physical-log buffer, if required
3. Writes the new data to the page buffer and creates a logical-log record of the transaction, if required
4. Flushes the physical-log buffer to the physical log on disk
5. Flushes the logical-log buffer to a logical-log file on disk
6. Flushes the page buffer and writes it back to disk

Blobspace logging

The database server logs blobspace data, but the data does not pass through either shared memory or the logical-log files on disk. The database server copies data stored in a blobspace directly from disk to tape. Records of modifications to the blobspace overhead pages (the free-map and bitmap pages) are the only blobspace data that reaches the logical log.

Chapter 14. Manage logical-log files

You must manage logical-log files even if none of your databases uses transaction logging. See Chapter 13, "Logical log," on page 13-1 for background information about logical logs.

You must log-in as either **informix** or **root** on UNIX to make any of the changes described in this chapter. You must be a member of the **Informix-Admin** group on Windows.

You perform these tasks when setting up your logical log:
* Before you initialize or restart the database server, use the LOGFILES parameter to specify the number of logical-log files to create.
* After the database server is online, estimate the size and number of logical-log files that your system requires.

  See "Estimate the size and number of log files."
* If you do not want to use the default values, change the LOGSIZE and LOGBUFF configuration parameters.

  See "Changing logging configuration parameters" on page 14-13.
* Add the estimated number of logical-log files.

  See "Allocate log files" on page 14-8.

You perform the following tasks routinely:
* Backing up a logical-log file
* Switching to the next logical-log file
* Freeing a logical-log file
* Monitoring logging activity and log-backup status

You perform these tasks occasionally, if necessary:
* Adding a logical-log file
* Dropping a logical-log file
* Changing the size of a logical-log file
* Moving a logical-log file
* Changing the logical-log configuration parameters
* Monitoring event alarms for logical logs
* Setting high-watermarks for transactions

For information about using SQL administration API commands instead of some **oncheck**, **onmode**, **onparams** and **onspaces** commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Administrator's Reference*.

Estimate the size and number of log files

Use the LOGSIZE configuration parameter to set the size of the logical-log files.

The amount of log space that is optimal for your database server system depends on the following factors:

- Your application requirements and the amount of update activity your applications experience. Increased update activity requires increased log space.
- The recovery time objective (RTO) standards for the amount of time, in seconds, that the server is given to recover from a problem after you restart the server and bring it into online or quiescent mode.

   In the case of a catastrophic event, consider how much data loss you can tolerate. More frequent log backups, which reduce the risk of data and transaction loss, require increased log space.
- Whether you use Enterprise Replication or data replication configurations such as HDR secondary, SD secondary or RS secondary servers.

   These replication services can influence the number and size of log files. If your system uses any of these replication services, see guidelines in Chapter 21, "High-availability cluster configuration," on page 21-1 or in the *IBM Informix Enterprise Replication Guide*.

Some guidelines for determining log size are:
- Generally, you can more easily manage a few large log files than you can manage many small log files.
- Having too much log space does not affect performance. However, not having enough log files and log space can affect performance, because the database server triggers frequent checkpoints.
- Smart large objects in blobspaces are not logged, but they are included in the log backup in which the object was created. This means that the objects are not freed until the server backs up the log in which they were created. Therefore, if smart large objects in a blobspace are frequently updated, you might require more frequent log backups to acquire additional free space within a blobspace.
- For applications that generate a small amount of log data, start with 10 log files of 10 megabytes each.
- For applications that generate a large amount of log data, start with 10 log files with 100 megabytes.

There are two ways to maintain an RPO policy, which determines the tolerance for loss of data in case of a catastrophic event such as the loss of the data server:
- One way to maintain an RPO policy is to use automatic log backups that trigger log backups whenever a log file fills up. This limits data loss to the transactions contained in the log file during the backup, plus any additional transactions that occur during the log backup.
- Another way to maintain an RPO policy is to use the Scheduler. You can create a task that automatically backs up any new log data at timed intervals since the last log backup. This limits data loss to the transactions not backed up between time intervals. For information about using the Scheduler, see Chapter 27, "The Scheduler," on page 27-1.

If an RPO policy is required, you can use the Scheduler to insert a task that executes at an appropriate frequency to maintain the policy. This automatically backs up log files at certain times within the daily cycle. If the log space fills before the logs being backed up and recycled, you can back up the logs and add a new log file to allow transaction processing to continue, or you can use the Scheduler to add a new task to detect this situation and perform either operation automatically.

You can add log files at any time, and the database server automatically adds log files when required for transaction consistency, for example, for long transactions that might consume large amounts of log space.

The easiest way to increase the amount of space for the logical log is to add another logical-log file. See "Adding logical-log files manually" on page 14-10.

The following expression provides an example total-log-space configuration, in KB:

```
LOGSIZE = (((connections * maxrows) * rowsize) / 1024) / LOGFILES
```

Expression element	Explanation
LOGSIZE	Specifies the size of each logical-log file in KB.
connections	Specifies the maximum number of connections for all network types that you specify in the `sqlhosts` file or registry and in the NETTYPE parameter. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, add the users fields for each NETTYPE, and substitute this total for *connections* in the preceding formula.
maxrows	Specifies the largest number of rows to be updated in a single transaction.
rowsize	Specifies the average size of a table row in bytes. To calculate the *rowsize*, add the length (from the **syscolumns** system catalog table) of the columns in the row.
1024	Converts the LOGSIZE to the specified units of KB.
LOGFILES	Specifies the number of logical-log files.

Estimate the log size when logging smart large objects

If you plan to log smart-large-object user data, you must ensure that the log size is considerably larger than the amount of data being written. If you store smart large objects in standard sbspaces, the metadata is always logged, even if the smart large objects are not logged. If you store smart large objects in temporary sbspaces, there is no logging at all.

Estimate the number of logical-log files

The LOGFILES parameter provides the number of logical-log files at system initialization or restart. If all your logical-log files are the same size, you can calculate the total space allocated to the logical-log files as follows:

```
total logical log space = LOGFILES * LOGSIZE
```

If the database server contains log files of different sizes, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you must add the sizes for each individual log file on disk. Check the **size** field in the **onstat -l** output. For more information, see "The onstat -l command" on page 14-7.

For information about LOGSIZE, LOGFILES, and NETTYPE, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

Back up logical-log files

The logical logs contain a history of the transactions that have been performed. The process of copying a logical-log file to media is called *backing up* a logical-log file. Backing up logical-log files achieves the following two objectives:

- It stores the logical-log records on media so that they can be rolled forward if a data restore is required.
- It makes logical-log-file space available for new logical-log records.

If you neglect to back up the log files, you can run out of log space.

You can initiate a manual logical-log backup or set up continuous logical-log backups. After you restore the storage spaces, you must restore the logical logs to bring the data to a consistent state. For more information about log backups, see the *IBM Informix Backup and Restore Guide*.

Backing up blobspaces

It does not matter whether you back up the logical logs or blobspaces first.

To back up blobspace data:
1. Close the current logical log if it contains transactions on simple large objects in a blobspace.
2. Perform a backup of the logical logs and blobspace as soon as possible after updating simple-large-object data.

Warning: If you do not back up these blobspaces and logical logs, you might not be able to restore the blobspace data. If you wait until a blobspace is down to perform the log backup, the database server cannot access the blobspace to copy the changed data into the logical log.

Back up sbspaces

When you turn on logging for smart large objects, you must perform a level-0 backup of the sbspace.

The following figure shows what happens if you turn on logging in an sbspace that is not backed up. The unlogged changes to smart large object **LO1** are lost during the failure, although the logged changes are recoverable. You cannot fully restore **LO1**.

During fast recovery, the database server rolls forward all committed transactions for **LO1**. If **LO1** is unlogged, the database server would be unable to roll back uncommitted transactions. Then the **LO1** contents would be incorrect. For more information, see "Fast recovery" on page 15-6.
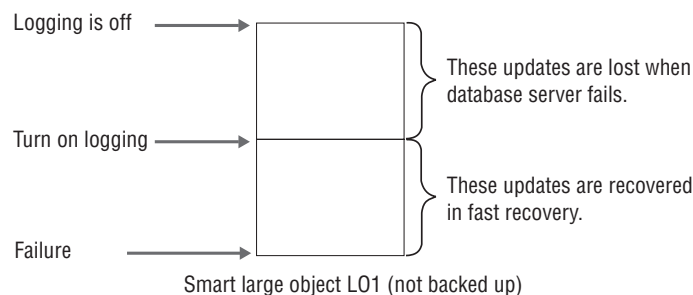
Figure 14-1. Turn on logging in an sbspace

Switch to the next logical-log file

You might want to switch to the next logical-log file before the current log file becomes full for the following reasons:
- To back up the current log
- To activate new blobspaces and blobspace chunks

The database server can be in online mode to make this change. Run the following command to switch to the next available log file: **onmode -l**

The change takes effect immediately. (Be sure that you type a lowercase L on the command line, not a number 1.)

Free a logical-log file

If a log file is newly added (status **A**), it is immediately available for use. It also can be dropped immediately.

You might want to free a logical-log file for the following reasons:
- So that the database server does not stop processing
- To free the space used by deleted blobpages

The procedures for freeing log files vary, depending on the status of the log file. Each procedure is described in the following topics. To find out the status of logical-log files, see "Status flags of logical-log files" on page 13-2 and "Monitor logging activity" on page 14-6.

Tip: For information using ON-Bar or **ontape** to back up storage spaces and logical logs, see the *IBM Informix Backup and Restore Guide*.

Delete a log file with status D

When you drop a used log file, it is marked as deleted (status D) and cannot be used again, and **onparams** prints this message:

```
Log file log_file_number has been pre-dropped. It will be
deleted from the log list and its space can be reused
once you take level 0 archives of all BLOBspaces,
Smart BLOBspaces and non-temporary DBspaces.
```

The level 0 archive is necessary to make sure that the log file itself and all of the associated information in the different dbspaces has been archived. The log file is deleted at the end of the level 0 archive; however, because the removal of the log file is itself a change in the root reserved pages structure on the disk, the next archive to be taken also must be a level 0 archive. The level 0 archive must occur before a level 1 or level 2 archive can be performed.

Free a log file with status U

If a log file contains records, but is not yet backed up (status U), back up the file using the backup tool that you usually use.

If backing up the log file does not change the status to free (F), its status changes to either U-B or U-B-L. See "Freeing a log file with status U-B or F" or "Free a log file with status U-B-L" on page 14-6.

Freeing a log file with status U-B or F

If a log file is backed up but still in use (status U-B), some transactions in the log file are still under way, or the log file contains the oldest update that is required for fast recovery. Because a log file with status F has been used in the past, it follows the same rules as for status U-B.

To free a backed up log file that is in use:

1. If you do not want to wait until the transactions complete, take the database server to quiescent mode. See "Change immediately from online to quiescent mode" on page 3-12. Any active transactions are rolled back.
2. Use the **onmode -c** command to force a checkpoint. Do this because a log file with status U-B might contain the oldest update.

A log file that is backed up but not in use (status U-B) is not required to be freed. In the following example, log 34 is not required to be freed, but logs 35 and 36 do. Log 35 contains the last checkpoint, and log 36 is backed up but still in use.

```
34 U-B--   Log is used, backed up, and not in use
35 U-B-L   Log is used, backed up, contains last checkpoint
36 U-B--   Log is used, backed up, and not in use
37 U-C--   This is the current log file, not backed up
```

Tip: You can free a logical log with a status of U-B (and not L) only if it is not spanned by an active transaction and does not contain the oldest update.

Freeing a log file with status U-C or U-C-L

Follow these steps to free the current log file.

To free the current log file (status C):
1. Run the following command to switch the current log file to the next available log file: **onmode -l**
2. Back up the original log file with ON-Bar or **ontape**.
3. After all full log files are backed up, you are prompted to switch to the next available log file and back up the new current log file.

   You are not required to do the backup because you just switched to this log file.

After you free the current log file, if the log file has status U-B or U-B-L, see "Freeing a log file with status U-B or F" on page 14-5 or "Free a log file with status U-B-L."

Free a log file with status U-B-L

If a log file is backed up and all transactions within it are closed but the file is not free (status U-B-L), this logical-log file contains the most-recent checkpoint record. You can free log files with a status U-B-L.

To free log files with a status U-B-L, the database server must create a new checkpoint. You can run the following command to force a checkpoint: **onmode -c**

UNIX only: To force a checkpoint with ON-Monitor, select the **Force-Ckpt** option.

Monitor logging activity

Monitor the logical-log files to determine the total available space (in all the files), the space available in the current file, and the status of a file (for example, whether the log has been backed up yet). For information about monitoring the logical-log buffers, see "Monitor physical and logical-logging activity" on page 16-2.

Monitor the logical log for fullness

You can use the following command-line utilities to monitor logical-log files.

The onstat -l command

The **onstat -l** command displays information about physical and logical logs.

The output section that contains information about each logical-log file includes the following information:

- The address of the logical-log file descriptor
- The log file number
- Status flags that indicate the status of each log (free, backed up, current, and so on)
- The unique ID of the log file
- The beginning page of the file
- The size of the file in pages, the number of pages used, and the percentage of pages used

The log file numbers in the **numbers** field can get out of sequence if you drop several logs in the middle of the list or if the database server dynamically adds log files.

For more information about and an example of **onstat -l** output, see the *IBM Informix Administrator's Reference*.

The oncheck -pr command

The database server stores logical-log file information in the reserved pages dedicated to checkpoint information. Because the database server updates this information only during a checkpoint, it is not as recent as the information that the **onstat -l** option displays. For more details on using these options to display reserved page information, see the *IBM Informix Administrator's Reference*.

You can view the checkpoint reserved pages with the **oncheck -pr** command. The following example shows sample output for one of the logical-log files.

```
...
Log file number            1
Unique identifier          7
Log contains last checkpoint Page 0, byte 272
Log file flags             0x3  Log file in use
                                 Current log file
Physical location          0x1004ef
Log size                   750 (p)
Number pages used          1
Date/Time file filled      01/29/2001 14:48:32
...
```

Monitor temporary logical logs

The database server uses *temporary logical logs* to roll forward transactions during a warm restore, because the permanent logs are not available then. When the rollforward completes, the database server frees the temporary log files. If you issue **onstat -l** during a warm restore, the output includes a fourth section on temporary log files in the same format as regular log files. Temporary log files use only the B, C, F, and U status flags.

SMI tables

Query the **syslogs** table to obtain information about logical-log files. This table contains a row for each logical-log file. The columns are as follows.

number
> Identification number of the logical-log file

uniqid
> Unique ID of the log file

size Size of the file in pages

used Number of pages used

is_used
> Flag that indicates whether the log file is being used

is_current
> Flag that indicates whether the log file is current

is_backed_up
> Flag that indicates whether the log file has been backed up

is_new
> Flag that indicates whether the log file has been added since the last storage space backup

is_archived
> Flag that indicates whether the log file has been written to the archive tape

is_temp
> Flag that indicates whether the log file is flagged as a temporary log file

Monitor log status with ON-Monitor (UNIX)

You can use ON-Monitor to view information about logical logs.

The **Status** > **Logs** option displays much of the same information for logical-log files as the **onstat -l** option displays. In addition, a column contains the dbspace in which each logical-log file is located.

Monitor log-backup status

To monitor the status of the logs and to see which logs have been backed up, use the **onstat -l** command. A status flag of B indicates that the log has been backed up.

Allocate log files

When you initialize or restart the database server, it creates the number of logical-log files that you specify in the LOGFILES configuration parameter. These log files are the size that you specify in the LOGSIZE parameter.

Dynamically add a logical-log file

The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file. When you use the default value of 2 for DYNAMIC_LOGS, the database server dynamically adds a new log file and sets off an alarm if the next active log file contains the beginning of the oldest open transaction.

The database server checks the logical-log space at these points:
- After switching to a new log file
- At the beginning of the transaction-cleanup phase of logical recovery

If the DYNAMIC_LOGS parameter is set to 1 and the next active log file contains records from an open transaction, the database server prompts you to add a log file manually and sets off an alarm. After you add the log file, the database server resumes processing the transaction.

If the DYNAMIC_LOGS parameter is set to 0 and the logical log runs out of space during a long transaction rollback, the database server can hang. (The long transaction prevents the first logical-log file from becoming free and available for reuse.) To fix the problem and complete the long transaction, set DYNAMIC_LOGS to 2 and restart the database server.

For more information, see "Monitor events for dynamically added logs" on page 14-14 and "Set high-watermarks for rolling back long transactions" on page 14-15.

Size and number of dynamically added log files

The purpose of dynamic logs is to add enough log space to allow transactions to roll back. When dynamically adding a log file, the database server uses the following factors to calculate the size of the log file:

- Average log size
- Amount of contiguous space available

If the logical log is low on space, the database server adds as many log files as necessary to allow the transaction to roll back. The number of log files is limited by:

- The maximum number of log files supported
- The amount of disk space for the log files
- The amount of free contiguous space in the root chunk

If the database server stops adding new log files because it is out of disk space, it writes an error message and sets off an alarm. Add a dbspace or chunk to an existing dbspace. Then the database server automatically resumes processing the transaction.

The reserve pages in the root chunk store information about each log file. The extents that contain this information expand as more log files are added. The root chunk requires two extents of 1.4 megabytes each to track 32,767 log files, the maximum number supported.

If during reversion, the chunk reserve page extent is allocated from a non-root chunk, the server attempts to put it back in the root chunk. If not enough space is available in the root chunk, the reversion fails. A message containing the required space displays in the online log. The required space must be freed from the root chunk before trying the reversion again.

Location of dynamically added log files

The database server allocates log files in dbspaces, in the following search order. A dbspace becomes critical if it contains logical-log files or the physical log.

Pass	Allocate log file in
1	The dbspace that contains the newest log files
	(If this dbspace is full, the database server searches other dbspaces.)
2	Mirrored dbspace that contains log files (but excluding the root dbspace)
3	All dbspaces that already contain log files (excluding the root dbspace)

4	The dbspace that contains the physical log
5	The root dbspace
6	Any mirrored dbspace
7	Any dbspace

If you do not want to use this search order to allocate the new log file, you must set the DYNAMIC_LOGS parameter to 1 and run **onparams -a -i** with the location you want to use for the new log. For details, see "Monitor events for dynamically added logs" on page 14-14.

Adding logical-log files manually

You can use an **onparams** command or ON-Monitor to add logical-log files.

You might add logical-log files manually for the following reasons:
- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- To enable an open transaction to roll back
- As part of moving logical-log files to a different dbspace

Restriction: You cannot do the following actions:
- Add a log file to a blobspace or sbspace.
- Add logical or physical logs to dbspaces that have non-default page sizes.

Add logical-log files one at a time, up to a maximum of 32,767 files, to any dbspace. As soon as you add a log file to a dbspace, it becomes a critical dbspace. You can add a logical-log file during a storage space backup.

You can add a logical-log file in either of the following locations:
- At the end of the file list using the **onparams -a** command
- After the current logical-log file using the **onparams -a -i** command

To add a logical-log file using **onparams**
1. Log-in as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows.
2. Ensure that the database server is in online, administration, or quiescent, or cleanup phase of fast-recovery mode.

   The database server writes the following message to the log during the cleanup phase:

   ```
   Logical recovery has reached the transaction cleanup phase.
   ```
3. Decide whether you want to add the log file to the end of the log file list or after the current log file.

   You can insert a log file after the current log file regardless of the DYNAMIC_LOGS parameter value. Adding a log file of a new size does not change the value of LOGSIZE.

   - The following command adds a logical-log file to the end of the log file list in the **logspace** dbspace, using the log-file size specified by the LOGSIZE configuration parameter:

     ```
     onparams –a –d logspace
     ```
   - The following command inserts a 1000 KB logical-log file after the current log file in the **logspace** dbspace:

```
onparams -a -d logspace -s 1000 -i
```
   - To add a logical-log file with a new size (in this case, 250 KB), run the following command:
```
onparams -a -d logspace -s 250
```
4. Use **onstat -l** to check the status of the log files. The status of the new log file is **A** and is immediately available.
5. The next time you must back up data, perform a level-0 backup of the root dbspace and the dbspaces that contain the new log files.

   Although you are no longer required to back up immediately after adding a log file, your next backup must be level-0 because the data structures have changed. For information about backing up data, see the *IBM Informix Backup and Restore Guide*.

For more information about using **onparams** to add a logical-log file, see the *IBM Informix Administrator's Reference*.

To add a logical-log file with ON-Monitor (UNIX)

1. Follow the instructions on adding a log file in "Adding logical-log files manually" on page 14-10, except use ON-Monitor instead of **onparams**.
2. Select **Parameters** > **Add-Log** to add a logical-log file.
3. In the field labeled **Dbspace Name**, enter the name of the dbspace where the new logical-log file is located.

   The size of the log file automatically is in the **Logical Log Size** field. The new log file is always the value specified by LOGSIZE.

Dropping logical-log files

You can use an **onparams** command or ON-Monitor to drop logical-log files.

To drop a logical-log file and increase the amount of the disk space available within a dbspace, you can use **onparams**. The database server requires a minimum of three logical-log files at all times. You cannot drop a log if your logical log is composed of only three log files.

The rules for dropping log files have changed:
- If you drop a log file that has never been written to (status **A**), the database server deletes it and frees the space immediately.
- If you drop a used log file (status **U-B**), the database server marks it as deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbspace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use or contains the last checkpoint record (status **C** or **L**).

To drop a logical-log file with **onparams**:

1. Ensure that the database server is in online, administration, or quiescent mode.
2. Run the following command to drop a logical-log file whose log file number is 21: **onparams -d -l 21**

   Drop log files one at a time. You must know the log file number of each logical log that you intend to drop.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.

   If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.

This backup prevents the database server from using the dropped log files during a restore and ensures that the reserved pages contain information about the current number of log files.

For information about using **onparams** to drop a logical-log file, see the *IBM Informix Administrator's Reference*.

For information about using **onlog** to display the logical-log files and unique ID numbers, see "Display logical-log records" on page 14-14.

To drop a logical-log file with ON-Monitor (UNIX):
1. Ensure that the database server is in online, administration, or quiescent mode.
2. To drop a logical-log file, select **Parameters** > **Drop-Log**.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.

   If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.

   Tip: If the root dbspace has never been backed up, you can drop a **used** log file immediately.

Change the size of logical-log files

If you want to change the size of the log files, it is easier to add new log files of the appropriate size and then drop the old ones. You can change the size of logical-log files in the following ways:
- Use **onparams** with the **-s** option to add a new log file of a different size.
  See "Adding logical-log files manually" on page 14-10.
- Increase the LOGSIZE value in the `onconfig` file if you want the database server to create larger log files.
  See "Changing logging configuration parameters" on page 14-13.

Moving a logical-log file to another dbspace

You might want to move a logical-log file for performance reasons or to make more space in the dbspace, as explained in "Location of logical-log files" on page 13-1. To find the location of logical-log files, see "Monitor logging activity" on page 14-6. Although moving the logical-log files is not difficult, it can be time-consuming.

Moving logical-log files is a combination of two simpler actions:
- Dropping logical-log files from their current dbspace
- Adding the logical-log files to their new dbspace

The following procedure provides an example of how to move six logical-log files from the **root** dbspace to another dbspace, **dbspace_1**.

Restriction: You cannot move logical and physical log files in dbspaces that have non-default page sizes.

To move the logical-log files out of the root dbspace (example):
1. Ensure that the database server is in online, administration, quiescent, or fast-recovery mode.

2. Add six new logical-log files to **dbspace_1**. See "Adding logical-log files manually" on page 14-10.
3. Take a level-0 backup of all storage spaces to free all log files except the current log file.

   (If you use **onbar -l -b -c**, you back up all log files including the current log file.) See "Free a logical-log file" on page 14-5.
4. Use **onmode -l** to switch to a new current log file. See "Switch to the next logical-log file" on page 14-4.
5. Drop all six logical-log files in the root dbspace.

   You cannot drop the current logical-log file.

   See "Dropping logical-log files" on page 14-11.
6. Create a level-0 backup of the root dbspace and **dbspace_1**.

   For more information, see the *IBM Informix Backup and Restore Guide*.

Changing logging configuration parameters

You can use a text editor to change the configuration parameters that the database server uses for logging

The following table shows the configuration parameters for logical logs. For more information, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

Configuration parameter	Minimum Value	Default Value	Maximum Value
DYNAMIC_LOGS	0 or 1	2	2
LOGBUFF	2 * page size	64 KB	LOGSIZE value
LOGFILES	3 files	5 files	32,767 files
LOGSIZE	10000 KB on UNIX 500 KB on Windows	10000 KB	See the *IBM Informix Administrator's Reference*
LTXEHWM	LTXHWM value	90%	100%
LTXHWM	1%	80%	100%

Important: The change to LOGFILES does not take effect until you reinitialize or restart the disk space.

To change the logical-log configuration parameters in the `onconfig` file:
1. Bring the database server offline or into quiescent or administration mode.
2. Use a text editor to update the configuration parameters.

   The DYNAMIC_LOGS, LTXHWM, and LTXEHWM parameters are not in the `onconfig.std` file. To change the values of these parameters, add them to your `onconfig` file.
3. Shut down and restart the database server.
4. Perform this step only if you are changing LOGFILES and want all of the log files to be in continuous space. (Normally you add and drop LOGFILES using the **onparams** utility.)
   a. Unload all the database server data. Do this because you cannot rely on storage space backups to unload and restore the data because a restore returns the parameters to their previous value.

b. Reinitialize the database server disk space. See "Initializing disk space" on page 3-1.

c. Recreate all databases and tables.

d. Reload all the database server data.

For information about loading and unloading data, see the *IBM Informix Migration Guide*.

5. Back up the root dbspace to enable your changed logical logs.

Display logical-log records

Use the **onlog** utility to display and interpret logical-log records. For information about using **onlog**, see the *IBM Informix Administrator's Reference*.

Monitor events for dynamically added logs

Monitor the following event alarms that dynamically added log files trigger (see the following table). When each alarm is triggered, a message is written to the message log. For more information, see the chapters on event alarms and configuration parameters in the *IBM Informix Administrator's Reference*.

You can include the **onparams** command to add log files in your alarm script for event class ID 27, log file required. Your script can also run the **onstat -d** command to check for adequate space and run the **onparams a -i** command with the location that has enough space. You must use the **-i** option to add the new log right after the current log file.

Table 14-1. Event alarms for dynamically added log files

Class ID	Severity	Class message	Message
26	3	Dynamically added log file *log_number*	This message displays when the database server dynamically adds a log file. Dynamically added log file `log_number` to DBspace `dbspace_number`.
27	4	Log file required	This message displays when DYNAMIC_LOGS is set to 1 and the database server is waiting for you to add a log file. ALERT: The oldest logical log `log_number` contains records from an open transaction `transaction_address`. Logical logging remains blocked until a log file is added. Add the log file with the **onparams -a** command, using the **-i** (insert) option, as in: `onparams -a -d dbspace -s size-i` Then complete the transaction as soon as possible.
28	4	No space for log file	ALERT: Because the oldest logical log *log_number* contains records from an open transaction *transaction_address*, the server is attempting to dynamically add a log file. But there is no space available. Add a dbspace or chunk, then complete the transaction as soon as possible.

The following table shows the actions that the database server takes for each setting of the DYNAMIC_LOGS configuration parameter.

Table 14-2. The DYNAMIC_LOGS settings

DYNAMIC_ LOGS	Meaning	Event alarm	Wait to add log	Dynamic log add
2 (default)	Allows automatic allocation of new log files to prevent open transactions from hanging the system.	Yes (26, 28)	No	Yes
1	Allows manual addition of new log files.	Yes (27)	Yes	No
0	Does not allocate log files but issues the following message about open transactions: **Warning:** The oldest logical-log file *log_number* contains records from an open transaction *transaction_address,* but the dynamic log feature is turned off.	No	No	No

Set high-watermarks for rolling back long transactions

The database server uses the LTXHWM and LTXEHWM configuration parameters to set high-watermarks for long transactions. If DYNAMIC_LOGS is set to 1 or 2, the default LTXHWM value is 80 percent and LTXEHWM is 90 percent. If DYNAMIC_LOGS is set to 0, the default LTXHWM value is 50 percent and the default LTXHEWM value is 60 percent.

If you decrease your high-watermark values, you increase the likelihood of long transactions. To compensate, allocate additional log space. For information about LTXHWM and LTXEHWM, see the chapter on configuration parameters in the *IBM Informix Administrator's Reference.*

Long-transaction high-watermark (LTXHWM)

The *long-transaction high-watermark* is the percentage of total log space that a transaction is allowed to span before it is rolled back. If the database server finds an open transaction in the oldest used log file, it dynamically adds log files. Because the log space is increasing, the high-watermark expands outward. When the log space reaches the high-watermark, the database server rolls back the transaction. The transaction rollback and other processes also generate logical-log records. The database server continues adding log files until the rollback is complete to prevent the logical log from running out of space. More than one transaction can be rolled back if more than one long transaction exists.

For example, the database server has ten logical logs and LTXHWM is set to 98. A transaction begins in log file 1 and update activity fills logs 1 through 9. The database server dynamically adds log file 11 after log file 10. As long as the transaction does not complete, this process continues until the database server has added 40 log files. When the database server adds the fiftieth log, the transaction has caught up to the high-watermark and the database server rolls it back.

Exclusive access, long-transaction high-watermark (LTXEHWM)

The *exclusive-access, long-transaction high-watermark* occurs when the long transaction currently being rolled back is given exclusive access to the logical log. The database server dramatically reduces log-record generation. Only threads that are currently rolling back transactions and threads that are currently writing COMMIT records are allowed access to the logical log. Restricting access to the logical log

preserves as much space as possible for rollback records that are being written by the user threads that are rolling back transactions.

Important: If you set both LTXHWM and LTXEHWM to 100, long transactions are never stopped. Therefore, you must set LTXHWM to below 100 for normal database server operations. Set LTXHWM to 100 to run scheduled transactions of unknown length. Set LTXEHWM to 100 if you never want to block other users while a long transaction is rolling back and you have ample disk space.

Adjust the size of log files to prevent long transactions

Use larger log files when many users are writing to the logs at the same time. If you use small logs and long transactions are likely to occur, reduce the high-watermark. Set the LTXHWM value to 50 and the LTXEHWM value to 60.

If the log files are too small, the database server might run out of log space while rolling back a long transaction. In this case, the database server cannot block fast enough to add a new log file before the last one fills. If the last log file fills, the system hangs and displays an error message. To fix the problem, shut down and restart the database server. For details, see "Recovering from a long transaction hang."

Recovering from a long transaction hang

If your system has ample disk space and you want to perform transactions of unknown length, consider setting LTXHWM to 100 to force the database server to continue adding log files until you complete the transaction.

A transaction might hang because the database server has run out of disk space. The database server stops adding new log files, writes an error message, and sets off an alarm.

To continue the transaction:
1. Add a dbspace or chunk to a dbspace.
2. Resume processing the transaction.

If you cannot add more disk space to the database server, end the transaction.

To end the transaction
• Issue the **onmode -z** command.
• Shut down and restart the database server.

When the database server comes up in fast-recovery mode, the transaction is rolled back. Then perform the following steps:

To recover from a long transaction hang
1. Add more disk space or another disk until the transaction is successfully rolled back.
2. Perform a point-in time restore to a time before the long transaction began or early enough for the database server to roll back the transaction.
3. Drop the extra log files, dbspaces, or chunks from the database server instance.
4. Perform a complete level-0 backup to free the logical-log space.

Chapter 15. Physical logging, checkpoints, and fast recovery

These topics cover the three procedures that the database server uses to achieve data consistency:

- Physical logging
- Checkpoints
- Fast recovery

The *physical log* is a set of disk pages where the database server stores an unmodified copy of the page called a *before-image*. *Physical logging* is the process of storing a before-image of a page that the database server is going to change. A *checkpoint* is a point when the database server synchronizes the pages on disk with the pages in the shared-memory buffers. *Fast recovery* is an automatic procedure that restores the database server to a consistent state after it goes offline under uncontrolled conditions.

These procedures ensure that multiple, logically related writes are recorded as a unit, and that data in shared memory is periodically made consistent with data on disk.

For the tasks to manage and monitor the physical log and checkpoints, see Chapter 16, "Manage the physical log," on page 16-1.

Critical sections

A *critical section* is a section of code (or machine instructions) that must be performed as a single unit. A critical section ensures the integrity of a thread by allowing it to run a series of instructions before it is swapped out.

Physical logging

Physical logging is the process of storing the pages that the database server is going to change before the changed pages are actually recorded on disk. Before the database server modifies certain pages in the shared-memory buffer pool, it stores before-images of the pages in the physical-log buffer in shared memory.

The database server maintains the before-image page in the physical-log buffer in shared memory for those pages until one or more page cleaners flush the pages to disk. The unmodified pages are available in case the database server fails or the backup procedure requires them to provide an accurate snapshot of the database server data. Fast recovery and database server backups use these snapshots.

The database server recycles the physical log at each checkpoint, except in the special circumstances. For more information about checkpoints, see "Checkpoints" on page 15-4.

Fast recovery use of physically-logged pages

After a failure, the database server uses the before-images of pages to restore these pages on the disk to their state at the last checkpoint. Then the database server uses the logical-log records to return all data to physical and logical consistency, up to the point of the most-recently completed transaction. "Fast recovery" on page 15-6 explains this procedure in more detail.

Backup use of physically-logged pages

When you perform a backup, the database server performs a checkpoint and coordinates with the physical log to identify the correct version of pages that belong on the backup. In a level-0 backup, the database server backs up all disk pages. For more details, see the *IBM Informix Backup and Restore Guide*.

Database server activity that is physically logged

If multiple modifications were made to a page between checkpoints, typically only the first before-image is logged in the physical log.

The physical log is a cyclical log in which the pages within the physical log are used once per checkpoint. If the RTO_SERVER_RESTART configuration parameter is set, additional physical logging occurs to improve fast recovery performance.

Physical recovery messages

When fast recovery begins, the database server logs the following message with the name of the chunk and offset:

```
Physical recovery started at page chunk:offset.
```

When the fast recovery completes, the database server logs the following message with the number of pages examined and restored:

```
Physical recovery complete: number pages examined, number pages restored.
```

Physical logging and simple large objects

The database server pages in the physical log can be any database server page, including simple large objects in table spaces (tblspaces). Even overhead pages (such as chunk free-list pages, blobspace free-map pages, and blobspace bitmap pages) are copied to the physical log before data on the page is modified and flushed to disk.

Blobspace blobpages are not logged in the physical log. For more information about blobspace logging, see "Log blobspaces and simple large objects" on page 13-6.

Physical logging and smart large objects

The user-data portion of smart large objects is not physically logged. However, the metadata is physically logged. For information about smart large objects, see "Sbspaces" on page 8-13.

Size and location of the physical log

These topics describe how to configure the size and location of the physical log.

Related reference:

"Change the physical-log location and size" on page 16-1

Specify the location of the physical log

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace. You have no initial control over this placement. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize disk contention), you can move the physical log out of the root dbspace to another dbspace, preferably to a disk that does not contain active tables or the logical-log files.

Recommendation: Locate critical dbspaces on fault-tolerant storage devices. If the storage that the physical log is located in is not fault-tolerant, use IBM Informix mirroring for the dbspace that contain the physical log. This protects the database if the storage device fails.

Strategy for estimating the size of the physical log

The size of the physical log depends on two factors: the rate at which transactions generate physical log activity and whether you set the RTO_SERVER_RESTART configuration parameter

The rate at which transactions generate physical log activity can affect checkpoint performance. During checkpoint processing, if the physical log starts getting too full as transactions continue to generate physical log data, the database server blocks transactions to allow the checkpoint to complete and to avoid a physical log overflow.

To avoid transaction blocking, the database server must have enough physical log space to contain all of the transaction activity that occurs during checkpoint processing. Checkpoints are triggered whenever the physical log becomes 75 percent full. When the physical log becomes 75 percent full, checkpoint processing must complete before the remaining 25 percent of the physical log is used. Transaction blocking occurs as soon as the system detects a potential for a physical log overflow, because every active transaction might generate physical log activity.

For example, suppose you have a one gigabyte physical log and 1000 active transactions. 1000 active transactions have the potential to generate approximately 80 megabytes of physical log activity if every transaction is in a critical section simultaneously. When 750 megabytes of the physical log fills, the database server triggers a checkpoint. If the checkpoint has not completed by the time the 920 megabytes of the physical log are used, transaction blocking occurs until the checkpoint completes. If transaction blocking takes place, the server automatically triggers more frequent checkpoints to avoid transaction blocking. You can disable the generation of automatic checkpoints.

The server might also trigger checkpoints if many dirty partitions exist, even if the physical log is not 75 percent full, because flushing the modified partition data to disk requires physical log space. When the server checks if the Physical Log is 75 percent full, the server also checks if the following condition is true:

```
(Physical Log Pages Used + Number of Dirty Partitions) >=
(Physical Log Size * 9) /10)
```

For more information about checkpoint processing and automatic checkpoints, see "Checkpoints" on page 15-4.

The second factor to consider when estimating the size of the physical log depends on your use of the RTO_SERVER_RESTART configuration parameter to specify a target amount of time for fast recovery. If you are not required to consider fast recovery time, you are not requires to enable the RTO_SERVER_RESTART configuration parameter. If you specify a value for the RTO_SERVER_RESTART configuration parameter, transaction activity generates additional physical log activity.

Typically, this additional physical log activity has little or no affect on transaction performance. The extra logging is used to assist the buffer pool during fast recovery, so that log replay performs optimally. If the physical log is considerably

larger than the combined sizes of all buffer pools, page flushing and page faulting occur during fast recovery. The page flushing and page faulting substantially reduce fast recovery performance, and the database server cannot maintain the RTO_SERVER_RESTART policy.

For systems with less that four gigabytes of buffer pool space, the physical log can be sized at 110 percent of the combined size of all the buffer pools. For larger buffer pools, start with four gigabytes of physical log space and then monitor checkpoint activity. If checkpoints occur too frequently and seem to affect performance, increase the physical log size.

A rare condition, called a physical-log overflow, can occur when the database server is configured with a small physical log and has many users. Following the previously described size guidelines helps avoid physical-log overflow. The database server generates performance warnings to the message log whenever it detects suboptimal configurations.

You can use the **onstat -g ckp** command to display configuration recommendations if a suboptimal configuration is detected.

Physical-log overflow when transaction logging is turned off

The physical log can overflow if you use simple large objects or smart large objects in a database with transaction logging turned off, as the following example shows.

When the database server processes simple large objects, each portion of the simple large object that the database server stores on disk can be logged separately, allowing the thread to exit the critical sections of code between each portion. However, if logging is turned off, the database server must carry out all operations on the simple large object in one critical section. If the simple large object is large and the physical log small, this scenario can cause the physical logs to become full. If this situation occurs, the database server sends the following message to the message log:

```
Physical log file overflow
```

The database server then initiates a shutdown. For the suggested corrective action, see your message log.

Checkpoints

Periodically, the database server flushes transactions and data within the buffer pool to disk. Until the transactions and data are flushed to disk, the data and transactions are in a state of flux. Instead of forcing every transaction to disk immediately after a transaction is completed, the database server writes transactions to the logical log. The database server logs the transactions as they occur. In the event of a system failure, the server:

- Replays the log to redo and restore the transactions.
- Returns the database to a state consistent with the state of the database system at the time of the failure.

To facilitate the restoration or logical recovery of a database system, the database server generates a consistency point, called a *checkpoint*. A checkpoint is a point in time in the log when a known and consistent state for the database system is established. Typically, a checkpoint involves recording a certain amount of information so that, if a failure occurs, the database server can restart at that established point.

The purpose of a checkpoint is to periodically move the restart point forward in the logical log. If checkpoints did not exist and a failure occurred, the database server would be required to process all the transactions that were recorded in the logical log since the system restarted.

A checkpoint can occur in one of these situations:

- When specific events occur. For example, a checkpoint occurs whenever a dbspace is added to the server or a database backup is performed.

  Typically, these types of events trigger checkpoints that block transaction processing. Therefore, these checkpoints are called *blocking* checkpoints.

- When resource limitations occur. For example, a checkpoint is required for each span of the logical log space to guarantee that the log has a checkpoint at which to begin fast recovery. The database server triggers a checkpoint when the physical log is 75 percent full to avoid physical log overflow.

  Checkpoints triggered by resource limitations usually do not block transactions. Therefore, these checkpoints are called *nonblocking checkpoints*.

  However, if the database server begins to run out of resources during checkpoint processing, transaction blocking occurs in the midst of checkpoint processing to make sure that the checkpoint completes before a resource is depleted. If transactions are blocked, the server attempts to trigger checkpoints more frequently to avoid transaction blocking during checkpoint processing. For more information, see "Strategy for estimating the size of the physical log" on page 15-3.

Automatic checkpoints cause the database server to trigger more frequent checkpoints to avoid transaction blocking. Automatic checkpoints attempt to monitor system activity and resource usage (physical and logical log usage along with how dirty the buffer pools are) to trigger checkpoints in a timely manner so that the processing of the checkpoint can complete before the physical or logical log is depleted.

The database server generates at least one automatic checkpoint for each span of the logical-log space. This guarantees the existence of a checkpoint where fast recovery can begin.

Use the AUTO_CKPTS configuration parameter to enable or disable automatic checkpoints when the database server starts. (You can dynamically enable or disable automatic checkpoints by using **onmode -wm** or **onmode -wf**.)

Manual checkpoints are event-based checkpoints that you can initiate.

The database server provides two methods for determining how long fast recovery takes in the event of an unplanned outage.

- Use the CKPTINTVL configuration parameter to specify how frequently the server triggers checkpoints.
- Use the RTO_SERVER_RESTART configuration parameter to specify how much time fast recovery takes.

  When you use the RTO_SERVER_RESTART configuration parameter:

  – The database server ignores the CKPTINTVL configuration parameter.
  – The database server monitors the physical and logical log usage to estimate the duration of fast recovery. If the server estimates that fast recovery exceeds the time specified in the RTO_SERVER_RESTART configuration parameter, the server automatically triggers a checkpoint.

The RTO_SERVER_RESTART configuration parameter is intended to be a target amount of time, not a guaranteed amount of time.

Several factors that can increase restart time can also influence fast recovery time. These factors include rolling back long transactions that were active at the time of an unplanned outage.

For more information about the RTO_SERVER_RESTART and AUTO_CKPTS configuration parameters, see the topics on configuration parameters in the *IBM Informix Administrator's Reference*.

LRU values for flushing a buffer pool between checkpoints

The LRU values for flushing a buffer pool between checkpoints are not critical for checkpoint performance. The **lru_max_dirty** and **lru_min_dirty** values, which are set in the BUFFERPOOL configuration parameter, are usually necessary only for maintaining enough clean pages for page replacement. Start by setting **lru_min_dirty** to 70 and **lru_max_dirty** to 80.

If transactions are blocked during a checkpoint, the database server subsequently attempts to increase checkpoint frequency to eliminate the transaction being blocked. When the server searches for a free page to perform page replacement and a foreground write occurs, the server subsequently automatically increases the LRU flushing frequency to prevent this event from occurring again. When the database server completes page replacement and finds a frequently accessed page, the server automatically increases LRU flushing. Any automatic adjustments to LRU flushing do not persist to the onconfig file.

For more information about monitoring and tuning checkpoint parameters and information about LRU tuning and adjustments, see the *IBM Informix Performance Guide*.

Checkpoints during backup

If you perform a backup, the database server runs a checkpoint and flushes all changed pages to the disk. If you perform a restore, the database server reapplies all logical-log records.

For information about ON-Bar or **ontape**, see the *IBM Informix Backup and Restore Guide*.

Fast recovery

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time that it moves from offline to quiescent, administration, or online mode. You are not required to take any administrative actions for fast recovery; it is an automatic feature.

The fast-recovery process checks if, the last time that the database server went offline, it did so in uncontrolled conditions. If so, fast recovery returns the database server to a state of physical and logical consistency.

If the fast-recovery process finds that the database server came offline in a controlled manner, the fast-recovery process terminates, and the database server moves to online mode.

See "Fast recovery after a checkpoint" on page 15-7.

Need for fast recovery

Fast recovery restores the database server to physical and logical consistency after any failure that results in the loss of the contents of memory for the database server. For example, the operating system fails without warning. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Situations when fast recovery is initiated

Every time that the administrator brings the database server to quiescent, administration, or online mode from offline mode, the database server checks to see if fast recovery is required.

As part of shared-memory initialization, the database server checks the contents of the physical log. The physical log is empty when the database server shuts down under control. The move from online mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if the database server finds pages in the physical log, the database server clearly went offline under uncontrolled conditions, and fast recovery begins.

Fast recovery and buffered logging

If a database uses buffered logging (as described in "Buffered transaction logging" on page 11-6), some logical-log records associated with committed transactions might not be written to the logical log at the time of the failure. If this occurs, fast recovery cannot restore those transactions. Fast recovery can restore only transactions with an associated COMMIT record stored in the logical log on disk. (For this reason, buffered logging represents a trade-off between performance and data vulnerability.)

Possible physical log overflow during fast recovery

During fast recovery, the physical log can overflow. If this occurs, the database server tries to extend the physical log space to a disk file named `plog_extend.`*`servernum`*. The default location of this file is `$INFORMIXDIR/tmp`.

Use the ONCONFIG parameter PLOG_OVERFLOW_PATH to define the location for creating this file.

The database server removes the `plog_extend.`*`servernum`* file when the first checkpoint is performed during a fast recovery.

Fast recovery and no logging

For databases or tables that do not use logging, fast recovery restores the database to its state at the time of the most recent checkpoint. All changes made to the database since the last checkpoint are lost.

Fast recovery after a checkpoint

Fast recovery returns the database server to a consistent state as part of shared-memory initialization. All committed transactions are restored, and all uncommitted transactions are rolled back.

Fast recovery occurs in the following steps:

1. The database server uses the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint. This point is known as *physical consistency*.

2. The database server locates the most recent checkpoint record in the logical-log files.
3. The database server rolls forward all logical-log records written after the most recent checkpoint record.
4. The database server rolls back all uncommitted transactions. Some XA transactions might be unresolved until the XA resource manager is available.

The server returns to the last-checkpoint state

To return all disk pages to their condition at the time of the most recent checkpoint, the database server writes the before-images stored in the physical log to shared memory and then back to disk. Each before-image in the physical log contains the address of a page that was updated after the checkpoint. When the database server writes each before-image page in the physical log to shared memory and then back to disk, changes to the database server data since the time of the most recent checkpoint are undone. The database server is now physically consistent. The following figure illustrates this step.
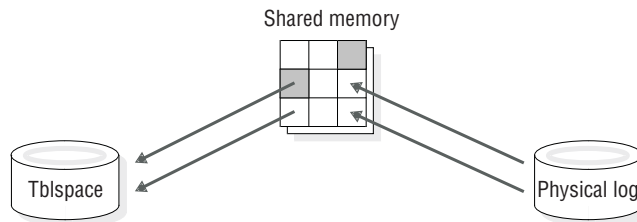
Figure 15-1. Writing all remaining before-images in the physical log back to disk

The server locates the checkpoint record in the logical log

After returning to the last checkpoint state, the database server locates the address of the most recent checkpoint record in the logical log. The most recent checkpoint record is guaranteed to be in the logical log on disk.

The server rolls forward logical-log records

After locating the checkpoint record in the logical log, the database server rolls forward the logical-log records that were written after the most recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point at which the uncontrolled shutdown occurred. The following figure illustrates this step.
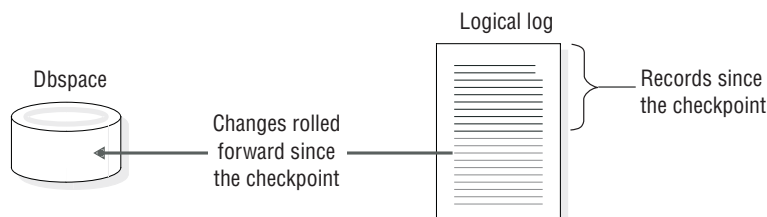
Figure 15-2. Rolling forward the logical-log records written since the most recent checkpoint

The server rolls back uncommitted transactions

After rolling the logical-log records forward, the database server rolls back all logical-log records for transactions that were not committed at the time the system failed. All databases are logically consistent because all committed transactions are rolled forward and all uncommitted transactions are rolled back. Some XA transactions might be unresolved until the XA resource manager is available.

Transactions that have completed the first phase of a two-phase commit are exceptional cases. For more information, see "How the two-phase commit protocol handles failures" on page 25-8.

Because one or more transactions possibly spanned several checkpoints without being committed, this rollback procedure might read backward through the logical log, past the most recent checkpoint record. All logical-log files that contain records for open transactions are available to the database server because a log file is not freed until all transactions that it contains are closed.

The following figure illustrates the rollback procedure. Here, uncommitted changes are rolled back from the logical log to a dbspace on a particular disk. When fast recovery is complete, the database server returns to quiescent, administration, or online mode.
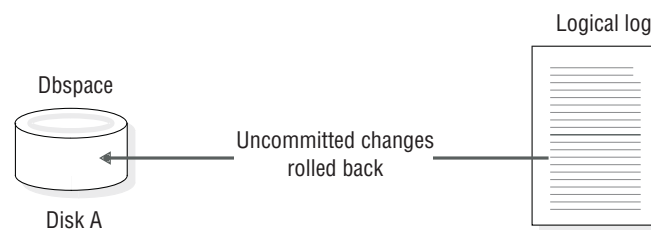
Figure 15-3. Rolling back all incomplete transactions

Chapter 16. Manage the physical log

These topics describe the following procedures:

- Changing the location and size of the physical log
- Monitoring the physical log, physical-log buffers, and logical-log buffers
- Monitoring and forcing checkpoints

See Chapter 15, "Physical logging, checkpoints, and fast recovery," on page 15-1 for background information.

Change the physical-log location and size

You can change use the **onparams** utility to change the location and size of the physical log.

You can move the physical-log file to try to improve performance. When the database server initializes disk space, it places the disk pages allocated for the logical log and the physical log in the root dbspace. You might improve performance by moving the physical log, the logical-log files, or both to other dbspaces.

Restriction: You cannot add logical or physical logs to dbspaces that have non-default page sizes.

Prerequisites:

- Log-in as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows when you make the changes.
- Determine if adequate contiguous space is available by running the **oncheck -pe** command.

  The space allocated for the physical log must be contiguous. When changing the physical log size or location, if the target dbspace does not contain adequate contiguous space, the server does not change the physical log. Additionally, if insufficient resources for the physical log exist when you initialize the database server, the initialization fails.

To change the size and location of the physical log, run the following command while the database server is in administration, quiescent, or online mode:

```
onparams -p -s size -d dbspace -y
```

size   The new size of the physical log in KB

dbspace
         Specifies the dbspace where the physical log is to be located

The following example changes the size and location of the physical log. The new physical-log size is 400 KB, and the log is located in the **dbspace6** dbspace:

```
onparams -p -s 400 -d dbspace6 -y
```

Related reference:

"Monitor physical and logical-logging activity"

"Size and location of the physical log" on page 15-2

Monitor physical and logical-logging activity

Monitor the physical log to determine the percentage of the physical-log file that gets used before a checkpoint occurs. You can use this information to find the optimal size of the physical-log file. It must be large enough that the database server is not required to force checkpoints too frequently and small enough to conserve disk space and guarantee fast recovery.

Monitor physical-log and logical-log buffers to determine if they are the optimal size for the current level of processing. The important statistic to monitor is the pages-per-disk-write statistic. For more information about tuning the physical-log and logical-log buffers, see your *IBM Informix Performance Guide*.

To monitor the physical-log file, physical-log buffers, and logical-log buffers, use the following commands.

Utility	Command	Additional information
Command line Command line or ISA	**onstat -l**	The first line displays the following information for each physical-log buffer: • The number of buffer pages used (**bufused**) • The size of each physical log buffer in pages (**bufsize**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**) The second line displays the following information about the physical log: • The page number of the first page in the physical-log file (**phybegin**) • The size of the physical-log file in pages (**physize**) • The current position in the log where the next write occurs, specified as a page number (**physpos**) • The number of pages in the log that have been used (**phyused**) • The percentage of the total physical-log pages that have been used (**%used**) The third line displays the following information about each logical-log buffer: • The number of buffer pages used (**bufused**) • The size of each logical-log buffer in pages (**bufsize**) • The number of records written to the buffer (**numrecs**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of records to pages in the buffer (**recs/pages**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)

Utility	Command	Additional information
Command line Command line or ISA	**onparams -p**	Moves or resizes the physical log.
Command line Command line or ISA	**onmode -l**	Advances to the next logical-log file.
ISA	**Logs** > **Logical**	Click **Advance Log File**.

For more information about and an example of **onstat -l** output, see the *IBM Informix Administrator's Reference*.

For information about using SQL administration API commands instead of some **onparams** and **onmode** commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Guide to SQL: Syntax*.

Related reference:

"Change the physical-log location and size" on page 16-1

Monitor checkpoint information

Monitor checkpoint activity to view information that includes the number of times that threads were required to wait for the checkpoint to complete. This information is useful for determining if the checkpoint interval is appropriate.

To monitor checkpoints, use the following commands.

Utility	Command	Additional Information
The **onstat** utility	**onstat -m**	View the last 20 lines in the message log. If a checkpoint message is not in the last 20 lines, read the message log directly with a text editor. The database server writes individual checkpoint messages to the log when the checkpoint ends. If a checkpoint occurs, but the database server has no pages to write to disk, the database server does not write any messages to the message log.
The **onstat** utility	**onstat -p**	Obtains these checkpoint statistics: • **numckpts**: Number of checkpoints that occurred since the database server was brought online. • **ckptwaits**: Number of times that a user thread waits for a checkpoint to finish. The database server prevents a user thread from entering a critical section during a checkpoint.
ON-Monitor (UNIX)	**Status** > **Profile**	The **Checkpoints** and **Check Waits** fields display the same information as the **numckpts** and **ckpwaits** fields in **onstat -p**.

For information about tuning the checkpoint interval, see your *IBM Informix Performance Guide*.

Turn checkpoint tuning on or off

To turn automatic checkpoint tuning on, issue an **onmode –wf AUTO_CKPTS=1** command. To turn automatic checkpoint tuning off, issue an **onmode –wf AUTO_CKPTS=0** command.

Force a checkpoint

When necessary, you can force a checkpoint with an **onmode** or SQL administration API command.

Force a checkpoint in any of the following situations:

- To free a logical-log file that contains the most recent checkpoint record and that is backed up but not yet released (**onstat -l** status of U-B-L or U-B)
- Before you issue **onmode -sy** to place the database server in quiescent mode
- After building a large index, if the database server terminates before the next checkpoint. The index build restarts the next time that you restart the database server.
- If a checkpoint has not occurred for a long time and you are about to attempt a system operation that might interrupt the database server
- If foreground writes are taking more resources than you want (force a checkpoint to bring this down to zero temporarily)
- Before you run **dbexport** or unload a table, to ensure physical consistency of all data before you export or unload it
- After you perform a large load of tables using PUT or INSERT statements (Because table loads use the buffer cache, forcing a checkpoint cleans the buffer cache.)

To force a checkpoint, run **onmode -c**.

Alternatively, if you use ON-Monitor (UNIX), choose the **Force-Ckpt** option from the main menu. The time in the **Last Checkpoint Done** field does not change until a checkpoint occurs. The **Last Checkpoint Check** field shows the time of the last checkpoint check. If no modifications have been made since the time of the last checkpoint, the database server does not perform a checkpoint.

For information about using SQL administration API commands instead of some **onmode** commands, see Chapter 28, "Remote administration with the SQL administration API," on page 28-1 and the *IBM Informix Guide to SQL: Syntax*.

Server-provided checkpoint statistics

The database server provides history information about the previous twenty checkpoints. You can access this information through the SMI **sysckptinfo** table.

Related reference:

sysckptinfo (Administrator's Reference)

SMI tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The **sysprofile** table also provides the same checkpoint statistics that are available from the **onstat -p** option. These rows contain the following statistics.

plgpagewrites
> Number of pages written to the physical-log buffer

plgwrites
> Number of writes from the physical-log buffer to the physical log file

llgrecs Number of records written to the logical-log buffer

llgpagewrites
> Number of pages written to the logical-log buffer

llgwrites
> Number of writes from the logical-log buffer to the logical-log files

numckpts
> Number of checkpoints that have occurred since the database server was brought online)

ckptwaits
> Number of times that threads waited for a checkpoint to finish entering a critical section during a checkpoint

value Values for **numckpts** and **ckptwaits**

Turn automatic LRU tuning on or off

Use the AUTO_LRU_TUNING configuration parameter to enable or disable automatic LRU tuning when the database server starts.

If the RTO_SERVER_RESTART configuration parameter is set, the database server automatically triggers checkpoints so that it can bring the server online within the specified time. The database server prints warning messages in the message log if the server cannot meet the RTO_SERVER_RESTART policy.

To turn off automatic LRU tuning for a particular session, issue an **onmode –wm AUTO_LRU_TUNING=**_0_ command.

To turn on automatic LRU tuning after turning it off during a session, issue an **onmode –wm AUTO_LRU_TUNING=**_1_ command

Automatic LRU tuning changes affect all buffer pools and adjust **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

For more information about LRU tuning, see the _IBM Informix Performance Guide_.

Part 4. Fault tolerance

Chapter 17. Mirroring

These topics describe the database server mirroring feature. For instructions on how to perform mirroring tasks, see Chapter 18, "Using mirroring," on page 18-1.

Related concepts:

"Chunks" on page 8-2

Mirroring

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace, blobspace, or sbspace with an equal-sized *mirror chunk*.

Every write to the primary chunk is automatically accompanied by an identical write to the mirror chunk. This concept is illustrated in the following figure. If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirror chunk until you can recover the primary chunk, all without interrupting user access to data.
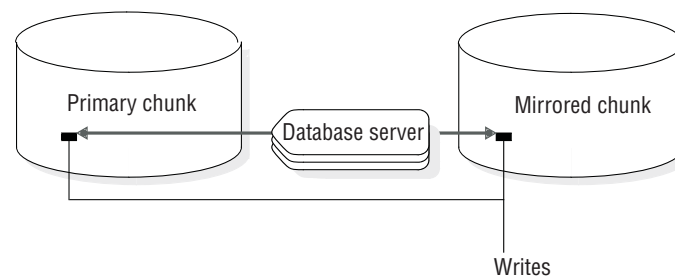
Figure 17-1. Writing data to both the primary chunk and the mirror chunk

Mirroring is not supported on disks that are managed over a network. The same database server instance must manage all the chunks of a mirrored set.

Benefits of mirroring

If media failure occurs, mirroring provides the database server administrator with a means of recovering data without taking the database server offline. This feature results in greater reliability and less system downtime. Furthermore, applications can continue to read from and write to a database whose primary chunks are on the affected media, provided that the chunks that mirror this data are located on separate media.

Any critical database must be located in a mirrored dbspace. The root dbspace, which contains the database server reserved pages, must be mirrored.

Costs of mirroring

Disk-space costs and performance costs are associated with mirroring. The disk-space cost is due to the additional space required for storing the mirror data. The performance cost results from performing writes to both the primary and mirror chunks. The use of multiple virtual processors for disk writes reduces this performance cost. The use of *split reads*, whereby the database server reads data from either the primary chunk or the mirror chunk, depending on the location of the data within the chunk, actually causes performance to improve for read-only

data. For more information about how the database server performs reads and writes for mirror chunks, see "Actions during processing" on page 17-4.

Consequences of not mirroring

If you do not mirror your dbspaces, the frequency with which you must restore from a storage-space backup after media failure increases.

When a mirror chunk suffers media failure, the database server reads exclusively from the chunk that is still online until you bring the down chunk back online. When the second chunk of a mirrored pair goes down, the database server cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, the database server goes offline immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, the database server can continue to operate, but threads cannot read from or write to the down chunk. If an unmirrored chunk goes down, you must restore it by recovering the dbspace from a backup.

Data to mirror

Ideally, you must mirror all of your data. If disk space is an issue, however, you might not be able to do so. In this case, select certain critical chunks to mirror.

Critical chunks always include the chunks that are part of the root dbspace, the chunk that stores the logical-log files, and the chunk that stores the physical logs. If any one of these critical chunks fail, the database server goes offline immediately.

If some chunks hold data that is critical to your business, give these chunks high priority for mirroring.

Also give priority for mirroring to chunks that store frequently used data. This action ensures that the activities of many users would not be halted if one widely used chunk goes down.

Alternatives to mirroring

Mirroring, as explained in this manual, is a database server feature. Your operating system or hardware might provide alternative mirroring solutions.

If you are considering a mirroring feature provided by your operating system instead of database server mirroring, compare the implementation of both features before you decide which to use. The slowest step in the mirroring process is the actual writing of data to disk. The database server strategy of performing writes to mirror chunks in parallel helps to reduce the time required for this step. (See "Disk writes to mirror chunks" on page 17-4.) In addition, database server mirroring uses split reads to improve read performance. (See "Disk reads from mirror chunks" on page 17-5.) Operating-system mirroring features that do not use parallel mirror writes and split reads might provide inferior performance.

Nothing prevents you from running database server mirroring and operating-system mirroring at the same time. They run independently of each other. In some cases, you might decide to use both the database server mirroring and the mirroring feature provided by your operating system. For example, you might have both database server data and other data on a single disk drive. You can use the operating-system mirroring to mirror the other data and database server mirroring to mirror the database server data.

Logical volume managers

Logical volume managers are an alternative mirroring solution. Some operating-system vendors provide this type of utility to have multiple disks seem to be one file system. Saving data to more than two disks gives you added protection from media failure, but the additional writes have a performance cost.

Hardware mirroring

Another solution is to use hardware mirroring such as redundant array of inexpensive disks (RAID). An advantage of this type of hardware mirroring is that it requires less disk space than database server mirroring does to store the same amount of data to prevent media failure.

Some hardware mirroring systems support *hot swapping*. You can swap a bad disk while keeping the database server online. Reducing I/O activity before performing a hot swap is recommended.

Important: If problems occur with the database server while using hardware mirroring, see the operating-system or disk documentation or technical support for assistance.

External backup and restore

If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional ON-Bar commands. For more information about external backup and restore, see the *IBM Informix Backup and Restore Guide*.

Mirroring process

This section describes the mirroring process in greater detail. For instructions on how to perform mirroring operations such as creating mirror chunks, starting mirroring, changing the status of mirror chunks, and so forth, see Chapter 18, "Using mirroring," on page 18-1.

Creation of a mirror chunk

When you specify a mirror chunk, the database server copies all the data from the primary chunk to the mirror chunk. This copy process is known as *recovery*. Mirroring begins as soon as recovery is complete.

The recovery procedure that marks the beginning of mirroring is delayed if you start to mirror chunks within a dbspace that contains a logical-log file. Mirroring for dbspaces that contain a logical-log file does not begin until you create a level-0 backup of the root dbspace. The delay ensures that the database server can use the mirrored logical-log files if the primary chunk that contains these logical-log files becomes unavailable during a dbspace restore.

The level-0 backup copies the updated database server configuration information, including information about the new mirror chunk, from the root dbspace reserved pages to the backup. If you perform a data restore, the updated configuration information at the beginning of the backup directs the database server to look for the mirrored copies of the logical-log files if the primary chunk becomes unavailable. If this new storage-space backup information does not exist, the database server is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbspace that contains a logical-log file while a dbspace backup is being created. The new information that must be in the first block of the dbspace backup tape cannot be copied there after the backup has begun.

For more information about creating mirror chunks, see Chapter 18, "Using mirroring," on page 18-1.

Mirror status flags

Dbspaces, blobspaces, and sbspaces have status flags that indicate whether they are mirrored or unmirrored.

You must perform a level-0 backup of the root dbspace before mirroring starts.

Chunks have status flags that indicate the following information:
- Whether the chunk is a primary or mirror chunk
- Whether the chunk is currently online, down, a new mirror chunk that requires a level-0 backup of the root dbspace, or in the process of being recovered

For descriptions of these chunk status flags, see the description of the **onstat -d** option in the *IBM Informix Administrator's Reference*. For information about how to display these status flags, see "Monitor disk usage" on page 9-44.

Recovery

When the database server recovers a mirror chunk, it performs the same recovery procedure that it uses when mirroring begins. The mirror-recovery process consists of copying the data from the existing online chunk onto the new, repaired chunk until the two are identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the online chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives online status. You perform the same steps whether you are recovering the primary chunk of a mirrored pair or recovering the mirror chunk.

Tip: You can still use the online chunk during the recovery process. If data is written to a page that has already been copied to the recovery chunk, the database server updates the corresponding page on the recovery chunk before it continues with the recovery process.

For information about how to recover a down chunk, see the information about recovering a mirror chunk on page "Recover a mirror chunk" on page 18-6.

Actions during processing

These topics explain some of the details of disk I/O for mirror chunks and how the database server handles media failure for these chunks.

Disk writes to mirror chunks

During database server processing, the database server performs mirroring by executing two parallel writes for each modification: one to the primary chunk and one to the mirror chunk.

Disk reads from mirror chunks

The database server uses mirroring to improve read performance because two versions of the data are located on separate disks. A data page is read from either the primary chunk or the mirror chunk, depending on which half of the chunk includes the address of the data page. This feature is called a *split read*. Split reads improve performance by reducing the disk-seek time. Disk-seek time is reduced because the maximum distance over which the disk head must travel is reduced by half. The following figure illustrates a split read.
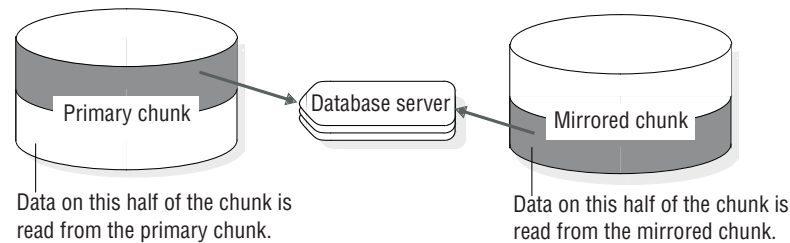
Figure 17-2. Split read reducing the maximum distance over which the disk head must travel

Detection of media failures

The database server checks the return code when it first opens a chunk and after any read or write. Whenever the database server detects that a primary (or mirror) chunk device has failed, it sets the chunk-status flag to down (D). For information about chunk-status flags, see "Mirror status flags" on page 17-4.

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains online. This statement is true even if the administrator intentionally brings down one of the chunks.

After the administrator recovers the down chunk and returns it to online status, reads are again split between the primary and mirror chunks, and writes are made to both chunks.

Chunk recovery

The database server uses asynchronous I/O to minimize the time required for recovering a chunk. The read from the chunk that is online can overlap with the write to the down chunk, instead of the two processes occurring serially. That is, the thread that performs the read is not required to wait until the thread that performs the write has finished before it reads more data.

Result of stopping mirroring

When you end mirroring, the database server immediately frees the mirror chunks and makes the space available for reallocation. The action of ending mirroring takes only a few seconds.

Create a level-0 backup of the root dbspace after you end mirroring to ensure that the reserved pages with the updated mirror-chunk information are copied to the backup. This action prevents the restore procedure from assuming that mirrored data is still available.

Structure of a mirror chunk

The mirror chunk contains the same control structures as the primary chunk, as follows:

- Mirrors of blobspace chunks contain blobspace overhead pages.
- Mirrors of dbspace chunks contain dbspace overhead pages.
- Mirrors of sbspaces contain metadata pages.

For information about these structures, see the section on the structure of a mirror chunk in the disk structures and storage chapter of the *IBM Informix Administrator's Reference*.

A display of disk-space use, provided by one of the methods explained under "Monitor chunks" on page 9-44, always indicates that the mirror chunk is *full*, even if the primary chunk has free space. The full mirror chunk indicates that none of the space in the chunk is available for use other than as a mirror of the primary chunk. The status remains full for as long as both primary chunk and mirror chunk are online.

If the primary chunk goes down and the mirror chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Chapter 18. Using mirroring

These topics describe the various mirroring tasks that are required to use the database server mirroring feature. It provides an overview of the steps required for mirroring data.

Preparing to mirror data

This section describes how to start mirroring data on a database server that is not running with the mirroring function enabled.

To prepare to mirror data:

1. Take the database server offline and enable mirroring.

   See "Enable the MIRROR configuration parameter."

2. Bring the database server back online.

3. Allocate disk space for the mirror chunks.

   You can allocate this disk space at any time, as long as the disk space is available when you specify mirror chunks in the next step. The mirror chunks must be on a different disk than the corresponding primary chunks. See "Allocate disk space for mirrored data" on page 18-2.

4. Choose the dbspace, blobspace, or sbspace that you want to mirror, and specify a mirror-chunk path name and offset for each primary chunk in that storage space.

   The mirroring process starts after you perform this step. Repeat this step for all the storage spaces that you want to mirror. See "Using mirroring" on page 18-2.

Enable the MIRROR configuration parameter

You can set the MIRROR configuration parameter to enable (or disable) mirroring.

Enabling mirroring starts the database server functions required for mirroring tasks. However, when you enable mirroring, you do not initiate the mirroring process. Mirroring does not actually start until you create mirror chunks for a dbspace, blobspace, or sbspace. See "Using mirroring" on page 18-2.

Enable mirroring when you initialize the database server if you plan to create a mirror for the root dbspace as part of initialization; otherwise, leave mirroring disabled. If you later decide to mirror a storage space, you can change the value of the MIRROR configuration parameter.

To enable mirroring for the database server, you must set the MIRROR parameter in `onconfig` to 1. The default value of MIRROR is 0, indicating that mirroring is disabled.

Do not set the MIRROR parameter to 1 if you are not using mirroring.

To change the value of MIRROR, you can edit the `onconfig` file with a text editor while the database server is in online mode. After you change the `onconfig` file, take the database server offline and then to quiescent for the change to take effect.

Allocate disk space for mirrored data

Before you can create a mirror chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirror chunks. For an explanation of allocating disk space, see "Allocate disk space" on page 9-1.

Always allocate disk space for a mirror chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. You can use this setup to access the mirror chunk if the disk on which the primary chunk is located goes down, or vice versa.

Link chunks (UNIX)

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirror chunks to mirror path names. If a disk failure occurs, you can link a new file or raw device to the path name, eliminating the necessity to physically replace the disk that failed before the chunk is brought back online.

Relink a chunk to a device after a disk failure

On UNIX, if the disk on which the actual mirror file or raw device is located goes down, you can relink the chunk to a file or raw device on a different disk. If you do this, you can recover the mirror chunk before the disk that failed is brought back online. Typical UNIX commands that you can use for relinking are shown in the following examples.

The original setup consists of a primary root chunk and a mirror root chunk, which are linked to the actual raw disk devices, as follows:

```
ln -l
lrwxrwxrwx 1 informix 10 May 3 13:38 /dev/root@->/dev/rxy0h
lrwxrwxrwx 1 informix 10 May 3 13:40 /dev/mirror_root@->/dev/rsd2b
```

Assume that the disk on which the raw device /dev/rsd2b is located has gone down. You can use the **rm** command to remove the corresponding symbolic link, as follows:

```
rm /dev/mirror_root
```

Now you can relink the mirror chunk path name to a raw disk device, on a disk that is running, and proceed to recover the chunk, as follows:

```
ln -s /dev/rab0a /dev/mirror_root
```

Using mirroring

Mirroring starts when you create a mirror chunk for each primary chunk in a dbspace, blobspace, or sbspace.

When you create a mirror chunk, the database server copies data from the primary chunk to the mirror chunk. When this process is complete, the database server begins mirroring data. If the primary chunk contains logical-log files, the database server does not copy the data immediately after you create the mirror chunk but waits until you perform a level-0 backup. For an explanation of this behavior see "Creation of a mirror chunk" on page 17-3.

Important: You must always start mirroring for an entire dbspace, blobspace, or sbspace. The database server does not permit you to select particular chunks in a dbspace, blobspace, or sbspace to mirror. You must create mirror chunks for every chunk in the space.

You start mirroring a storage space when you perform the following operations:
- Create a mirrored root dbspace during system initialization
- Change the status of a dbspace from unmirrored to mirrored
- Create a mirrored dbspace, blobspace, or sbspace

Each of these operations requires you to create mirror chunks for the existing chunks in the storage space.

Mirroring the root dbspace during initialization

If you enable mirroring when you initialize the database server, you can also specify a mirror path name and offset for the root chunk. The database server creates the mirror chunk when the server is initialized. However, because the root chunk contains logical-log files, mirroring does not actually start until you perform a level-0 backup.

To specify the root mirror path name and offset, set the values of MIRRORPATH and MIRROROFFSET in the `onconfig` file before you start the database server.

If you do not provide a mirror path name and offset, but you do want to start mirroring the root dbspace, you must change the mirroring status of the root dbspace after the database server is initialized.

Change the mirror status

You can make the following two changes to the status of a mirror chunk:
- Change a mirror chunk from online to down
- Change a mirror chunk from down to recovery

You can take down or restore a chunk only if it is part of a mirrored pair. You can take down either the primary chunk or the mirror chunk, as long as the other chunk in the pair is online.

For information about how to determine the status of a chunk, see "Monitor disk usage" on page 9-44.

Manage mirroring

You can use the **onspaces** utility to manage mirroring.

On UNIX, you can also use ON-Monitor to manage mirroring.

For a full description of the **onspaces** syntax, see The **onspaces** utility in the *IBM Informix Administrator's Reference*.

Start mirroring for unmirrored storage spaces

You can prepare mirroring for a dbspace, blobspace, or sbspace at any time. However, the mirroring does not start until you perform a level-0 backup.

Start mirroring for unmirrored dbspaces using onspaces

You can use the **onspaces** utility to start mirroring a dbspace, blobspace, or sbspace. For example, the following **onspaces** command starts mirroring for the dbspace **db_project**, which contains two chunks, **data1** and **data2**:

```
onspaces -m db_project\
-p /dev/data1 -o 0 -m /dev/mirror_data1 0\
-p /dev/data2 -o 5000 -m /dev/mirror_data2 5000
```

The following example shows how to turn on mirroring for a dbspace called **sp1**. You can either specify the primary path, primary offset, mirror path, and mirror offset in the command or in a file.

```
onspaces -m sp1 -f mirfile
```

The `mirfile` file contains the following line:

```
/ix/9.3/sp1 0 /ix/9.2/sp1mir 0
```

In this line, `/ix/9.3/sp1` is the primary path, `0` is the primary offset, `/ix/9.3/sp1mir` is the mirror path, and `0` is the mirror offset.

Starting mirroring using ISA

To start mirroring with IBM Informix Server Administrator (ISA):

1. Select **Storage** > **Chunks**.
2. Select the dbspace name and click **Start Mirroring**.

Start mirroring for unmirrored dbspaces with ON-Monitor (UNIX)

Use the ON-Monitor **Dbspaces** > **Mirror** option to start mirroring a dbspace.

To select the dbspace that you want to mirror, move the cursor down the list to the correct dbspace and press **CTRL-B**. The **Mirror** option then displays a screen for each chunk in the dbspace. You can enter a **mirror** path name and offset in this screen.

After you enter information for each chunk, press **ESC** to exit the option. The database server recovers the new mirror chunks, meaning it copies data from the primary to the mirror chunk. If the chunk contains logical-log files, recovery is postponed until after you create a level-0 backup.

Start mirroring for new storage spaces

You can also start mirroring when you create a new dbspace, blobspace, or sbspace.

Start mirroring for new spaces using onspaces

You can use the **onspaces** utility to create a mirrored dbspace. For example, the following command creates the dbspace **db_acct** with an initial chunk **/dev/chunk1** and a mirror chunk /dev/mirror_chk1:

```
onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

Another way to start mirroring is to select **Index by Utility** > **onspaces -m**.

Starting mirroring for new spaces using ISA

To start mirroring for new storage spaces with IBM Informix Server Administrator (ISA):

1. Select **Storage** > **Spaces**.
2. Click **Add Dbspace**, **Add Blobspace**, or **Add Sbspace**.

3. Enter the path and offset for the mirror chunk.

Start mirroring for new dbspaces using ON-Monitor (UNIX)

To create a dbspace with mirroring, choose the **Dbspaces** > **Create** option.

This option displays a screen in which you can specify the path name, offset, and size of a primary chunk and the path name and offset of a mirror chunk for the new dbspace.

Add mirror chunks

If you add a chunk to a dbspace, blobspace, or sbspace that is mirrored, you must also add a corresponding mirror chunk.

Add mirror chunks using onspaces

You can use the **onspaces** utility to add a primary chunk and its mirror chunk to a dbspace, blobspace, or sbspace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Because the dbspace is mirrored, a mirror chunk, **mirror_chk2**, is also added.

```
onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```

Adding mirror chunks using ISA

To add mirror chunks with IBM Informix Server Administrator (ISA):

1. Select **Storage** > **Chunks**.
2. Select the dbspace name and click **Add Chunk**.
3. Enter the path and offset for the mirror chunk.

Add mirror chunks using ON-Monitor (UNIX)

To add mirror chunks, choose the **Dbspaces** > **Add-chunk** option.

The **Dbspaces** > **Add-chunk** option displays fields in which to enter the primary-chunk path name, offset, and size, and the mirror-chunk path name and offset.

Take down a mirror chunk

When a mirror chunk is down, the database server cannot write to it or read from it. You might take down a mirror chunk to relink the chunk to a different device. (See "Relink a chunk to a device after a disk failure" on page 18-2.)

Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbspace, which causes the database server to drop all the mirror chunks for that dbspace.

Take down mirror chunks using onspaces

You can use the **onspaces** utility to take down a chunk. The following example takes down a chunk that is part of the dbspace **db_acct**:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -D
```

Take down mirror chunks using ON-Monitor (UNIX)

To use ON-Monitor to take down a mirror chunk, choose the **Dbspaces** > **Status** option.

With the cursor on the dbspace that contains the chunk that you want to take down, press **F3** or **CTRL-B**.

The database server displays a screen that lists all the chunks in the dbspace. Move the cursor to the chunk that you want to take down and press **F3** or **CTRL-B** to change the status (take it down).

Recover a mirror chunk

To begin mirroring the data in the chunk that is online, you must recover the down chunk.

Recover a mirror chunk using onspaces

You can use the **onspaces -s** utility to recover a down chunk. For example, to recover a chunk that has the path name /dev/mirror_chk1 and an offset of 0 KB, issue the following command:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```

Recover a mirror chunk using ISA

To recover a mirror chunk with IBM Informix Server Administrator (ISA), select **Index by Utility** > **onspaces -s**.

Recover a mirror chunk using ON-Monitor (UNIX)

To use ON-Monitor to recover a mirror chunk that is down, choose the **Dbspaces** > **Status** option.

End mirroring

When you end mirroring for a dbspace, blobspace, or sbspace, the database server immediately releases the mirror chunks of that space. These chunks are immediately available for reassignment to other storage spaces.

Only users **informix** and **root** on UNIX or members of the **Informix-Admin** group on Windows can end mirroring.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in online mode when you end mirroring.

End mirroring using onspaces

You can end mirroring with the **onspaces** utility. For example, to end mirroring for the root dbspace, enter the following command:

```
onspaces -r rootdbs
```

Another way to end mirroring is to select **Index by Utility** > **onspaces -r**.

End mirroring using ON-Monitor (UNIX)

To end mirroring for a dbspace or blobspace with ON-Monitor, select the **Dbspaces** > **Mirror** option.

Select a dbspace or blobspace that is mirrored and press **CTRL-B** or **F3**.

Ending mirroring using ISA

To end mirroring with IBM Informix Server Administrator (ISA):

1. Select **Storage** > **Chunks**.
2. Select the dbspace name and click **Stop Mirroring**.

Chapter 19. Consistency checking

IBM Informix database servers are designed to detect database server malfunctions or problems caused by hardware or operating-system errors. It detects problems by performing *assertions* in many of its critical functions. An assertion is a consistency check that verifies that the contents of a page, structure, or other entity match what would otherwise be assumed.

When one of these checks finds that the contents are incorrect, the database server reports an assertion failure and writes text that describes the check that failed in the database server message log. The database server also collects further diagnostics information in a separate file that might be useful to IBM Informix Technical Support staff.

These topics provide an overview of consistency-checking measures and ways of handling inconsistencies.

Perform periodic consistency checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, you must periodically take the following actions:
- Verify that all data and the database server overhead information is consistent.
- Check the message log for assertion failures while you verify consistency.
- Create a level-0 dbspace backup after you verify consistency.

The following topics describe each of these actions.

Verify consistency

Because of the time required for this check and the possible contention that the check can cause, schedule this check for times when activity is at its lowest. You must perform this check just before you create a level-0 backup.

Run the commands shown in the following table as part of the consistency check.

Table 19-1. Checking data consistency

Type of validation	Command
System catalog tables	**oncheck -cc**
Data	**oncheck -cD** *dbname*
Extents	**oncheck -ce**
Indexes	**oncheck -cI** *dbname*
Reserved pages	**oncheck -cr**
Logical logs and reserved pages	**oncheck -cR**
Metadata and smart large objects	**oncheck -cs**

You can run each of these commands while the database server is in online mode. For information about how each command locks objects as it checks them and which users can perform validations, see **oncheck** in the *IBM Informix Administrator's Reference*.

In most cases, if one or more of these validation procedures detects an error, the solution is to restore the database from a dbspace backup. However, the source of the error might also be your hardware or operating system.

Validate system catalog tables

To validate system catalog tables, use the **oncheck -cc** command.

Each database contains its own system catalog, which contains information about the database tables, columns, indexes, views, constraints, stored procedures, and privileges.

If a warning is displayed when validation completes, its only purpose is to alert you that no records of a specific type were found. These warnings do not indicate any problem with your data, your system catalog, or even your database design. This warning indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**. For example, the following warning might be displayed if you validate system catalog tables for a database that has no synonyms defined for any table:

```
WARNING: No syssyntable records found.
```

However, if you receive an error message when you validate system catalog tables, the situation is quite different. Contact IBM Informix Technical Support immediately.

Validate data pages

To validate data pages, use the **oncheck -cD** command.

If data-page validation detects errors, try to unload the data from the specified table, drop the table, recreate the table, and reload the data. For information about loading and unloading data, see the *IBM Informix Migration Guide*. If this procedure does not succeed, perform a data restore from a storage-space backup.

Validate extents

To validate extents in every database, use the **oncheck -ce** command.

Extents must not overlap. If this command detects errors, perform a data restore from a storage-space backup.

Validate indexes

If an index is corrupted, the database server cannot use it in queries.

You can validate indexes on each of the tables in the database by using the **oncheck -cI** command.

In addition, the Scheduler task **bad_index_alert** looks for indexes that have been marked as corrupted by the server. This task runs nightly. An entry is made into the **sysadmin:ph_alert** table for each corrupted index found by the task.

If an index is corrupted, drop and recreate it.

Validate logical logs

To validate logical logs and the reserved pages, use the **oncheck -cR** command.

Validate reserved pages

To validate reserved pages, use the **oncheck -cr** command.

Reserved pages are pages that are located at the beginning of the initial chunk of the root dbspace. These pages contain the primary database server overhead information. If this command detects errors, perform a data restore from storage-space backup.

This command might provide warnings. In most cases, these warnings call your attention to situations of which you are already aware.

Validate metadata

Run **oncheck -cs** for each database to validate metadata for all smart large objects in a database. If necessary, restore the data from a dbspace backup.

Monitor for data inconsistency

If the consistency-checking code detects an inconsistency during database server operation, an assertion failure is reported to the database server message log. (See the message-log topics in the *IBM Informix Administrator's Reference*.)

Read assertion failures in the message log and dump files

The following example shows the form that assertion failures take in the message log.

```
Assert Failed: Short description of what failed
   Who: Description of user/session/thread running at the time
   Result: State of the affected database server entity
   Action: What action the database server administrator should take
   See Also: file(s) containing additional diagnostics
```

The See Also: line contains one or more of the following file names:

- af.xxx
- shmem.xxx
- gcore.xxx
- gcore.xxx
- */path name*/core

In all cases, xxx is a hexadecimal number common to all files associated with the assertion failures of a single thread. The files af.xxx, shmem.xxx, and gcore.xxx are in the directory that the ONCONFIG parameter DUMPDIR specifies.

The file af.xxx contains a copy of the assertion-failure message that was sent to the message log, and the contents of the current, relevant structures and data buffers.

The file shmem.xxx contains a complete copy of the database server shared memory at the time of the assertion failure, but only if the ONCONFIG parameter DUMPSHMEM is set to 1 or to 2.

UNIX only: On UNIX, gcore.xxx contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPGCORE is set to 1 and your operating system supports the **gcore** utility. The core file contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPCORE is set to 1. The path name for the core file is the directory from which the database server was last invoked.

Validate table and tblspace data

To validate table and tblspace data, use the **oncheck -cD** command on the database or table.

Most of the general assertion-failure messages are followed by additional information that usually includes the tblspace where the error was detected. If this check verifies the inconsistency, unload the data from the table, drop the table, recreate the table, and reload the data. Otherwise, no other action is required.

In many cases, the database server stops immediately when an assertion fails. However, when failures seem to be specific to a table or smaller entity, the database server continues to run.

When an assertion fails because of inconsistencies on a data page that the database server accesses on behalf of a user, an error is also sent to the application process. The SQL error depends on the operation in progress. However, the ISAM error is almost always either -105 or -172, as follows:

```
-105 ISAM error: bad isam file format
-172 ISAM error: Unexpected internal error
```

For additional details about the objectives and contents of messages, see the topics about message-log messages in the *IBM Informix Administrator's Reference*.

Retain consistent level-0 backups

After you perform the checks described in "Verify consistency" on page 19-1 without errors, create a level-0 backup. Retain this storage-space backup and all subsequent logical-log backup tapes until you complete the next consistency check. Perform the consistency checks before every level-0 backup. If you do not, then at minimum keep all the tapes necessary to recover from the storage-space backup that was created immediately after the database server was verified to be consistent.

Deal with corruption

This section describes some of the symptoms of database server system corruption and actions that the database server or you, as administrator, can take to resolve the problems. Corruption in a database can occur as a consequence of hardware or operating-system problems, or from some unknown database server problems. Corruption can affect either data or database server overhead information.

Find symptoms of corruption

You can find information about corruption several different ways.

The database server alerts the user and administrator to possible corruption in the following ways:

- Error messages reported to the application state that pages, tables, or databases cannot be found. One of the following errors is always returned to the application if an operation has failed because of an inconsistency in the underlying data or overhead information:

  ```
  -105 ISAM error: bad isam file format
  -172 ISAM error: Unexpected internal error
  ```

- Assertion-failure reports are written to the database server message log. They always indicate files that contain additional diagnostic information that can help you determine the source of the problem. See "Verify consistency" on page 19-1.

- The **oncheck** utility returns errors
- The **ph_alert** table shows information about corrupted indexes.

Fix index corruption

At the first indication of corruption, run the **oncheck -cI** command to determine if corruption exists in the index.

If you check indexes while the database server is in online mode, **oncheck** detects the corruption but does not prompt you for repairs. If corruption exists, you can drop and recreate the indexes using SQL statements while you are in online mode (the database server locks the table and index). If you run **oncheck -cI** in quiescent mode and corruption is detected, you are prompted to confirm whether the utility attempts to repair the corruption.

Fix I/O errors on a chunk

If an I/O error occurs during the database server operation, the status of the chunk on which the error occurred changes to down.

If a chunk is down, the **onstat -d** display shows the chunk status as PD– for a primary chunk and MD– for a mirror chunk. For an example of **onstat -d** output, see the *IBM Informix Administrator's Reference*.

In addition, the message log lists a message with the location of the error and a suggested solution. The listed solution is a possible fix, but does not always correct the problem.

If the down chunk is mirrored, the database server continues to operate using the mirror chunk. Use operating-system utilities to determine what is wrong with the down chunk and correct the problem. You must then direct the database server to restore mirror chunk data.

For information about recovering a mirror chunk, see "Recover a mirror chunk" on page 18-6.

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate but cannot write to or read from the down chunk or any other chunks in the dbspace of that chunk. You must take steps to determine why the I/O error occurred, correct the problem, and restore the dbspace from a backup.

If you take the database server to offline mode when a chunk is marked as down (D), you can restart the database server, provided that the chunk marked as down does not contain critical data (logical-log files, the physical log, or the root dbspace).

Collect diagnostic information

Several ONCONFIG parameters affect the way in which the database server collects diagnostic information. Because an assertion failure is generally an indication of an unforeseen problem, notify IBM Informix Technical Support whenever one occurs. The diagnostic information collected is intended for the use of IBM Informix technical staff. The contents and use of af.*xxx* files and shared core are not further documented.

To determine the cause of the problem that triggered the assertion failure, it is critically important that you not delete diagnostic information until IBM Informix Technical Support indicates that you can do so. The af.xxx file often contains information that they require to resolve the problem.

Several ONCONFIG parameters direct the database server to preserve diagnostic information whenever an assertion failure is detected or whenever the database server enters into an end sequence:

- DUMPDIR
- DUMPSHMEM
- DUMPCNT
- DUMPCORE
- DUMPGCORE

For more information about the configuration parameters, see the *IBM Informix Administrator's Reference*.

You decide whether to set these parameters. Diagnostic output can use a large amount of disk space. (The exact content depends on the environment variables set and your operating system.) The elements of the output can include a copy of shared memory and a core dump.

Tip: A *core dump* is an image of a process in memory at the time that the assertion failed. On some systems, core dumps include a copy of shared memory. Core dumps are useful only if this is the case.

Database server administrators with disk-space constraints might prefer to write a script that detects the presence of diagnostic output in a specified directory and sends the output to tape. This approach preserves the diagnostic information and minimizes the amount of disk space used.

Disable I/O errors

IBM Informix divides disabling I/O errors into two general types: destructive and nondestructive. A disabling I/O error is destructive when the disk that contains a database becomes damaged in some way. This type of event threatens the integrity of data, and the database server marks the chunk and dbspace as down. The database server prohibits access to the damaged disk until you repair or replace the disk and perform a physical and logical restore.

A disabling I/O error is nondestructive when the error does not threaten the integrity of your data. Nondestructive errors occur when someone accidentally disconnects a cable, you somehow erase the symbolic link that you set up to point to a chunk, or a disk controller becomes damaged.

Before the database server considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when the database server attempts to perform an operation on a chunk that has at least one of the following characteristics:

- The chunk has no mirror.
- The primary or mirror companion of the chunk under question is offline.

Second, the error must occur when the database server attempts unsuccessfully to perform one of the following operations:

- Seek, read, or write on a chunk
- Open a chunk
- Verify that chunk information about the first used page is valid

  The database server performs this verification as a sanity check immediately after it opens a chunk.

You can prevent the database server from marking a dbspace as down while you investigate disabling I/O errors. If you find that the problem is trivial, such as a loose cable, you can bring the database server offline and then online again without restoring the affected dbspace from backup. If you find that the problem is more serious, such as a damaged disk, you can use **onmode -O** to mark the affected dbspace as down and continue processing.

Monitor the database server for disabling I/O errors

The database server notifies you about disabling I/O errors in two ways:

- Message log
- Event alarms

The message log to monitor disabling I/O errors

The database server sends a message to the message log when a disabling I/O error occurs.

The message is:

```
Assert Failed: Chunk {chunk-number} is being taken OFFLINE.
Who: Description of user/session/thread running at the time
Result: State of the affected database server entity
Action: What action the database server administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics
```

The result and action depend on the current setting of ONDBSPACEDOWN, as described in the following table.

ONDBSPACEDOWN setting	Result	Action
0	Dbspace {*space_name*} is disabled.	Restore dbspace {*space_name*}.
	Blobspace {*space_name*} is disabled.	Restore blobspace {*space_name*}.
1	The database server must stop.	Shut down and restart the database server.
2	The database server blocks at next checkpoint.	Use **onmode -k** to shut down, or use **onmode -O** to override.

The value of ONDBSPACEDOWN has no affect on temporary dbspaces. For temporary dbspaces, the database server continues processing regardless of the ONDBSPACEDOWN setting. If a temporary dbspace requires fixing, you can drop and recreate it.

For more information about interpreting messages that the database server sends to the message log, see the topics about message-log messages in the *IBM Informix Administrator's Reference*.

Event alarms to monitor disabling I/O errors

When a dbspace incurs a disabling I/O error, the database server passes the specified values as parameters to your event-alarm executable file.

Event alarm values:

Severity
> 4 (Emergency)

Class 5

Class message
> Dbspace is disabled: 'dbspace-name'

Specific message
> Chunk {chunk-number} is being taken OFFLINE.

Event ID
> 5001

If you want the database server to use event alarms to notify you about disabling I/O errors, write a script that the database server executes when it detects a disabling I/O error. For information about how to set up this executable file that you write, see the appendix on event alarms and the topics on configuration parameters in the *IBM Informix Administrator's Reference*.

No bad-sector mapping

IBM Informix relies on the operating system of your host computer for bad-sector mapping. The database server learns of a bad sector or a bad track when it receives a failure return code from a system call. When this situation occurs, the database server retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, the database server marks as down the chunk where the read or write was attempted.

The database server cannot take any action to identify the bad cylinder, track, or sector location because the only information available is the byte displacement within the chunk where the I/O operation was attempted.

If the database server detects an I/O error on a chunk that is not mirrored, it marks the chunk as down. If the down chunk contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

Part 5. High availability and scalability

A successful production environment requires database systems that are always available, with minimal if any planned outages, and that can be scaled quickly and easily as business requirements change.

Businesses must provide continuous access to database resources during planned and unplanned outages. Planned outages include scheduled maintenance of software or hardware. Unplanned outages are unexpected system failures such as power interruptions, network outages, hardware failures, operating system or other software errors. In the event of a disaster, such as an earthquake or a tsunami, there is the possibility of extensive system failure.

Businesses want to avoid overloading a server in the system to ensure data availability and to prevent, for example, denial of service attacks.

Businesses also want to quickly and easily expand their systems as their business grows, during seasonal business peak periods, and for end-of-month or end-of-year processing.

Systems with one or more of the following abilities can be resilient to outages and can improve the availability of data:

Redundancy
> The ability of a system to maintain secondary servers that are copies of the primary server and that can take over from the primary server if a failure occurs.

Failover
> The ability of a system to transfer all of the workload from a failed server to another server.

Workload balancing
> The ability of a system to automatically direct client requests to the server with the most workload capacity.

Scalability
> The ability of a system to take advantage of additional resources, such as processors, memory, or disk space.

Minimized affect on maintenance
> The ability to maintain all servers quickly and easily so that user applications are affected a little as possible.

Related concepts:

Related reference:

Chapter 20. Strategies for high availability and scalability

IBM Informix database software can be customized to create the appropriate high availability and scalability solution to match your business goals and environment.

To determine the best way to customize your database system for high availability and scalability, you must identify the strategies that help you achieve your business goals. You can use the appropriate Informix technologies and components to support those strategies.

Components supporting high availability and scalability

IBM Informix database software contains several components that can be customized to create systems that provide uninterrupted and continuous services to minimize downtime and maintenance.

- "Clusters"
- "Enterprise Replication" on page 20-3
- "Connection Manager" on page 20-3
- "Grid" on page 20-4

Clusters

A cluster consists of a single primary server and one or more secondary servers connected by a network. If the primary server in a cluster fails, a secondary server can take over the role of the primary server.

A primary server is the database server that owns the master copy of the data and that is connected to all of the secondary servers in the cluster. A secondary server is a database server that is connected to the primary server and that contains or has access to the same data as the primary server. Both the primary and the secondary servers in a cluster are configured with identical hardware and software.

Applications can start on any server. Applications are not required to be "aware" of any particular server. Traffic between the primary and secondary servers is secure. Also, traffic is secure between client applications and all servers.

A secondary server can be located near the primary server (in the same room, on the same floor, or in the same building) or it can be geographically remote from the primary server (such as in another building, in another town, or in another country). Some secondary servers have access to the same data as the primary server, while other secondary servers are synchronized so that they contain the same data as the primary server. The primary server sends whole transactions to the secondary servers after the transactions are committed on the primary server. The primary server does not send uncommitted transactions, so the database data is reliable. See Table 20-1 on page 20-2 for descriptions of secondary servers. Those secondary servers are synchronized with the primary server to build a continuously available, scalable data store.

Table 20-1. Types of secondary servers

Secondary server type	Description
Shared-disk (SD) secondary server	A server that shares disk space with a primary server. This type of secondary server does not maintain a copy of the physical database on its own disk space; rather, it shares disks with the primary server.
High-availability data replication (HDR) secondary server	A server that maintains a backup copy of the entire primary server through synchronous or asynchronous data replication. Applications can access this type of secondary server quickly if the primary server fails.
Remote stand-alone (RS) secondary server	A server that maintains a complete copy of the data, with updates transmitted asynchronously from the primary server over secure network connections. This type of secondary server can be geographically distant from the primary server and can be a remote back-up server in disaster-recovery scenarios.

Advantages of clustering are as follows:
- Clients at the site to which the data is replicated experience improved performance because those clients can access data locally rather than connecting to a remote database server over a network.
- Clients at all sites experience improved availability of replicated data. If the local copy of the replicated data is unavailable, clients can still access the remote copy of the data.

As the following figure illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.
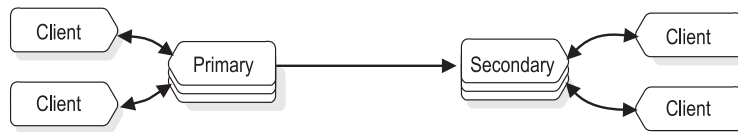
Figure 20-1. A primary and secondary database server in a cluster configuration

If one of the database servers fails, as the following figure shows, you can redirect the clients that use that database server to the other database server in the pair, which becomes the primary server.
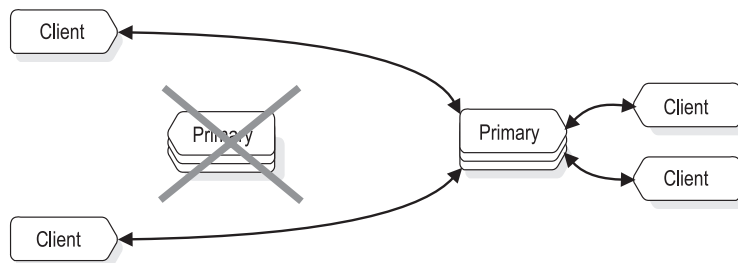
Figure 20-2. Database servers and clients in a data replication configuration after a failure

Enterprise Replication

An asynchronous, log-based tool for replicating data between IBM Informix database servers. Enterprise Replication implements asynchronous data replication, in which network and target database server outages are tolerated. In the event of a database server or network failure, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

Enterprise Replication ensures that all data reaches the appropriate server, regardless of a network or system failure. In the event of a failure, Enterprise Replication stores data until the network or system is operational. Enterprise Replication replicates data efficiently with a minimum of data copying and sending.

Connection Manager

The Connection Manager is a program that automatically manages and directs client connection requests. It redirects the requests based on redirection rules called service level agreements (SLAs) that are configured by the system administrator.

You can configure automatic failover using the Connection Manager failover configuration feature. If the Connection Manager detects that the primary server has failed, and no action is taken by the primary server to reconnect during the ensuing timeout period, the most appropriate secondary server is converted to the primary server. You configure Connection Manager failover parameters by using a configuration file.

The Connection Manager performs configurable load balancing, in which client redirection is based on server work load. The Connection Manager connects to each of the servers in the cluster and gathers statistics regarding the type of server, unused workload capacity, and the current state of the server. From this information, the Connection Manager redirects the client connection to the server with the least amount of activity.

You can configure multiple instances of the Connection Manager on different machines to avoid having the Connection Manager become a single point of failure. You can install and run the Connection Manager on an IBM Informix server instance or, to isolate the Connection Manager from database server failures, you can install it on a separate, non-Informix machine. In addition, you can configure the Connection Manager to run on a hardware platform other than the machines that make up the cluster.

To further increase availability, you can configure multiple Connection Manager instances. Configuring multiple Connection Manager instances is especially important to avoid having the Connection Manager become a single point of failure. Clients remain connected to servers after a failure of the Connection Manager; however, new client connections cannot connect to servers unless another Connection Manager instance is configured to serve as a backup. If the Connection Manager fails, all client connections are lost unless another Connection Manager instance is configured to serve as a backup.

In addition to performing client application redirection within a high-availability cluster, you can use the Connection Manager to route connection requests to one or more servers within a replicate set. You configure the Connection Manager to specify a name and an ordered list of servers that are all within the same Enterprise Replication domain, and then specify the service level agreement (SLA) name and redirection policy. The redirection policy specifies the type or the name of the target database server to which the client connects. Client applications use the SLA name to connect to the appropriate database server.

Grid

A grid is a named set of interconnected replication servers for propagating commands from an authorized server to the rest of the servers in the set. Much like the power grid distributes electrical power throughout a region, the grid that you define distributes SQL commands to the replication servers in the grid. A grid can be useful if you have multiple replication servers and you often require to perform the same tasks on every replication server. The following types of tasks are easily run through the grid:

- Administering servers; for example, adding chunks, removing logical logs, or changing configuration parameter settings
- Updating the database schema; for example, altering tables or adding tables
- Running or creating stored procedures or user-defined routines
- Updating data; for example, purging old data or updating values based on conditions
- Maintaining replication: Enabling replication when creating a table, and altering a replication definition when altering a replicated table.

Advantages of data replication

The advantages of data replication do not come without a cost. Data replication obviously requires more storage, and updating replicated data can take more processing time than updating a single object.

You can implement data replication in the logic of client applications by explicitly specifying where data must be updated. However, this method of achieving data replication is costly, prone to error, and difficult to maintain. Instead, the concept of data replication is often coupled with *replication transparency*. Replication transparency is built into a database server (instead of into client applications) to handle automatically the details of locating and maintaining data replicas.

Clustering versus mirroring

Clustering and mirroring are transparent methods for increasing fault tolerant.

Mirroring, described "Mirroring" on page 17-1, is the mechanism by which a single database server maintains a copy of a specific dbspace on a separate disk. This mechanism protects the data in mirrored dbspaces against disk failure because the database server automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

Alternatively, a cluster duplicates on an entirely separate database server all the data that a database server manages, not just the specified dbspaces. Because clustering involves two separate database servers, it protects the data that these database servers manage, not just against disk failures, but against all types of database server failures, including a computer failure or the catastrophic failure of

an entire site.
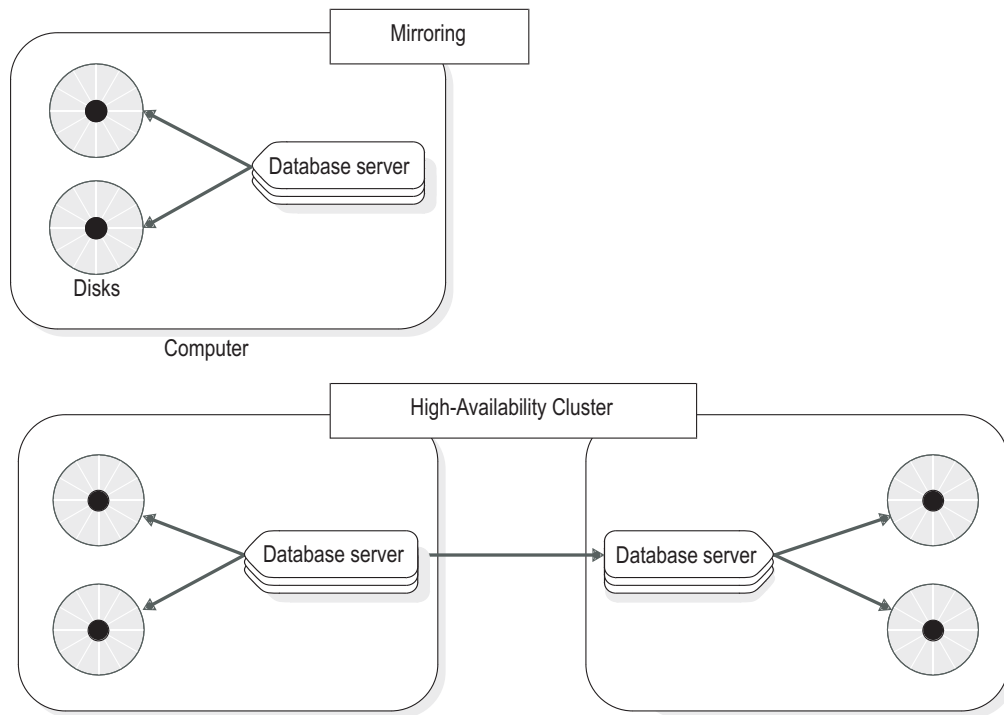
Figure 20-3. A comparison of mirroring and clustering

Clustering versus two-phase commit

The two-phase commit protocol, described in detail in Chapter 25, "Multiphase commit protocols," on page 25-1, ensures that transactions are uniformly committed or rolled back across multiple database servers.

In theory, you can take advantage of two-phase commit to replicate data by configuring two database servers with identical data and then defining triggers on one of the database servers that replicate updates to the other database server. However, this sort of implementation has numerous synchronization problems in different failure scenarios. Also, the performance of distributed transactions is inferior to clustering.

Clustering and ON-Monitor

You can combine a database cluster with Enterprise Replication to create a robust replication system. Clustering can ensure that an Enterprise Replication system remains fully connected by providing backup database servers for critical replication nodes.

When you combine a cluster and Enterprise Replication, only the primary server is connected to the Enterprise Replication system. No secondary servers participate in Enterprise Replication unless the primary server fails.

For more information, see *IBM Informix Enterprise Replication Guide* and "Enterprise Replication as part of the recoverable group" on page 22-12.

Type of data replicated in clusters

A high-availability cluster replicates data in dbspaces and sbspaces, but does not replicate data in blobspaces.

All built-in and extended data types are replicated to the secondary server. User-defined types (UDTs) must be logged and are located in a single database server. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server. For data stored in an sbspace to be replicated, the sbspace must be logged.

Data stored in operating system files or persistent external files or memory objects associated with user-defined routines are not replicated.

User-defined types, user-defined routines, and DataBlade modules have special installation and registration requirements. For instructions, see "How data initially replicates" on page 22-1.

Primary and secondary database servers

When you configure a set of database servers to use data replication, one database server is called the *primary* database server, and the others are called *secondary* database servers. (In this context, a database server that does not use data replication is called a *standard* database server.) The secondary server can include any combination of the SD secondary, RS secondary, and HDR secondary servers.

As the following figure illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.
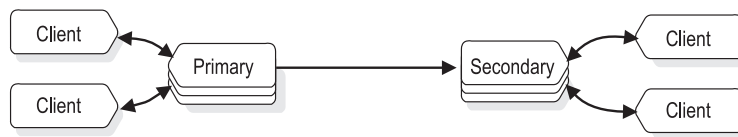
Figure 20-4. A primary and secondary database server in a data replication configuration

If one of the database servers fails, as the following figure shows, you can redirect the clients that use that database server to the other database server in the pair, which becomes the primary server.
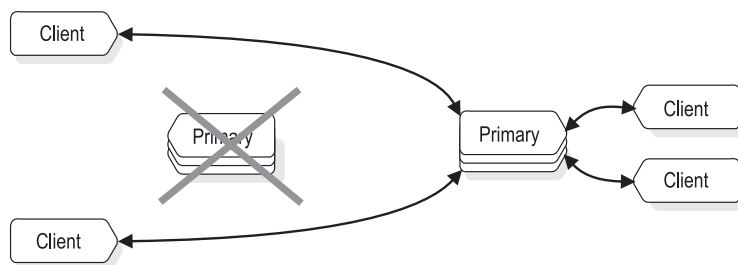
Figure 20-5. Database servers and clients in a data replication configuration after a failure

Transparent scaling and workload balancing strategies

IBM Informix servers scale easily and they dynamically balance workloads to ensure optimal use of resources.

IBM Informix can address these objectives:
- "Periodically increase capacity" on page 20-7
- "Geographically disperse processing with increased reporting capacity" on page 20-8

- "Balance workload to optimize resource use" on page 20-9

Periodically increase capacity

If your business environment experiences peak periods, you might be required to periodically increase capacity. You can increase capacity by adding a remote stand-alone secondary server. That type of secondary server maintains a complete copy of the data, with updates transmitted asynchronously from the primary server over secure network connections. If the amount of data is very large and making multiple copies of it is difficult, use shared-disk secondary servers instead of remote stand-alone secondary servers. You can use high-availability data replication (HDR) secondary servers if you want to increase capacity only for reporting (read-only) workloads.

Table 20-2. Scalability with shared-disk secondary servers

Advantages	Potential disadvantages
• Very high availability. This secondary server shares disks with the primary server.	• No failover. The secondary server might be configured to run on the same computer hardware as the primary server.
	• No data redundancy. This secondary server does not maintain a copy of the data. (Use SAN devices for disk storage.)
	• The primary and secondary servers require the same hardware, operating system, and version of the database server product.

Use an SD secondary server for these reasons:
- Increased reporting capacity

  Multiple secondary servers can offload reporting function without affecting the primary server.
- Server failure backup

  In the event of a failure of the primary server, an SD secondary can be promoted quickly and easily to a primary server. For example, if you are using SAN (storage area network) devices that provide ample and reliable disk storage but you are concerned with server failure, SD secondary servers can provide a reliable backup.

Table 20-3. Scalability with remote stand-alone secondary servers

Advantages	Potential disadvantages
• Data redundancy. This secondary server maintains a copy of the data.	• The primary and secondary servers require the same hardware, operating system, and version of the database server product.
• Failover. The secondary server can be geographically remote from the primary server, such as in another building, another town, or another country.	
• No requirement to change applications. Client connections to primary or secondary server can be automatically switched in the event of server failure.	

Geographically disperse processing with increased reporting capacity

Businesses with offices in various locations might want to use local servers for processing local requests instead of relying on a single, centralized server. In that case, you can set up a network of Enterprise Replication servers. Target database server outages are tolerated. If database server or network failure, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

Table 20-4. Scalability with Enterprise Replication

Advantages	Potential disadvantages
• The servers can be in another building, another town, or another country. • The servers can be on different hardware. • The servers can run on different operating systems. • The servers can run different versions of the database server product. • A subset of the data can be replicated (asynchronous, log-based replication). • It is possible to add shared-disk secondary servers to assist the replication servers, using multiple Connection Managers for automatic client redirection.	• Conflicts are possible. • Transaction failures are possible. In that case, you must repair inconsistent data.

Use an RS secondary server in your environment for the following reasons:

• Increased server availability

One or more RS secondary servers provide added assurance by maintaining multiple servers that can be used to increase availability.

• Geographically distant backup support

It is often desirable to have a secondary server located at some distance from the site for worst-case disaster recovery scenarios. An RS secondary server is an ideal remote backup solution. The high level of coordination between a primary and secondary HDR pair can cause performance issues if the secondary server is located on a WAN (Wide-Area Network). Keeping the primary and secondary servers relatively close together eases maintenance and minimizes the affect on performance.

• Improved reporting performance

Multiple secondary servers can offload reporting function without affecting the primary server. Also, an RS secondary server configuration makes it easier to isolate reporting requirements from the HA requirements, resulting in better solutions for both environments.

• Availability over unstable networks

A slow or unstable network environment can cause delays on both the primary and secondary server if checkpoints are achieved synchronously. RS secondary server configurations use fully duplexed networking and require no such

coordination. An RS secondary server is an attractive solution if network performance between the primary server and RS secondary server is less than optimal.

Balance workload to optimize resource use

You can configure workload balancing when you create or modify a service level agreement SLA. Informix gathers information from each server in a cluster and automatically connects the client application to the server that has the least amount of activity.

You can create groups within a cluster that are specific to certain types of applications, such as those for online transaction processing (OLTP) or (warehouse). Applications can choose to connect to the specific group for optimized performance of each type of query.

High availability strategies

IBM Informix can be configured to maximize availability in various business situations.

Goal	Strategy	Advantages	Potential disadvantages
Protect system from server failure	Use a secondary server that shares disk space with the primary server. (shared-disk secondary server)	• Very high availability. This secondary server has access to the same data as the primary server. If the primary server fails, the secondary server can take over quickly. • The database is always in sync because this secondary server has access to the same data as the primary server. • No requirement to change applications. Client connections to primary or secondary server are automatically switched in the event of server failure.	• This secondary server on the same computer as the primary server. • No data redundancy. This secondary server does not maintain a copy of the data. (Use SAN devices for disk storage.) • Primary and secondary servers require the same hardware, operating system, and version of the database server product. • Secondary server hardware must be able to handle the same load as the primary server. If the secondary server is too small, it might affect the performance of the primary.

Protection from site failure	• Use a secondary server that maintains a copy of the database server and the data. (high availability data replication server) • (Can also use RSS and ER)	• Very high availability. Applications can access this server quickly if they cannot connect to a primary server. • Data is replicated synchronously. • Increased scalability • No requirement to change applications	• Local to the primary • Requires an exact replica of the data (including table and database schemas). • Primary and secondary servers require the same hardware, operating system, and version of the database server product.
Multilevel site failure protection	• Use a secondary server that is geographically distant from the primary server and that is updated asynchronously from the primary server. (remote stand-alone secondary server) • (Can also use ER)	• Very high availability. Applications can access this server quickly if they cannot connect to a primary server. • Data is replicated asynchronously. • Increased scalability • No requirement to change applications	
Geographically dispersed processing with site failure protection	ER and HDR	ER and backup for ER	Multiple connection managers required

Chapter 21. High-availability cluster configuration

These topics describe how to plan, configure, start, and monitor high-availability clusters for IBM Informix, and how to restore data after a media failure. If you plan to use a high-availability cluster, read all the topics within this section first. If you plan to use IBM Informix Enterprise Replication, see the *IBM Informix Enterprise Replication Guide*.

Part 5, "High availability and scalability," explains what a high-availability cluster is, how it works, and how to design client applications for a cluster environment.

Related reference:

"Data replication environments" on page 1-21

Plan for a high-availability cluster

Before you start setting up computers and database servers to use a high-availability cluster, you might want to do some initial planning. The following list contains planning tasks to perform:

- Choose and acquire appropriate hardware.
- If you are using more than one database server to store data that you want to replicate, migrate and redistribute data, so that it can be managed by a single database server.
- Ensure that all databases you want to replicate use transaction logging. To turn on transaction logging, see Chapter 12, "Manage the database-logging mode," on page 12-1.
- Develop client applications to make use of both database servers in the replication pair. For an explanation of design considerations, see "Redirection and connectivity for data-replication clients" on page 24-6 and "Design data replication group clients" on page 22-21.
- Create a schedule for starting HDR for the first time.
- Design a storage-space and logical-log backup schedule for the primary database server.
- Produce a plan for how to handle failures of either database server and how to restart HDR after a failure. Read "Redirection and connectivity for data-replication clients" on page 24-6.

Configuring clusters

To configure your system as a high-availability cluster, you must take the following actions:

- Meet hardware and operating-system requirements.
- Meet database and data requirements.
- Meet database server configuration requirements.
- Configure connectivity.

Each of these topics are explained in this section.

You can configure your system to use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures the privacy and integrity of data transmitted over the network, for HDR communications. You can use the SSL protocol for

connections between primary and secondary servers and for connections with remote standalone (RS) and shared disk (SD) secondary servers in a high-availability configuration. For information about using the SSL protocol, see the "Secure Sockets Layer Communication Protocol Encryption" section of the *IBM Informix Security Guide*.

The Connection Manager also supports Distributed Relational Database Architecture (DRDA) connections. For more information, see "Distributed Relational Database Architecture (DRDA) communications" on page 2-45.

Hardware and operating-system requirements for clusters

For a high-availability cluster to function, your hardware must meet certain requirements.

Your hardware must meet the following requirements:
- The primary and secondary servers must be able to run the same IBM Informix executable image, even if they do not have identical hardware or operating systems. For example, you can use servers with different Linux 32-bit operating systems because those operating systems can run the same Informix executable image. In this situation, you cannot add a server on a Linux 64-bit operating system because that operating system requires a different Informix executable image. Check the machine notes file: you can use any combination of hardware and operating systems listed as supported in the same machine notes file.
- The hardware that runs the primary and secondary database servers must support network capabilities.
- The amount of disk space allocated to dbspaces for the primary and secondary database servers must be equal. The type of disk space is irrelevant; you can use any mixture of raw or cooked spaces on the two database servers.
- The chunks on each computer must have the same path names. Symbolic links are allowed for UNIX platforms, but not for Windows platforms.

Database and data requirements for clusters

For a high-availability cluster to function, your database and data must meet certain requirements.

Your database and data must meet the following requirements:
- All data must be logged.

  All databases that you want to replicate must have transaction logging turned on.

  This requirement is important because the secondary database server uses logical-log records from the primary database server to update the data that it manages. If databases managed by the primary database server do not use logging, updates to those databases do not generate log records, so the secondary database server has no means of updating the replicated data. Logging can be buffered or unbuffered.

  If you must turn on transaction logging before you start HDR, see "Turn on transaction logging with ontape" on page 12-3.
- The data must be located in dbspaces or sbspaces.

  If your primary database server has simple large objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal HDR processing. However, simple-large-object data within dbspaces is replicated.

Smart large objects, which are stored in sbspaces, are replicated. The sbspaces must be logged. User-defined types (UDTs) are replicated, unless they have out-of-row data stored in operating system files. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server.

- The secondary servers must not use disk compression.

  If you use the IBM Informix disk compression feature, data that is compressed in the source table is compressed in the target table. You cannot perform compression operations on an HDR secondary, RS secondary, or SD secondary server, because the HDR target server must have the same data and physical layout as the source server.

Database server configuration requirements for clusters

For a high-availability cluster server pair to function, you must fully configure each of the database servers. For information about configuring a database server, see Chapter 1, "Database server installation and configuration," on page 1-1. You can then use the relevant aspects of that configuration to configure the other database server in the pair. For more information about the configuration parameters, see the *IBM Informix Administrator's Reference*.

These topics describe the following configuration considerations for cluster database server pairs:

Database server version
The versions of the database server on the primary and secondary database servers must be identical.

Storage space and chunk configuration
The number of dbspaces, the number of chunks, their sizes, their path names, and their offsets must be identical on the primary and secondary database servers. In addition, the configuration must contain at least one temporary dbspace if the HDR secondary server is used for creating activity reports. See "Use of temporary dbspaces for sorting and temporary tables" on page 22-21.

UNIX Only:

> You must use symbolic links for the chunk path names, as explained in "Allocating raw disk space on UNIX" on page 9-3.
>
> **Important:** If you do not use symbolic links for chunk path names, you cannot easily change the path name of a chunk. For more information, see "Renaming chunks" on page 22-23.

The following ONCONFIG parameters must have the same value on each database server:
- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE

Non-default page sizes in an HDR environment
The page size of a dbspace and the buffer pool specifications are automatically propagated from the primary to the secondary database server. While both the primary and the secondary database servers must have the same buffer pools, the number of buffers in the buffer pools are not required to match.

Mirroring

You are not required to set the MIRROR parameter to the same value on the two database servers; you can enable mirroring on one database server and disable mirroring on the other. However, if you specify a mirror chunk for the root chunk of the primary database server, you must also specify a mirror chunk for the root chunk on the secondary database server. Therefore, the following ONCONFIG parameters must be set to the same value on both database servers:

- MIRROROFFSET
- MIRRORPATH

Physical-log configuration

The physical log must be identical on both database servers. The following ONCONFIG parameters must have the same value on each database server:

- PHYSBUFF
- PHYSFILE

Dbspace and logical-log tape backup devices

You can specify different tape devices for the primary and secondary database servers.

If you use ON-Bar, set the ON-Bar configuration parameters to the same value on both database servers. For information about the ON-Bar parameters, see the *IBM Informix Backup and Restore Guide*.

If you use **ontape**, the tape size and tape block size for the storage-space and logical-log backup devices must be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK
- LTAPESIZE
- TAPEBLK
- TAPESIZE

To use a tape to its full physical capacity, set LTAPESIZE and TAPESIZE to 0.

Logical-log configuration

All log records are replicated to the secondary server. You must configure the same number of logical-log files and the same logical-log size for both database servers. The following ONCONFIG parameters must have the same value on each database server:

- LOGBUFF
- LOGFILES
- LOGSIZE
- DYNAMIC_LOGS

The database server logs the addition of logical-log files. Logical-log files added dynamically on the primary server are automatically replicated on the secondary server. Although the DYNAMIC_LOGS value on the secondary server has no effect, keep DYNAMIC_LOGS in sync with the value on the primary server, in case their roles switch.

HDR configuration parameters

The following HDR configuration parameters must be set to the same value on both database servers in the replication pair:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT

For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1.

Cluster transaction coordination

You can set the CLUSTER_TXN_SCOPE configuration parameter or run the SET ENVIRONMENT CLUSTER_TXN_SCOPE statement to require that operations run by a client application do not receive commits until the operations are fully propagated across a cluster, or are applied on a secondary server.

If a client application performs a multistep operation, such as loading or updating data, and then attempts to process the data in another server session or on another node of a high-availability cluster, the operation can fail because of asynchronous log processing. The load or update must be applied on a cluster database node before it can be processed on the node. Cluster transaction coordination forces operations to wait for an apply, cluster-wide or on a secondary server, before a commit is returned. Cluster transaction coordination ensures that steps in a multistep process occur in serial order.

Remote stand-alone secondary (RSS) servers that have a DELAY_APPLY or STOP_APPLY configuration parameter value other than 0 cannot take part in cluster transaction coordination.

Related reference:

➡ CLUSTER_TXN_SCOPE configuration parameter (Administrator's Reference)

➡ DELAY_APPLY Configuration Parameter (Administrator's Reference)

➡ STOP_APPLY configuration parameter (Administrator's Reference)

➡ SET ENVIRONMENT statement (Guide to SQL: Syntax)

Configuring secure connections for clusters

For a high availability cluster to function, the database servers must establish trusted connection with each other. You can secure connections between cluster servers by using the connection security option in the sqlhosts file and by using hosts.equiv files or files specified by the REMOTE_SERVER_CFG configuration parameters.

The secure ports listed in the sqlhosts files can only be used for cluster communication. Client applications cannot connect to secure ports.

To configure a trusted environment for replication:
1. Edit the sqlhosts file on each server in the cluster:
   a. Add an entry for the database server that is running on that computer, and include the **s=6** option.
   b. Add entries for all the other servers in the cluster. Do not include the **s=6** option in these entries.
2. Set the nettype field of the sqlhosts file or registry and the NETTYPE configuration parameter to a network protocol such as ontlitcp or onsoctcp so

that the database servers on two different computers can communicate with each other. You cannot use a non-network protocol such as **onipcshm**, **onipcstr**, or **onipcnmp**.

3. Set trusted-host information in one of the following ways

   - Create a `hosts.equiv` file in the `$INFORMIXDIR/etc` directory and include the host names of the other cluster servers, each on a separate line. The file must be owned by user **informix**, belong to group **informix**, and the permissions must be restricted so that at most user **informix** can modify the file (using octal permissions, one of the values 644, 640, 444, or 440 is appropriate).
   - Set the S6_USE_REMOTE_SERVER_CFG configuration parameter to 1, then set the REMOTE_SERVER_CFG configuration parameter to a file that contains the host names of the other cluster servers.

4. Create a server alias for running utilities and client applications. For example, set the **INFORMIXSERVER** environment variable to the alias to run utilities such as **onstat** and **ontape** and client applications such as DB-Access.

Related concepts:

"Trusted-host information" on page 2-13

Related tasks:

"Changing client connectivity information" on page 24-9

Related reference:

"sqlhosts Connectivity information" on page 2-21

➡ S6_USE_REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

➡ REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

"sqlhosts file and SQLHOSTS registry key options" on page 2-25

Starting HDR for the First Time

After you complete the HDR configuration, you are ready to start HDR. This topic describes the necessary steps for starting HDR.

You want to start HDR on two database servers, **ServerA** and **ServerB**. The procedure for starting HDR, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. The following table lists the commands required to perform each step and the messages sent to the message log. You can use **ontape** or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure.

Important: Even if you use ON-Bar to perform the backup and restore, the **ontape** utility is still required on both database servers to perform back ups and to apply logical logs. Do not remove the **ontape** utility from database servers that participate in an HDR cluster environment.
You can also set up HDR using external backup and restore. See the *IBM Informix Backup and Restore Guide* for information about how to perform an external backup and restore. See "Decrease setup time using the ontape STDIO feature" on page 21-9 for the quickest way to set up your HDR secondary directly from the HDR primary.

To start HDR:

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.

2. Create a level-0 backup of **ServerA**.
3. Use the **onmode -d** command to set the type of **ServerA** to primary and to indicate the name of the associated secondary database server (in this case **ServerB**).

   When you issue an **onmode -d** command, the database server attempts to establish a connection with the other database server in the HDR pair and to start HDR operation. The attempt to establish a connection succeeds only if the other database server in the pair is already set to the correct type.

   At this point, **ServerB** is not online and is not set to type secondary, so the HDR connection is not established.
4. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 1 on page 21-6. Do not perform a logical restore.

   If you are using:
   - ON-Bar, use the **onbar -r -p** command to perform a physical restore.
   - ON-Bar and performing an external restore, use the **onbar -r -p -e** command to perform the physical restore.
   - **ontape**, use the **ontape -p** option. You cannot use the **ontape -r** option because it performs both a physical and a logical restore.

     Note: You must place the physical restore of your primary server on the secondary server if they are on two different machines. The location of the physical restore is defined by the onconfig parameter **TAPE**. You must set your *IFX_ONTAPE_FILE_PREFIX* variable on your secondary server before you can run **ontape -p**.
   - **ontape** and performing an external restore, use the **ontape -p -e** command to perform the physical restore.
5. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server.

   ServerB tries to establish an HDR connection with the primary database server (**ServerA**) and start operation. The connection must be successfully established.

   Before HDR begins, the secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 2. If all these logical-log records still are located on the primary database server disk, the primary database server sends these records directly to the secondary database server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 6.

   Important: You must complete steps 4 to 5 during the same session. If you must shut down and restart the secondary database server after step 4, you must redo step 4.
6. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.

   If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

   After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

The following table illustrates the preceding steps so that you can clearly determine which steps are performed on the primary server and which are performed on the secondary server. The table also shows information written to the log file after each step is performed.

Table 21-1. Steps to start HDR for the first time

Step	On the primary (ServerA)	On the secondary (ServerB)
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.

2.	**ontape** command:
	
	Run **ontape -s -L 0**
	
	ON-Bar command:Run **onbar -b -L 0**
	
	Messages to message log
	
	`Level 0 archive started on rootdbs`
	`Archive on rootdbs completed`	

3.	**onmode** command
	
	onmode -d primary *sec_name*
	
	Messages to message log
	
	`DR: new type = primary`
	`    server name = `*sec_name*
	`DR: Trying to connect to secondary server`
	`DR: Cannot connect to secondary server`	

4.		**ontape** command
		
		Run **ontape -p** or **ontape -p -e**
		
		Answer no when you are prompted to back up the logs.
		
		ON-Bar command
		
		Run **onbar -r -p** or **onbar -r -p -e**
		
		Messages to message log
		
		`IBM Informix Database Server Initialized --`
		`    Shared Memory Initialized`
		`Recovery Mode`
		`Physical restore of rootdbs started`
		`Physical restore of rootdbs completed`

5.		Run **onmode -d secondary** *prim_name* **Messages to message log** `DR: new type = secondary,`` primary server` ` name = `*prim_name* If all the logical-log records written to the primary database server since step 1 still are located on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 6 must be performed).

Table 21-1. Steps to start HDR for the first time (continued)

Step	On the primary (ServerA)	On the secondary (ServerB)

	Messages to message log
	
	`DR: Primary server connected`
	`DR: Primary server operational`
		Messages to message log
		
		`DR: Trying to connect to primary server`
		`DR: Secondary server connected`
		`DR: Failure recovery from disk in progress`
		`n recovery worker threads will be started`
		`Logical Recovery Started`
		`Start Logical Recovery - Start Log n, End Log ?`
		`Starting Log Position - n 0xnnnnn`
		`DR: Secondary server operational`

6.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l**

	Messages to message log
	
	`DR: Primary server connected`
	`DR: Primary server operational`
		Messages to message log
		
		`DR: Secondary server connected`
		`DR: Failure recovery from disk in progress`
		`n recovery worker threads will be started`
		`Logical Recovery Started`
		`Start Logical Recovery - Start Log n, End Log ?`
		`Starting Log Position - n 0xnnnnn`
		`DR: Secondary server operational`

Related concepts:

"Backup and restore with high-availability clusters" on page 22-37

Related reference:

"Recovering a cluster after critical data is damaged" on page 24-14

Decrease setup time using the ontape STDIO feature

You can dramatically improve the speed of setting up HDR by using the **ontape** STDIO feature. Using this feature, **ontape** writes the data to the shell's standard output during a backup, and then read it from standard input during a restore. Combining a STDIO backup with a simultaneous STDIO restore in a pipe using a remote command interpreter (such as rsh or ssh), allows performing the initial setup of an HDR (or RSS) secondary server using a single command line. This saves storage space by not writing to or reading from tape or disk, and does not require waiting for the backup to finish before the restore can start.

See the *IBM Informix Backup and Restore Guide* for details about using the STDIO value.

This method for setting up HDR using **ontape** can be used regardless of which backup utility is used (**ontape** or ON-Bar).

Important: When you use STDIO in this way, no persistent backup is saved anywhere that can be used to perform a restore. The use of the **-F** (fake) option on the source (backup) side does not record the backup in the database server's reserved pages. Also, any interactive dialog is suppressed and no prompts or questions are displayed. You must also ensure that the remote part of the pipe picks the appropriate environment for the remote Informix instance. The script must not produce any output other than the backup data since this would be read by the restore process (for example, do not enable tracing).

The steps in the following table must be performed by user **informix**, the scripts must be executable, and, if called without a complete path, must be located in your home directory. You can use ssh instead of rsh if you require secure data transmission across the network.

Table 21-2. Alternative method of setting up HDR from the primary server, using rsh

Step	On the primary	On the secondary
1.		Install UDRs, UDTs, and DataBlade modules.
2.	Install UDRs, UDTs, and DataBlade modules.	
3.	Register UDRs, UDTs, and modules.	
4.	Run **onmode -d primary** *sec_name*	
5	Run **ontape -s -L 0 -t STDIO -F	rsh** *sec_name* **ontape_HDR_restore.ksh**	
6.		Run **onmode -d secondary** *pri_name*

In the previous table, the script `ontape_HDR_restore.ksh` on the secondary server must contain the following commands:

```
#!/bin/ksh
# first get the proper Informix environment set
. hdr_sec.env
# redirecting stdout and stderr required since otherwise command might never return
ontape -p -t STDIO > /dev/null 2>&1
```

The following steps show how to set up HDR from the secondary server.

Table 21-3. alternative method of setting up HDR from the secondary server, using rsh:

Step	On the Primary	On the Secondary
1.		Install UDRs, UDTs, and DataBlade modules.
2.	Install UDRs, UDTs, and DataBlade modules.	
3.	Register UDRs, UDTs, and DataBlade modules.	
4.	Run **onmode -d primary** *sec_name*	
5.		run **rsh** *pri_name* **ontape_HDR_backup.ksh	ontape -p -t STDIO**
6.		Run **onmode -d secondary** *pri_name*

In the previous table, the script `ontape_HDR_backup.ksh` on the primary server must contain the following commands:

```
#!/bin/ksh
# first get the proper Informix environment set
. hdr_pri.env
ontape -s -L 0 -F -t STDIO
```

Remote standalone secondary servers

These topics provide an overview of setting up and configuring remote standalone (RS) secondary servers in a high availability environment.

Related reference:

"Data replication environments" on page 1-21

Comparison of RS secondary servers and HDR secondary servers

An RS secondary server is similar in many ways to an HDR secondary server. Logs are sent to the RS secondary server in much the same way as a primary server sends logs to an HDR secondary server. However, the RS secondary server is designed to function entirely within an asynchronous communication framework so that its affect on the primary server is minimized. Neither transaction commits nor checkpoints are synchronized between the primary server and RS secondary servers. Any transaction committed on the primary server is not guaranteed to be committed at the same time on the RS secondary server.

In a high-availability cluster, the log of the HDR secondary server must be ahead of the logs of any RS secondary servers. If the HDR secondary server becomes offline, the primary server continues to send logs to the RS secondary servers. However, when the HDR secondary comes back online, IBM Informix stops sending logs to RS secondary servers and prioritizes sending logs to the HDR secondary server until its log replay is ahead of the RS secondary server. This prioritization of the HDR secondary server logs is required because the HDR secondary server is the first failover choice in the cluster. If the RS secondary server logs are ahead of the HDR secondary server logs when a failover occurs, then the RS secondary server cannot synchronize with the new primary server.

While an RS secondary server is similar to an HDR secondary server, there are several things that an HDR secondary server supports that an RS secondary server does not support:

- SYNC mode
- DRAUTO parameter
- Synchronized checkpoints

For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1.

Index page logging

You must enable index page logging to use an RS secondary server.

Related concepts:

"Checkpoints between database servers" on page 22-6

How index page logging works

When an index is created, index page logging writes the pages to the logical log for the purpose of synchronizing index creation between servers in high-availability environments.

Index page logging writes the full index to the log file, which is then transmitted asynchronously to the secondary server. The secondary server can be either an RS secondary or an HDR secondary server. The log file transactions are then read into the database on the secondary server. The secondary server is not required to rebuild the index during recovery. For RS secondary servers, the primary server does not wait for an acknowledgment from the secondary server, which allows immediate access to the index on the primary server.

Control index page logging with the ONCONFIG parameter LOG_INDEX_BUILDS. Set the LOG_INDEX_BUILDS parameter to 1 (enabled), to build indexes on the primary server and send them to the secondary server.

Enable or disable index page logging

Use the LOG_INDEX_BUILDS configuration parameter to enable or disable index page logging when the database server starts. You can change the value of LOG_INDEX_BUILDS in the onconfig file by running **onmode -wf LOG_INDEX_BUILDS=1** (enable) or **0** (disable).

Index page logging must be enabled when an RS secondary server exists in a high-availability environment.

View index page logging statistics

You can use the **onstat** utility or system-monitoring interface (SMI) tables to view whether index page logging is enabled or disabled. The statistics also display the date and time index page logging was enabled or disabled.

To view index page logging statistics, use the **onstat -g ipl** command, or query the **sysipl** table.

For an example of **onstat -g ipl** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Server Multiplexer Group (SMX) connections

Server Multiplexer Group (SMX) is a communications interface that supports encrypted multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.

Enable SMX encryption

Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high availability configurations. If you set the ENCRYPT_SMX parameter to 1, encryption is used for SMX transactions only when the database server being connected to also supports encryption. If you set the ENCRYPT_SMX configuration parameter to 2 , only connections to encrypted database servers are allowed. Setting ENCRYPT_SMX to 0 disables encryption between servers.

Obtain SMX statistics

You can use the **onstat** utility or system-monitoring interface (SMI) tables to view SMX connection statistics or SMX session statistics.

To view SMX connection statistics, use the **onstat -g smx** command.

To view SMX session statistics, use the **onstat -g smx ses** command.

For examples of **onstat -g smx** and **onstat -g smx ses** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Starting an RS secondary server for the first time

After you complete the hardware configuration of the RS secondary server, you are ready to start the RS secondary server and connect it to the primary server.

Suppose you want to start a primary server and an RS secondary server, **ServerA** and **ServerB**. The procedure for starting the servers, using **ServerA** as the primary

database server and **ServerB** as the RS secondary database server, is described in the following steps. The table ahead lists the commands required to perform each step.

The procedure requires that the primary server be backed up and then restored onto the secondary server. You can use **ontape** or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure.

Important: Even if you use ON-Bar to perform the backup and restore, the **ontape** utility is still required on both database servers to perform back ups and to apply logical logs. Do not remove the **ontape** utility from database servers that participate in an HDR cluster environment.
You can also set up an RS secondary server using standard ON-Bar or **ontape** commands for external backup and restore.

To start a primary server with an RS secondary server:
1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.

   For information about how to install user-defined types or user-defined routines see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*. For information about how to install DataBlade modules, see the *IBM Informix DataBlade Module Installation and Registration Guide*.
2. Activate index page logging on the primary server.
3. Record the identity of the RS secondary server on the primary server. The optional password provides authentication between the primary and RS secondary server when the connection between the primary and secondary servers is established for the first time.
4. Create a level-0 backup of **ServerA**.
5. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 4. Do not perform a logical restore.

   Use the appropriate command:
   - Use the **onbar -r -p** command to perform a physical restore.
   - Use the **onbar -r -p -e** command to perform a physical external restore.
   - Use the **ontape -p** option. (Do not use the **ontape -r** option because it performs both a physical and a logical restore.)
   - Use the **ontape -p -e** command to perform the physical external restore.
6.
7. Use the **onmode -d RSS ServerA** *password* command to set the type of **ServerB** to an RS secondary server and indicate the associated primary database server.

   ServerB tries to establish a connection with the primary database server (**ServerA**) and start operation. The connection must be successfully established.

   The secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 4. If all these logical-log records are still located on the primary database server disk, the primary database server sends these records directly to the RS secondary server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 8 on page 21-14.

Important: You must complete steps 5 on page 21-13 through 6 on page 21-13 during the same session. If you must shut down and restart the secondary database server after step 5 on page 21-13, you must redo step 5 on page 21-13.

8. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.

If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

Table 21-4. Steps to start a primary with an RS secondary server for the first time

Step	On the primary	On the RS secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**onmode** command **onmode -wf LOG_INDEX_BUILDS=1**	
3.	**onmode** command **onmode -d add RSS** *rss_servername password*	
4.	**ontape** command **ontape -s -L 0** ON-Bar command **onbar -b -L 0**	
5.		**ontape** command **ontape -p** or **ontape -p -e** Answer no when you are prompted to back up the logs. ON-Bar command **onbar -r -p** or **onbar -r -p -e**
6.		**onmode** command **onmode -d RSS** *primary_servername password* If all the logical-log records written to the primary database server since step 1 still are located on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 8 must be performed).
7.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l** This step is required only when the secondary database server prompts you to recover the logical-log files from the tape backup.

Decrease setup time through an alternative backup method

You can dramatically improve the speed of setting up a secondary server by using the **ontape** STDIO feature. See "Decrease setup time using the ontape STDIO feature" on page 21-9 for more information.

See the *IBM Informix Backup and Restore Guide* for details about using the STDIO value.

Converting an offline primary server to an RS secondary server

After a planned or unplanned failover of the primary server to an RS secondary server, you can convert the old primary server to an RS secondary server.

For example, assume you have a primary server named **srv1** that has failed over to an RS secondary server named **srv2**. The following steps show how to convert the old primary server to an RS secondary server.

1. On the new primary server (**srv2**) register the old primary server (**srv1** as the RS secondary server.

   ```
   onmode -d add RSS srv1
   ```

2. If you are converting the old primary server to an RS secondary server and the server is offline, then initialize the server using backup and restore commands shown here: Starting an RS secondary server for the first time. Alternatively you can initialize the old primary server by running the following command:

   ```
   oninit -PHY
   ```

   See The oninit utility for more information.

3. Convert the server to an RS secondary server using the following commands:

   ```
   onmode -d RSS srv2
   ```

Delayed application of log records

To aid in disaster recovery scenarios, you can configure RS secondary servers to wait for a specified period of time before applying logs received from the primary server.

By delaying the application of log files you can recover quickly from erroneous database modifications by restoring the database from the RS secondary server. You can also stop the application of logs on an RS secondary server at a specified time.

For example, suppose a database administrator wants to delete certain rows from a table based on the age of the row. Each row in the table contains a timestamp that indicates when the row was created. If the database administrator inadvertently sets the filter to the wrong date, more rows than intended might be deleted. By delaying the application of log files, the rows would still exist on the RS secondary server. The database administrator can then extract the rows from the secondary server and insert them on the primary server.

Now suppose a database administrator is required to perform changes to the schema by renaming a table, but types the wrong command and drops the table **orders** instead of changing the table name to **store_orders**. If an RS secondary

server is configured to delay application of logs, the database administrator can recover the **orders** table from the secondary server.

When delayed application of log files configured, transactions sent from the primary server are not applied until after a specified period of time has elapsed. Log files received from the primary server are staged in a specified secure directory on the RS secondary server, and then applied after the specified period of time. There are two ways to delay the application of log files:

- Apply the staged log files after a specified time interval
- Stop applying log files at a specified time

You enable the delayed application of log files by setting configuration parameters in the `onconfig` file of the RS secondary server. You must specify the directory in which log files are staged by setting the LOG_STAGING_DIR configuration parameter before enabling the delayed application of log files. After specifying the LOG_STAGING_DIR configuration parameter, you configure the DELAY_APPLY or STOP_APPLY configuration parameters either by editing the `onconfig` file or dynamically using **onmode -wf** commands.

Where log records are stored

The server creates additional directories named `ifmxlog_##` in the directory specified by LOG_STAGING_DIR, where `##` is the instance specified by SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. If recovery of the RS secondary server becomes necessary, and the logs have wrapped on the primary server, then the logs in `ifmxlog_##` can be used to recover the server. The files within `ifmxlog_##` are purged when no longer required.

Conditions that trigger delays

The time values in the BEGIN WORK, COMMIT WORK, and ROLLBACK WORK log records are used to calculate how to delay or stop the application of log files. The time values are calculated before passing the log pages to the recovery process.

When a BEGIN WORK statement is issued, the BEGIN WORK log record is not written until the first update activity is performed by the transaction; therefore, there can be a delay between the time that the BEGIN WORK statement is issued and when the BEGIN WORK log is written.

Interaction with secondary server updates

You must consider the interaction between secondary server updates and delayed application of log files. If updates are enabled, and the secondary server is updated, the updates are not applied until after the amount of time specified by DELAY_APPLY. Disabling secondary server updates, however, also disables Committed Read, which guarantees that every retrieved row is committed in the table at the time that the row is retrieved.

To retain the Committed Read isolation level, consider enabling secondary server updates using the UPDATABLE_SECONDARY configuration parameter, but removing the RS secondary server used for delayed application of log files from the Connection Manager service-level agreement list. Alternatively, consider moving the RS secondary server to a new SLA.

See "Database updates on secondary servers" on page 22-32 and *IBM Informix Administrator's Reference* for more information.

Specifying the log staging directory

You configure the log staging directory to specify where log files on RS secondary servers are staged before being applied to the database.

You must specify a staging directory for log files sent from the primary server before enabling delayed application of log files. No default staging directory is defined. The server creates additional directories in the directory specified by LOG_STAGING_DIR named `ifmxlog_##`, where ## is the instance specified by SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. The staged log files are automatically removed when they are no longer required. If the files within LOG_STAGING_DIR are lost, and the primary server has overwritten the logs, then the RS secondary server must be rebuilt.

You must ensure that the directory specified by LOG_STAGING_DIR exists and is secure. The directory must be owned by user **informix**, must belong to group **informix**, and must not have public read, write, or execute permission. If role separation is enabled, the directory specified by LOG_STAGING_DIR must be owned by the user or group that owns `$INFORMIXDIR/etc`. If the directory specified by LOG_STAGING_DIR is not secure, then the server cannot be initialized. The following message is written to the online message log if the directory is not secure:

`The log staging directory (directory_name) is not secure.`

You must also ensure that the disk contains sufficient space to hold all of the logs from the primary server, and that the directory does not contain staged logs from previous instances that are no longer being used.

See *IBM Informix Administrator's Reference* for more information.

To set LOG_STAGING_DIR:

1. Ensure that the directory in which logs are to be stored exists and is secure.
2. Edit the RS secondary server `onconfig` file.
3. Specify the staging directory as follows: `LOG_STAGING_DIR directory_name` where *directory_name* is the name of the directory in which to store the logs.
4. Restart the server.

You can also set the LOG_STAGING_DIR configuration parameter without restarting the server by using the **onmode -wf** command; however, the delayed application of log files must not be active when the command is run.

Delay application of log records on an RS secondary server

You can delay the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

You enable the delayed application of log files by setting the DELAY_APPLY configuration parameter. You can manually edit the `onconfig` file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of DELAY_APPLY you must also set LOG_STAGING_DIR. If DELAY_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

Set DELAY_APPLY using both a *number* and a *modifier*. Number can contain up to three digits and indicates the number of modifier units. Modifier is one of:

- D (or d) for days
- H (or h) for hours
- M (or m) for minutes
- S (or s) for seconds

See *IBM Informix Administrator's Reference* for more information.

To delay the application of log files on the RS secondary for four hours:

```
onmode —wf DELAY_APPLY=4H
```

To delay the application of log files for one day:

```
onmode -wf DELAY_APPLY=1D
```

To disable delayed application of log files:

```
onmode —wf DELAY_APPLY=0
```

Stop the application of log records

You can halt the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

You stop the application of log files on the RS secondary server by setting the STOP_APPLY configuration parameter. You can manually edit the onconfig file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of STOP_APPLY you must also set LOG_STAGING_DIR. If STOP_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

See *IBM Informix Administrator's Reference* for more information.

To stop the application of log files on the RS secondary server immediately, run the following command:

```
onmode —wf STOP_APPLY=1
```

To stop the application of log files at 11:00 p.m. on April 15th, 2009:

```
onmode —wf STOP_APPLY="2009:04:15-23:00:00"
```

To resume the normal application of log files

```
onmode —wf STOP_APPLY=0
```

Shared disk secondary servers

These topics provide an overview of setting up and configuring SD (shared disk) secondary servers in a high-availability environment. SD secondary server options are available with the standard version of IBM Informix.

Related reference:
"Data replication environments" on page 1-21

SD secondary server

A shared-disk (SD) secondary server participates in high-availability cluster configurations. In such configurations, the primary server and the SD secondary server share the same disk or disk array.

An SD secondary server does not maintain a copy of the physical database on its own disk space. Rather, it shares disks with the primary server.

SD secondary servers must be configured to access shared disk devices that allow concurrent access. Do not configure an SD secondary server that uses operating system buffering, such as NFS cross-mounted file systems. If the SD secondary server instance and the primary server instance both are located on a single machine, then both servers can access local disks. If the SD secondary server and the primary server are on separate physical machines, then they must be configured to access shared disk devices that appear locally attached, such as Veritas or GPFS™.

SD secondary servers can be used in conjunction with HDR secondary servers, with RS secondary servers, and with Enterprise Replication.

SD secondary servers can be added to a high availability environment very quickly, because they do not require a separate copy of the disk. Because the SD server shares the disk storage resources of the primary server, it is recommended that you provide some other means of disk backup, such as disk mirroring, or the use of an RS secondary server or an HDR secondary server.

The following restrictions affect the promotion of database server instances that are shared-disk secondary servers:

- An SD secondary server cannot be promoted to an RS secondary server.
- An SD secondary server cannot be promoted to a standard server that would exist outside the primary high availability environment.

Disk requirements for SD secondary servers

Except for disk requirements (which are shared with the primary server), hardware and software requirements are generally the same as for HDR secondary servers (See the Machine Notes for specific supported platforms). In addition, the primary disk system must be shared across the computers that are hosting the database servers. This means that the path to the dbspaces from the SD secondary is the same dbspace path as the primary server. see "Configuring clusters" on page 21-1.

Setting up a shared disk secondary server

A shared disk secondary server is set up by first setting the SDS_TIMEOUT configuration parameter. Next, the **onmode** utility is used to set the primary server alias that the SD secondary server uses to connect to the primary server. Then, the configuration file on the SD secondary is modified to include the appropriate options. Finally, the **oninit** utility is run to start the SD secondary server.

1. On the primary server, set the SDS_TIMEOUT configuration parameter in the `onconfig` file:

   `SDS_TIMEOUT x`

SDS_TIMEOUT specifies the amount of time in seconds that the primary server waits for a log position acknowledgment to be sent from the SD secondary server. See the *IBM Informix Administrator's Reference* for information about the SDS_TIMEOUT configuration parameter.

2. On the primary server, configure the alias name of the SD primary server:

```
onmode -d set SDS primary <alias>
```

The server name specified by *<alias>* becomes the primary server of the shared disk environment and the source of logs for the SD secondary server.

3. On the SD secondary server set the following configuration parameters in the configuration file:

```
SDS_ENABLE 1
SDS_PAGING <path 1>,<path 2>
SDS_TEMPDBS <dbname>,<dbspath>,<pagesize>,<offset>,<size>
```

SDS_ENABLE must be set to 1 (enable) on the secondary server to enable support of the shared disk environment. SDS_PAGING specifies the path to two files that are used to hold pages that might be required to be flushed between checkpoints. Each file acts as temporary disk storage for chunks of any page size. SDS_TEMPDBS is used to define the temporary dbspace used by the SD secondary server. This dbspace is dynamically created when the server is started (it is not created by running **onspaces**). See the *IBM Informix Administrator's Reference* for additional information about these parameters.

4. On the SD secondary server, set the following configuration parameters to match those on the primary server:

   - ROOTNAME
   - ROOTPATH
   - ROOTOFFSET
   - ROOTSIZE
   - PHYSFILE
   - LOGFILES
   - LOGSIZE

   Map the other configuration parameters to match those of the primary server with the exception of DBSERVERALIASES, DBSERVERNAME, and SERVERNUM.

   For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1.

5. On the SD secondary server, optionally set the UPDATABLE_SECONDARY configuration parameter to a positive integer if you want to enable client applications to perform update, insert, and delete operations on the secondary server.

6. Add an entry to the `sqlhosts` file (or for Windows systems, the SQLHOSTS registry key) for the primary server:

   primary_dbservername nettype primary_hostname servicename

7. Start the SD secondary server using the **oninit** command.

   The primary server must be active before starting the SD secondary server.

   When a secondary server is started, it must first process any open transactions using fast recovery mode. Client applications can connect to the server only after all of the transactions open at the startup checkpoint have either committed or rolled back. After open transactions have been processed, client

applications can connect to the server as they normally do. You must examine the online.log file on the secondary server to verify that it has completed processing open transactions.

The following table illustrates the preceding steps so that you can clearly determine which steps are performed on the primary server and which are performed on the secondary server.

Table 21-5. Steps to start an SD secondary server for the first time

Step	On the primary	On the secondary
1.	Set the SDS_TIMEOUT configuration parameter in the onconfig file: `SDS_TIMEOUT x`	
2.	Configure the alias name of the SD primary server: `onmode -d set SDS primary <alias>`	

3		Set configuration parameters:
		`SDS_ENABLE 1`
		`SDS_PAGING <path 1>,<path 2>`
		`SDS_TEMPDBS <dbsname>,<dbspath>,`
		`<pagesize>,<offset>,`
		`<size>`

4		Set configuration parameters to match those on the primary server: • ROOTNAME • ROOTPATH • ROOTOFFSET • ROOTSIZE • PHYSFILE • LOGFILES • LOGSIZE
5		Optionally set the UPDATABLE_SECONDARY configuration parameter to a positive integer.
6		Add an entry to the sqlhosts file (or for Windows systems, the SQLHOSTS registry key) for the primary server: `dbservername nettype hostname servicename`
7		Start the SD secondary server `oninit`

There is increased memory usage in the LGR memory pool when a secondary server is added.

See the *IBM Informix Administrator's Reference* for information about configuration parameters.

Obtain SD secondary server statistics

Use the **onstat** utility or system-monitoring interface (SMI) tables to view SD secondary server statistics.

Use **onstat -g sds** to view SD secondary server statistics. The output of the **onstat** utility depends on whether the utility is run on the primary or secondary server.

Query the **syssrcsds** table to obtain information about shared disk statistics on the primary server.

Query the **systrgsds** table to obtain information about shared disk statistics on the secondary server.

For information about **onstat** and SMI tables see the *IBM Informix Administrator's Reference*.

SD secondary server configuration

Promote an SD secondary server to a primary server

Convert an SD secondary server to a primary server by issuing the following command on the SD secondary server:

```
onmode -d set SDS primary <alias>
```

An SD secondary server cannot be converted to a standard server.

Convert a primary server to a standard server

You can convert a primary server to a standard server and disconnect it from the shared disk environment using the following command on the primary server:

```
onmode -d clear SDS primary <alias>
```

SD secondary server security

SD secondary servers support similar encryption rules as HDR. See "Database server configuration requirements for clusters" on page 21-3 for details.

Encryption can be enabled or disabled between any primary and secondary server pair. That is, you can encrypt traffic between the primary server and one SD secondary server and not encrypt traffic between the primary server and another SD secondary server.

See the "Server Multiplexer Group (SMX) connections" on page 21-12 topic for additional information about setting up and configuring encryption between primary servers and SD secondary servers.

Chapter 22. Cluster administration

This chapter describes various administrative tasks, some optional, for monitoring and maintaining a cluster. For example, load-balancing to optimize performance, ensuring security.

How data replication works

These topics describe the mechanisms that the database server uses to perform replication of data to secondary servers. For instructions on how to set up, start, and administer the various types of secondary servers, see the table

Table 22-1. Secondary server setup information

Secondary server type	See
HDR secondary	See Chapter 21, "High-availability cluster configuration," on page 21-1, and information about starting an HDR pair using external backup and restore in the *IBM Informix Backup and Restore Guide*.
RS secondary	See "Remote standalone secondary servers" on page 21-10.
SD secondary	See "Shared disk secondary servers" on page 21-18

How data initially replicates

HDR secondary and RS secondary servers use storage-space backups and logical-log backups (both those backed up to tape and those on disk) to perform an initial replication of the data on the primary database server to the secondary database server.

SD secondary servers do not require a backup and restore from the primary server because SD secondary servers share the same disks as the primary.

To replicate data:
1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers.
2. Register user-defined types, user-defined routines, and DataBlade modules on the primary database server only.
3. To synchronize the data managed by the two database servers, create a level-0 backup of all the storage spaces on the primary database server.
4. Restore all the storage spaces from the backup on the secondary database server in the data-replication pair.

   The secondary database server that you restored from a storage-space backup in the previous step then reads all the logical-log records generated since that backup from the primary database server.

   The database server reads the logical-log records first from any backed-up logical-log files that are no longer on disk and then from any logical-log files on disk.

For detailed instructions about replicating data, see "Starting HDR for the First Time" on page 21-6. The *IBM Informix Backup and Restore Guide* explains how to start replication using ON-Bar.

You must perform the initial backup with a storage-space backup. You cannot use data-migration utilities such as **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical in order for data replication to work.

Reproduction of updates from the primary database server

All secondary server types use logs to replicate data from the primary server. RS secondary servers require index page logging, but HDR secondary and SD secondary servers also use index page logging if it is enabled. For HDR secondary and RS secondary servers, updates made to the primary server are replicated on the secondary server by having the primary server send all its logical-log records to the secondary server as the logs are generated. HDR secondary and RS secondary servers receive the logical-log records generated on the primary server and apply them to their own dbspaces. For SD secondary servers, the primary server sends the log position of the logical log page to the SD secondary server. Using the log position received from the primary server, the SD secondary server reads the logical log page from disk and applies it to the memory data buffers.

Restriction: The database server cannot replicate updates to databases that do not use transaction logging.

How the log records are sent to HDR secondary servers

As shown in Figure 22-1 on page 22-3, when the primary database server starts to flush the contents of the logical-log buffer in shared memory to the logical log on disk, the database server also copies the contents of the logical-log buffer to a *data-replication buffer* on the primary database server. The primary database server then sends these logical-log records to the HDR secondary database server.

The HDR secondary database server receives the logical-log records from the primary database server into a shared-memory *reception buffer* (which the database server automatically adjusts to an appropriate size for the amount of data being sent). The secondary database server then applies the logical-log records through logical recovery.

How the log records are sent to RS secondary and SD secondary servers

For RS secondary servers, the log pages can be cached as they are being flushed to disk, or they can be read directly from the log on disk. For SD secondary servers, the logs are not sent at all, but instead only the log position is sent.

HDR data replication buffers

The data replication buffers are part of the virtual shared memory that the primary database server manages. The data replication buffers hold logical-log records before the primary database server sends them to the HDR secondary database server. The data replication buffers are the same size as the logical-log buffers. The following figure shows this concept.
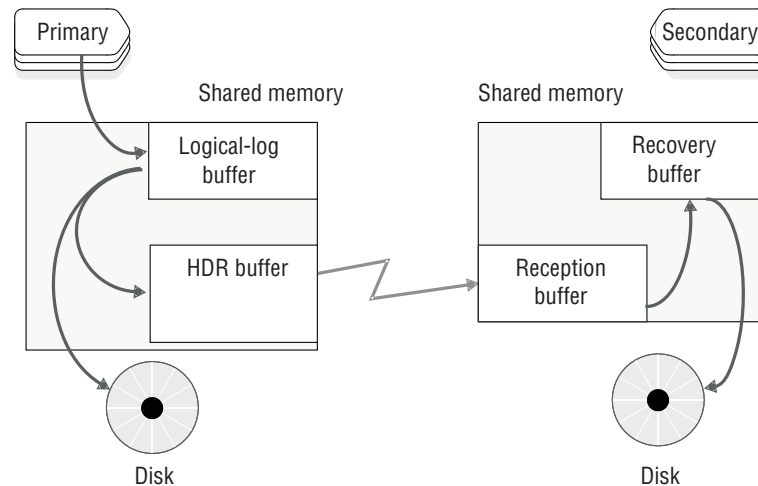
Figure 22-1. How logical-log records are sent from the primary database server to the secondary database server

When log records are sent

The primary database server sends the contents of the data replication buffer to an HDR secondary server either *synchronously* or *asynchronously*. The value of the ONCONFIG configuration parameter DRINTERVAL determines whether the database server uses synchronous or asynchronous updating. For more information about DRINTERVAL, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

For RS secondary servers, index pages are sent asynchronously from the primary server using index page logging. For SD secondary servers, only the log page position within the logical log is replicated.

HDR synchronous updating

If you set DRINTERVAL to -1, data replication to the HDR secondary server occurs *synchronously*. As soon as the primary database server writes the logical-log buffer contents to the HDR buffer, it sends those records from the buffer to the HDR secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the HDR secondary database server that the records were received.

With synchronous updating, no transactions committed on the primary database server are left uncommitted or partially committed on the HDR secondary database server if a failure occurs.

Log records sent to a secondary server must be applied in the order in which they were received. When the log transmission buffer contains many log records, you might see a delay in the application of those log records after they have been received on the secondary server.

You might receive error -7350 "Attempt to update a stale version of a row" if you use buffered logging within a high-availability cluster. You can avoid the error by using unbuffered logging.

HDR asynchronous updating

If you set DRINTERVAL to any value other than **-1**, data replication occurs *asynchronously* to the HDR secondary server. The primary database server flushes the logical-log buffer after it copies the logical-log buffer contents to the HDR buffer. Independent of that action, the primary database server sends the contents of the HDR buffer across the network when one of the following conditions occurs:

- The HDR buffer becomes full.
- The time interval, specified by the DRINTERVAL configuration parameter on the primary database server, has elapsed since the last time that records were sent to the secondary database server.

This method of updating might provide better performance than synchronous updating. However, as the following section explains, transactions can be lost.

Lost-and-found transactions: With asynchronous updating, a transaction committed on the primary database server might not be replicated on the secondary database server. This situation can result if a failure occurs after the primary database server copies a commit record to the HDR buffer but before the primary database server sends that commit record to the secondary database server.

If the secondary database server is changed to a standard database server after a failure of the primary database server, it rolls back any open transactions. These transactions include any that were committed on the primary database server but for which the secondary database server did not receive a commit record. As a result, transactions are committed on the primary database server but not on the secondary database server. When you restart data replication after the failure, the database server places all the logical-log records from the lost transactions in a file (which the DRLOSTFOUND configuration parameter specifies) during logical recovery of the primary database server. The following figure illustrates the process.
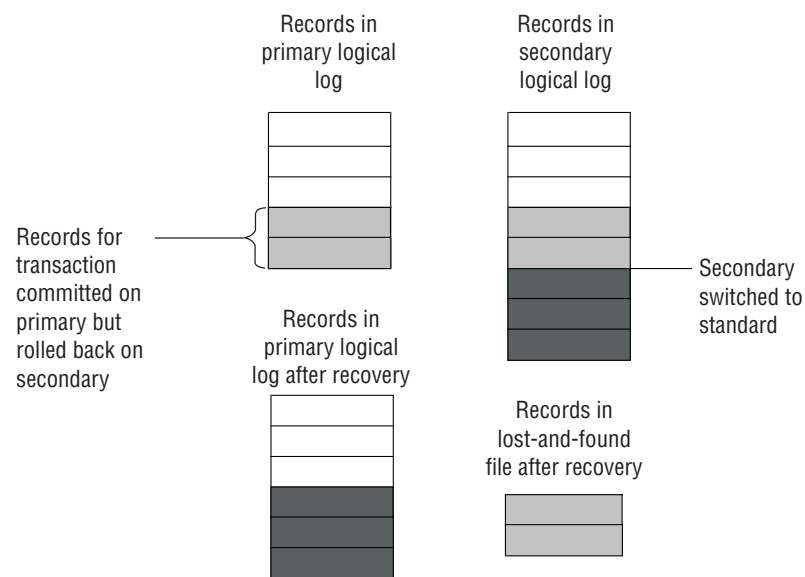
Figure 22-2. Using a lost-and-found file

If the lost-and-found file is created on the computer that is running the primary database server after it restarts data replication, a transaction has been lost. The

database server cannot reapply the transaction records in the lost-and-found file because conflicting updates might have occurred while the secondary database server was acting as a standard database server.

To reduce the risk of a lost transaction without running data replication in synchronous mode, use unbuffered logging for all the databases. This method reduces the amount of time between the writing and transfer of the transaction records from the primary database server to the secondary database server.

Threads that handle data replication

The primary database server starts specialized threads to support data replication. As the following figure shows, a thread called **drprsend** on the primary database server sends the contents of the primary server buffer across the network to a thread called **drsecrcv** on the secondary database server.
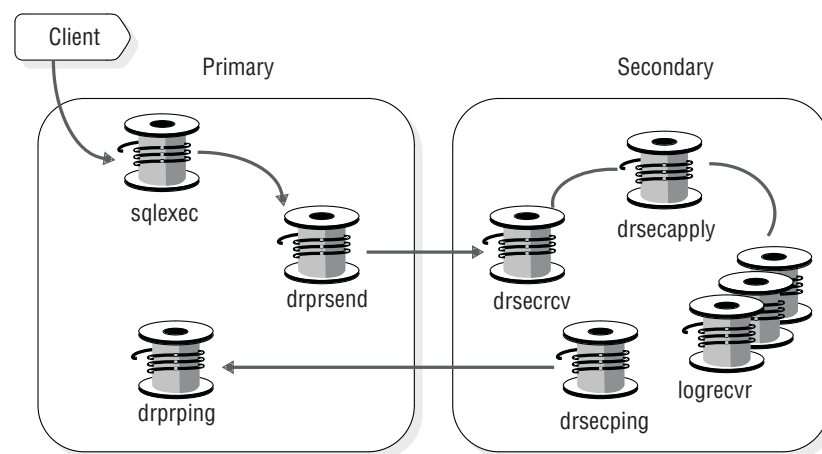
Figure 22-3. Threads that manage data replication

A thread called **drsecapply** on the secondary database server copies the contents of the reception buffer to the recovery buffer. The **logrecvr** thread (or threads) performs logical recovery with the contents of the recovery buffer, applying the logical-log records to the dbspaces that the secondary database server manages. The OFF_RECVRY_THREADS configuration parameter specifies the number of **logrecvr** threads used.

The remaining threads that the database server starts are the **drprping** and **drsecping** threads, which are responsible for sending and receiving the messages that indicate if the two database servers are connected.

To support RS secondary servers, the primary server also checks to see whether there are RS secondary servers attached and if so, copies the page to a log cache which is used to send that page to the RS secondary server. See the following figure.
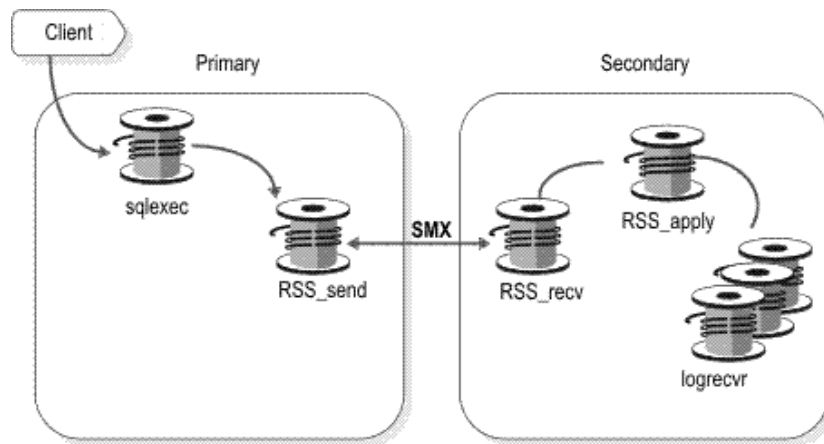
Figure 22-4. Threads that manage data replication for RS secondary servers

The RSS_Send thread transmits the log page to the RS secondary server. It is possible that the next page that must be sent is not in the log cache. In that case, the RSS_Send thread reads the log pages directly from disk. The RSS_Send thread interfaces with SMX to send data in a fully duplexed manner. With full duplex communication, the thread does not wait for an acknowledgment from the RS secondary server before sending the next buffer. Up to 32 buffer transmissions are sent before the primary server requires an acknowledgment from the RS secondary server. If the 32 buffer limit is reached, then the send thread waits for the RSS_Recv thread to receive an acknowledgment from the RS secondary server.

On the RS secondary server, RSS_Recv interfaces with SMX to receive the log pages from the primary server.

Checkpoints between database servers

Checkpoints between database servers in a replication pair are synchronous. The primary server waits for the HDR secondary server to acknowledge that it received the checkpoint log record before the primary server completes its checkpoint.

If the checkpoint does not complete within the time that the DRTIMEOUT configuration parameter specifies, the primary database server assumes that a failure has occurred.

Tip: Synchronize the time on the operating systems of both database servers in the pair and set DRTIMEOUT on both servers to the same value.
Checkpoints between a primary server and an RS secondary server are asynchronous, and require index page logging to be enabled.

Related concepts:

"Cluster failures" on page 24-3

"Index page logging" on page 21-11

How data synchronization is tracked

To keep track of synchronization, each database server in the pair keeps track of the following information in its reserved page:

* The ID of the logical-log file that contains the last completed checkpoint
* The position of the checkpoint record within the logical-log file
* The ID of the last logical-log file sent (or received)

- The page number of the last logical-log record sent (or received)

The database servers use this information internally to synchronize data replication.

Data replication configuration examples

These topics describe some examples of how a data replication environment can be configured.

Remote standalone secondary configuration examples

The following figure illustrates an example of a configuration consisting of multiple RS secondary servers. This configuration would be useful in a situation where the primary server is located a long distance from the RS secondary servers or if the network speed between the primary server and the RS secondary server is slow or erratic. Because RS secondary servers use fully duplexed communication protocols, and do not require checkpoints processed in SYNC mode, the additional servers must not significantly affect the performance of the primary server.
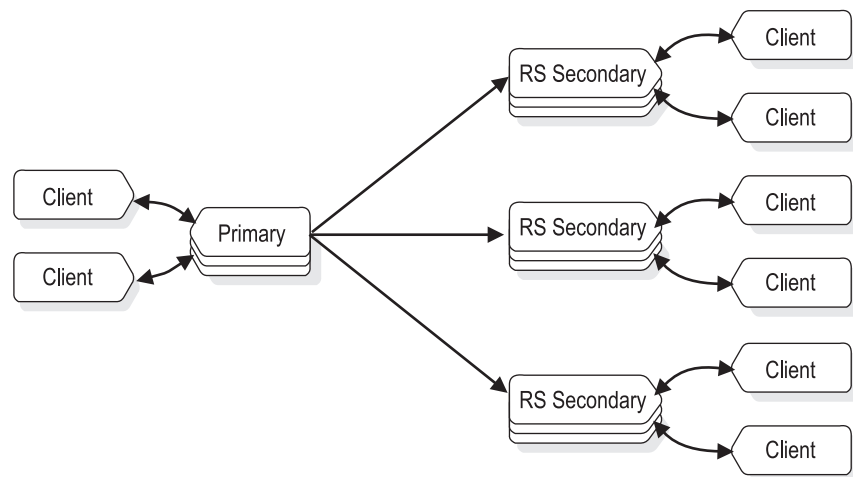
Figure 22-5. Primary server with three RS secondary servers

The next illustration shows an example of a configuration of an RS secondary server along with an HDR secondary server. In this example, the HDR secondary provides high availability while the RS secondary provides additional disaster recovery if both the primary and HDR secondary servers are lost. The RS secondary server can be geographically remote from the primary and HDR secondary servers so that a regional disruption such as an earthquake or flood would not affect the RS secondary server.
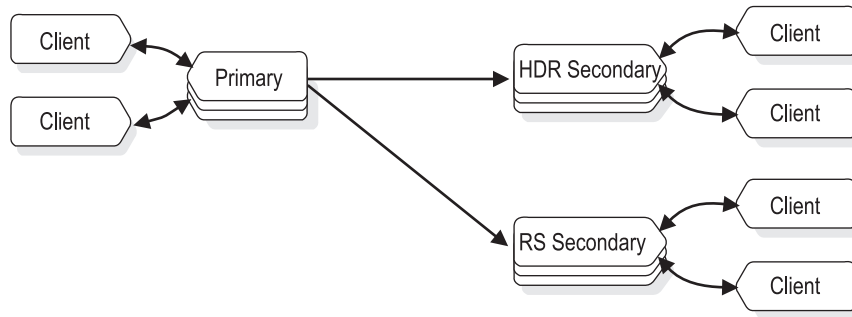
Figure 22-6. Primary server with HDR secondary and RS secondary servers

If a primary database server fails, it is possible to convert the existing HDR secondary server into the primary server, as in the following diagram:
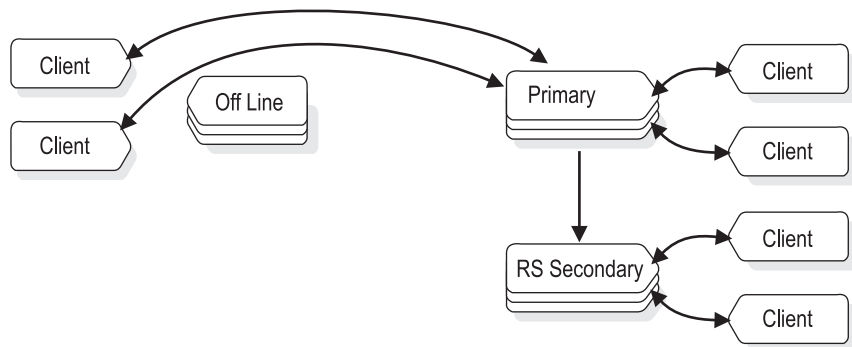
Figure 22-7. Failover of primary server to HDR secondary server

If the original primary is going to be offline for an extended period of time, then the RS secondary server can be converted to an HDR secondary server. Then when the original primary comes back online, it can be configured as an RS secondary server, as in the following illustration:
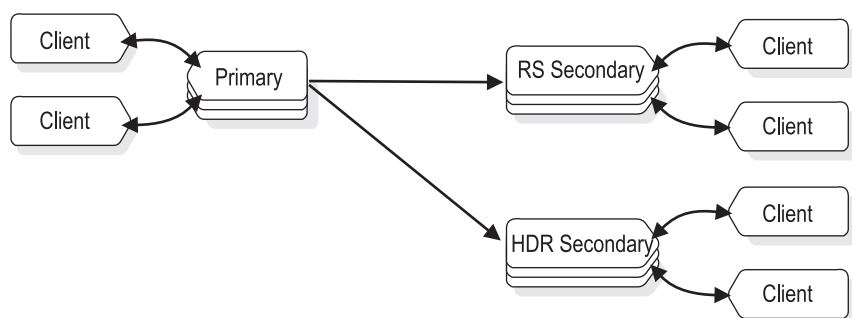
Figure 22-8. RS secondary server assuming role of HDR secondary server

Shared disk secondary configuration examples

The following figure shows an example of a primary server with two SD secondary servers. In this case the role of the primary server can be transferred to either of the two SD secondary servers. This is true whether the primary must be taken out of service because of a planned outage, or because of failure of the
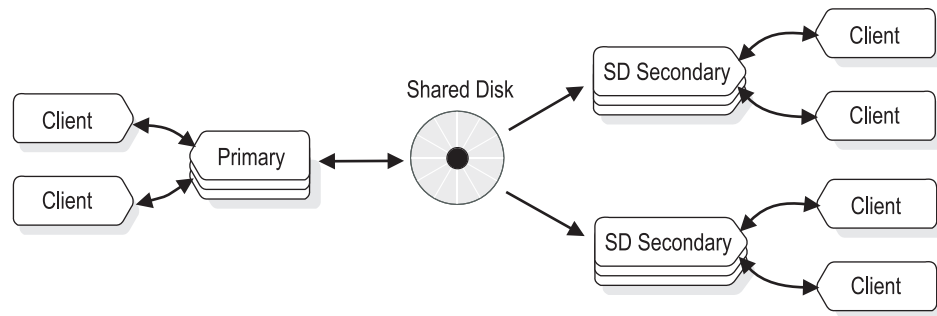
primary server.

Figure 22-9. Primary server configured with two SD secondary servers

Because both of the SD secondary servers are reading from the same disk
subsystem, they are both equally able to assume the primary server role. The
following figure illustrates a situation in which the primary server is offline.
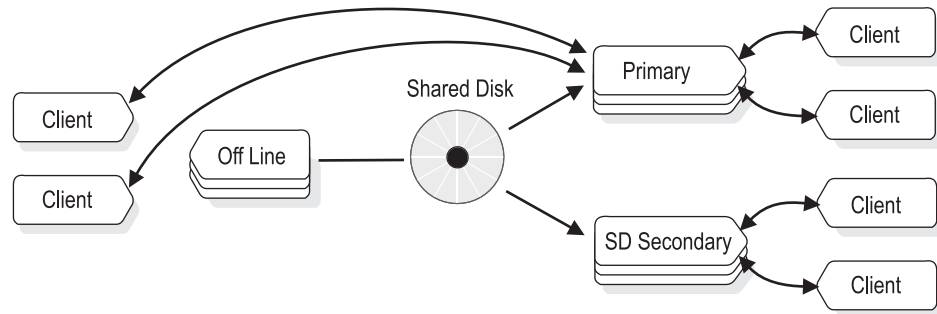
Figure 22-10. SD secondary server assuming role of primary server

There are several ways to protect against hardware failure of the shared disk.
Probably the most common way is to configure the disk array based on RAID
technology (such as RAID-5). Another way to protect against disk failure is to use
SAN (Storage Area Technology) to include some form of remote disk mirroring.
Since SAN disks can be located a short distance from the primary disk and its
mirror, this provides a high degree of availability for both the planned and
unplanned outage of either the server or of the disk subsystem. The following
illustration depicts such a configuration:
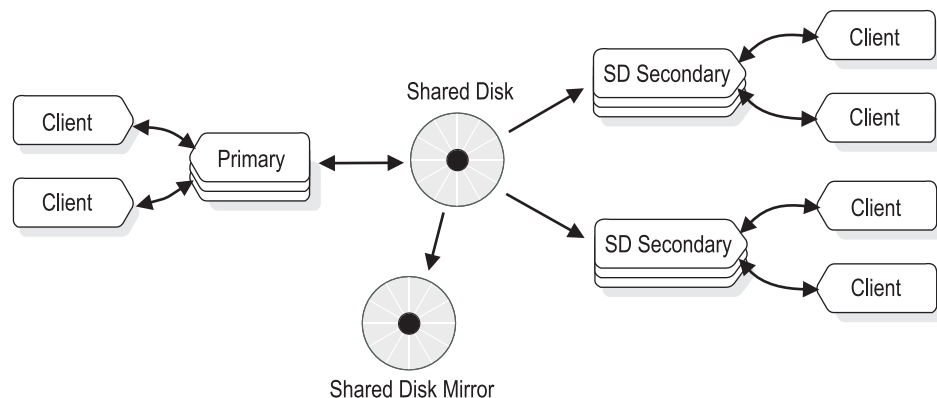
Figure 22-11. Primary server and SD secondary servers with mirrored disks

In the event of a disk failure, the servers can be reconfigured as in the following illustration:
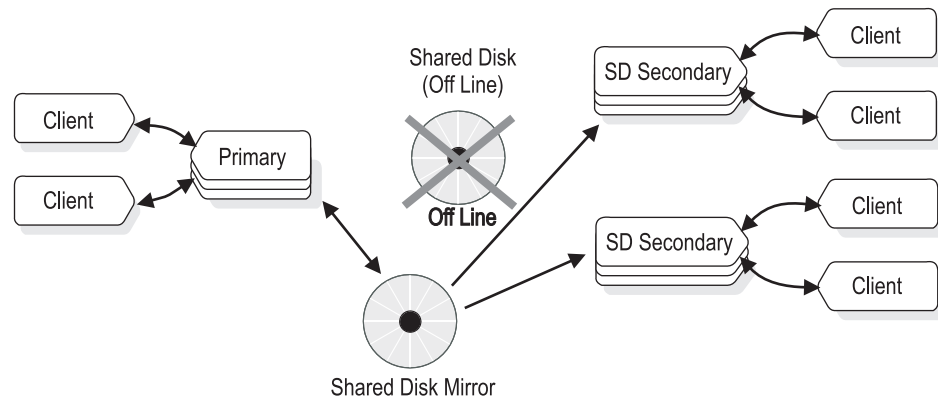
Figure 22-12. Shared disk mirror after failure of primary shared disk

In addition to configuring a mirrored disk subsystem as in the previous illustration, you might want to configure additional servers. For example, you might want to use the primary and two SD secondary servers within a single blade server enclosure. By placing the server group within a single blade server, the blade server itself can become a failure point. The configuration in the following illustration is an attractive solution when you must periodically increase read processing ability such as when performing large reporting tasks.
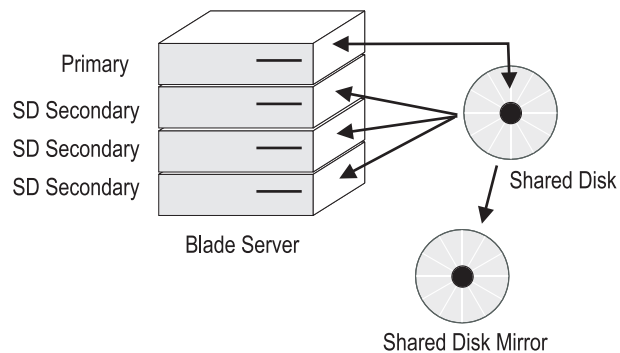
Figure 22-13. Primary and SD secondary servers in a blade server

You might decide to avoid the possible failure point of a single blade server by using multiple blade servers, as in the following illustration.
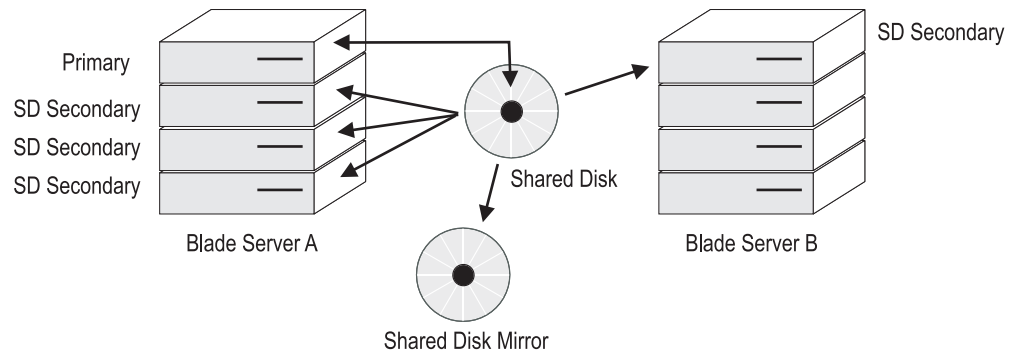
Figure 22-14. Multiple blade server configuration to prevent single point of failure

In the previous illustration, if Blade Server A fails, it would be possible to transfer the primary server role to the SD secondary server on Blade Server B. Since it is possible to bring additional SD secondary servers online very quickly, it would be possible to dynamically add additional SD secondary servers to Blade Server B as in the following illustration.
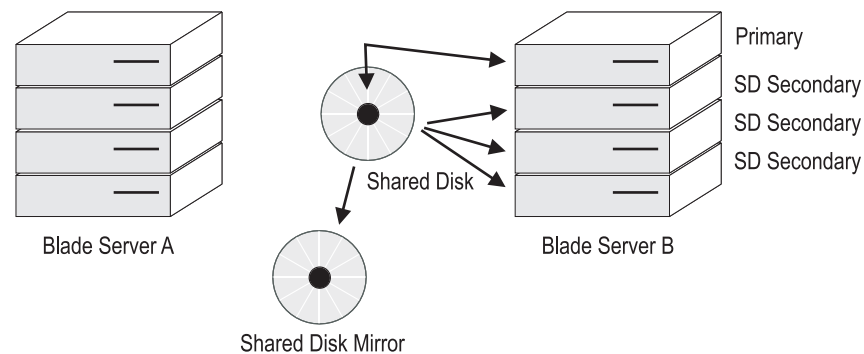
Figure 22-15. Failover after failure of blade server

Because of limits on the distance between the primary and mirrored disks that disk mirroring can support, you might be concerned about using shared disks and relying on shared disk mirroring to provide disk availability. For example, you might want significant distance between the two copies of the disk subsystem. In this case, you might choose to use either an HDR secondary or an RS secondary server to maintain the secondary copy of the disk subsystem. If the network connection is fairly fast (that is, if a ping to the secondary server is less than 50 milliseconds) you must consider using an HDR secondary server. For slower network connections, consider using an RS secondary server. The following illustration shows an example of an HDR secondary server in a blade server configuration.
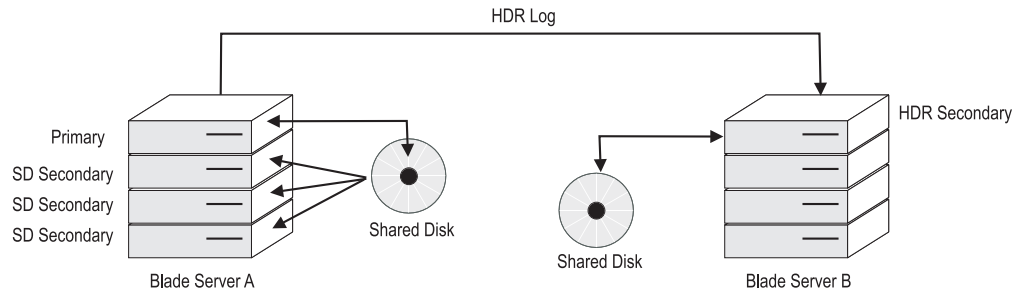
Figure 22-16. HDR secondary server in blade server configuration

In the configuration shown in the previous illustration, if the primary node fails, but the shared disks are intact and the blade server is still functional, it is possible to transfer the primary server role from the first server in Blade Server A to another server in the same blade server. Changing the primary server would cause the source of the remote HDR secondary server to automatically reroute to the new primary server, as illustrated in the following diagram:
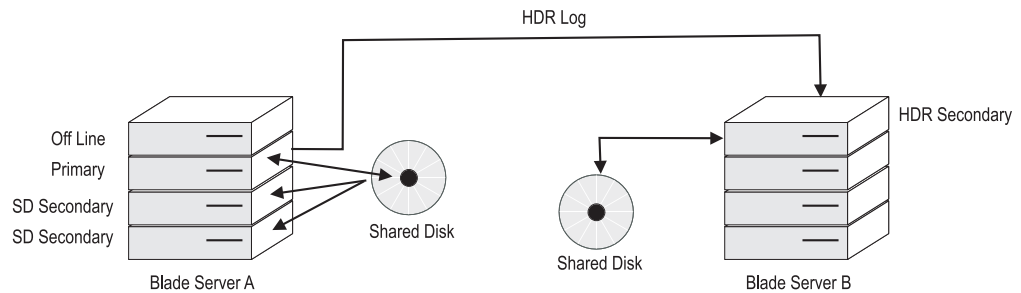
Figure 22-17. Failover of primary server to SD secondary server in blade server configuration

Suppose, however, that the failure described in the previous illustration was not a blade within the blade server, but the entire blade server. In this case you might be required to fail over to the HDR secondary. Since starting an SD secondary server is very quick, you can easily add additional SD secondary servers. Note that the SD secondary server can only work with the primary node; when the primary has been transferred to Blade Server B, then it becomes possible to start SD secondary servers on Blade Server B as well, as shown in the following illustration.
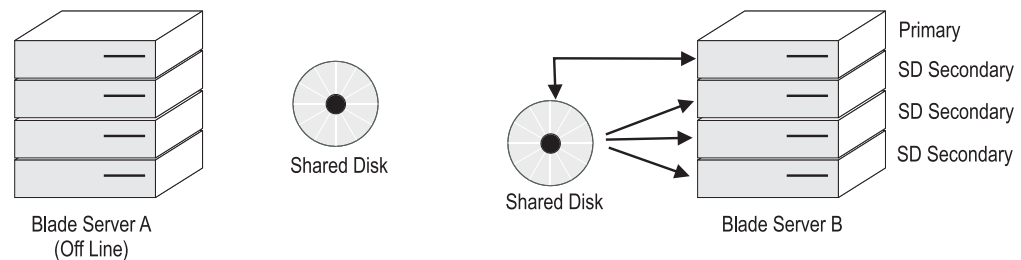
Figure 22-18. Failure of entire blade server

Enterprise Replication as part of the recoverable group

While Enterprise Replication does not support a SYNC (synchronous) mode of operation, it does provide the ability to support environments with multiple active servers. During a failover event, Enterprise Replication is able to reconcile database differences between two servers. You must consider Enterprise Replication as a means of improving synchronization between servers because each Enterprise

Replication system maintains an independent logging system. A configuration using Enterprise Replication is shown in the following figure.
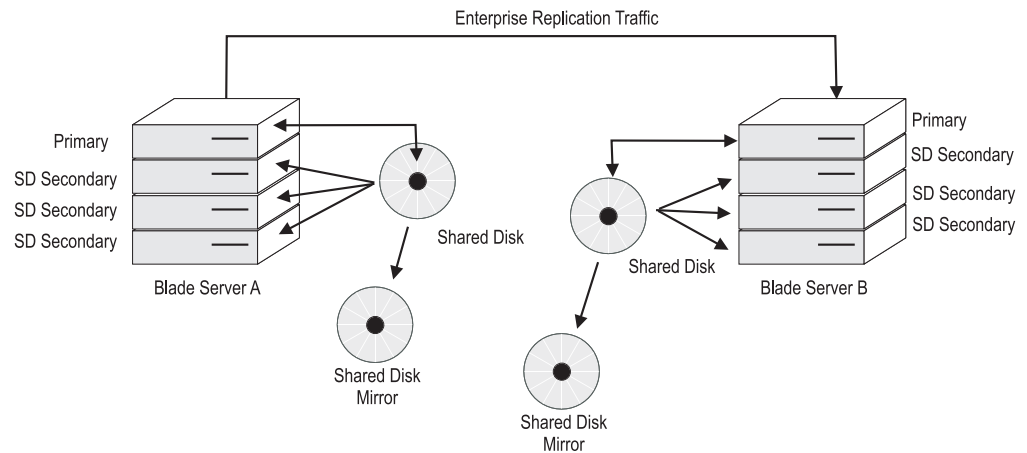
Figure 22-19. Configure Enterprise Replication as part of the recoverable group

High-availability clusters with Enterprise Replication configuration example

Suppose you require Enterprise Replication between two high-availability server clusters configured as follows:

Cluster 1:
- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2
- RS secondary server 1
- RS secondary server 2

Cluster 2:
- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2
- RS secondary server 1
- RS secondary server 2

Suppose further that each of the servers is named according to the following convention:
- First three characters: name of enterprise
- Character 4: host short number
- Characters 5, 6, and 7: cluster number
- Characters 8, 9, and 10: server type: "pri" for primary server, "sec" for secondary server
- Characters 11, 12, and 13: connection type: "shm" or "tcp"

For example, a server with the name: **srv4_1_pri_shm** is described as follows:

- srv = name of enterprise
- 4 = host short number
- _1_ = cluster number
- pri = this is a primary server
- shm = connection type uses shared memory communication

The following entries in the `sqlhosts` file would support the previous configuration:

```
srv4_1_pri_shm onipcshm sun-mach4 srv4_1_pri_shm
srv4_1_sec_shm onipcshm sun-mach4 srv4_1_sec_shm
srv5_1_rss_shm onipcshm sun-mach5 srv5_1_rss_shm
srv5_1_sds_shm onipcshm sun-mach5 srv5_1_sds_shm
srv6_1_rss_shm onipcshm sun-mach6 srv6_1_rss_shm
srv6_1_sds_shm onipcshm sun-mach6 srv6_1_sds_shm
srv_1_cluster group - - i=1
srv4_1_pri_tcp ontlitcp sun-mach4 21316 g=srv_1_cluster
srv4_1_sec_tcp ontlitcp sun-mach4 21317 g=srv_1_cluster
srv5_1_rss_tcp ontlitcp sun-mach5 21316 g=srv_1_cluster
srv5_1_sds_tcp ontlitcp sun-mach5 21317 g=srv_1_cluster
srv6_1_rss_tcp ontlitcp sun-mach6 21316 g=srv_1_cluster
srv6_1_sds_tcp ontlitcp sun-mach6 21317 g=srv_1_cluster

srv4_2_pri_shm onipcshm sun-mach4 srv4_2_pri_shm
srv4_2_sec_shm onipcshm sun-mach4 srv4_2_sec_shm
srv5_2_rss_shm onipcshm sun-mach5 srv5_2_rss_shm
srv5_2_sds_shm onipcshm sun-mach5 srv5_2_sds_shm
srv6_2_rss_shm onipcshm sun-mach6 srv6_2_rss_shm
srv6_2_sds_shm onipcshm sun-mach6 srv6_2_sds_shm
srv_2_cluster group - - i=2
srv4_2_pri_tcp ontlitcp sun-mach4 21318 g=srv_2_cluster
srv4_2_sec_tcp ontlitcp sun-mach4 21319 g=srv_2_cluster
srv5_2_rss_tcp ontlitcp sun-mach5 21318 g=srv_2_cluster
srv5_2_sds_tcp ontlitcp sun-mach5 21319 g=srv_2_cluster
srv6_2_rss_tcp ontlitcp sun-mach6 21318 g=srv_2_cluster
srv6_2_sds_tcp ontlitcp sun-mach6 21319 g=srv_2_cluster
```

Example of a complex failover recovery strategy

This topic describes a three-tiered server approach for achieving maximum availability in the case of a large region-wide disaster.

In general, an HDR Secondary server provides backup for SD secondary servers and provides support for a highly available system which is geographically remote from the main system. RS secondary servers provide additional availability for the HDR secondary and are viewed as a disaster-availability solution. If you must use an RS secondary server for availability, then you are forced to manually rebuild the other systems by performing backup and restore in order to return to normal operation. To further understand this, a scenario is presented in which a large region-wide disaster occurs, such as a hurricane.

To provide maximum availability to survive a regional disaster requires *layered* availability. The first layer provides availability solutions to deal with transitory local failures. For example, this might include having a couple of blade servers attached to a single disk subsystem running SD secondary servers. Placing the SD secondary servers in several locations throughout your campus makes it possible to provide seamless failover in the event of a local outage.

You might want to add a second layer to increase availability by including an alternative location with its own copy of the disks. To protect against a large regional disaster, you might consider configuring an HDR secondary server located

some distance away, perhaps hundreds of miles. You might also want to make the remote system a blade server or some other multiple-server system. By providing this second layer, if a fail-over occurs and the remote HDR secondary became the primary, then it would be possible to easily start SD secondary servers at the remote site.

However, even a two-tiered approach might not be enough. A hurricane in one region can create tornadoes hundreds of miles away. To protect against this, consider adding a third tier of protection, such as an RS secondary server located one or more thousand miles away. This three-tier approach provides for additional redundancy that can significantly reduce the risk of an outage.
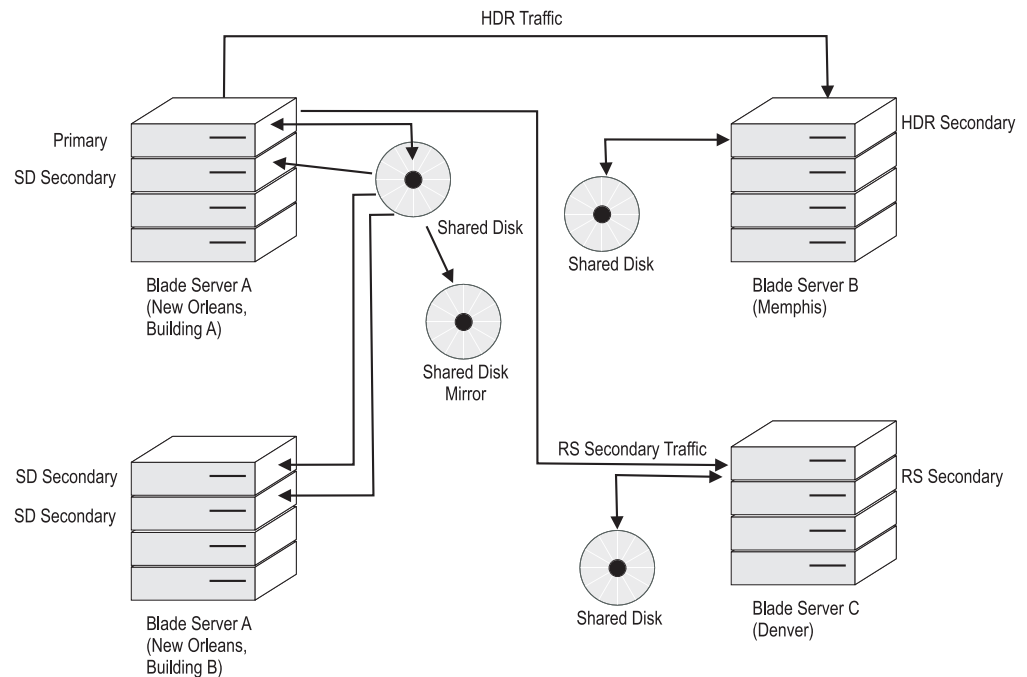
Figure 22-20. Configuration for three-tiered server availability

Now suppose that a local outage occurred in Building-A on the New Orleans campus. Perhaps a pipe burst in the machine room causing water damage to the blade server and the primary copy of the shared disk subsystem. You can switch the role of primary server to Building-B by running onmode -d make primary servername on one of the SD secondary servers running on the blade server in Building-B. This would cause all other secondary nodes to automatically connect to the new primary node.
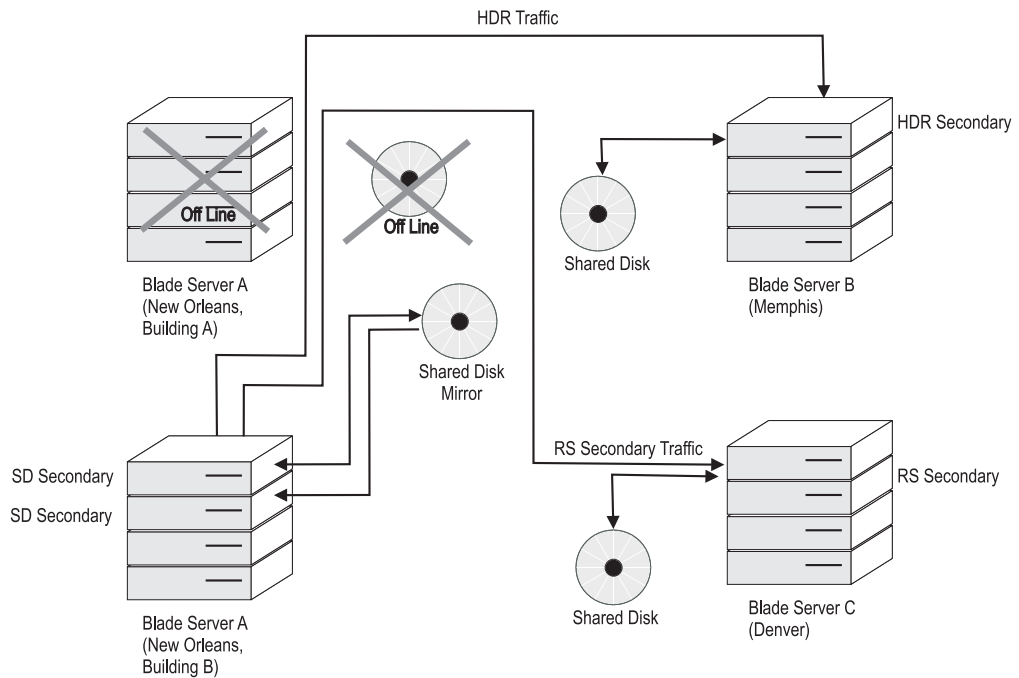
Figure 22-21. First tier of protection

If there be a regional outage in New Orleans such that both building A and building B were lost, then you can shift the primary server role to Memphis. In addition, you might also want to make Denver into an HDR secondary and possibly add additional SD secondary servers to the machine in Memphis.
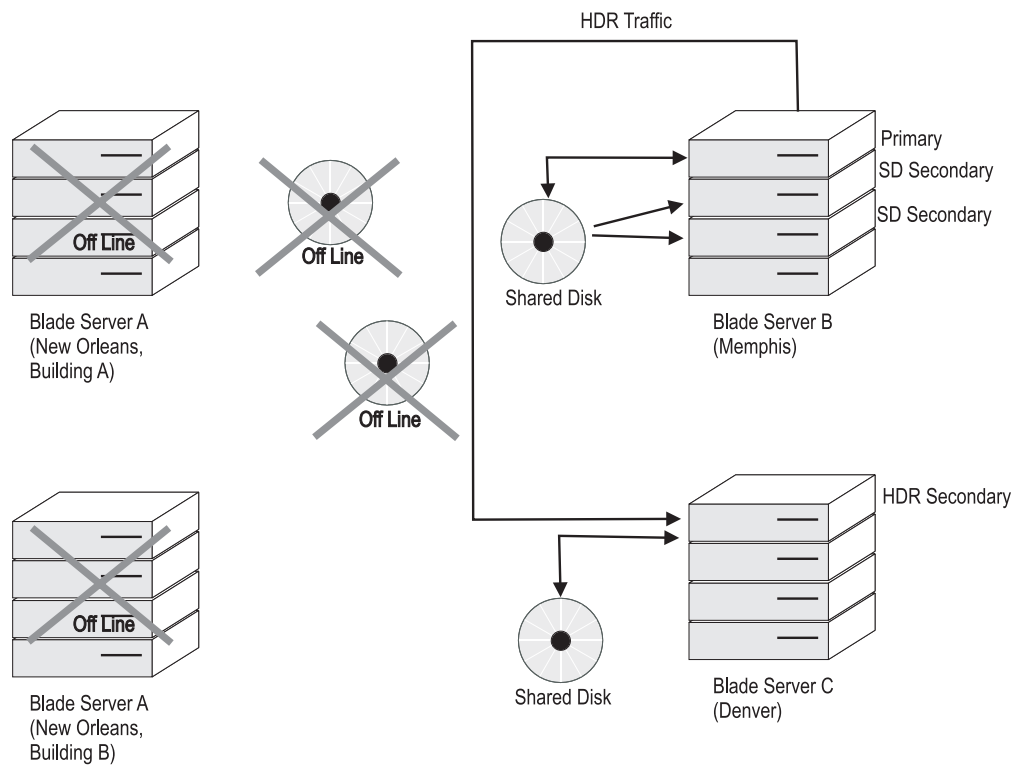
Figure 22-22. Second tier of protection

An even larger outage which affected both sites would require switching to the most remote system.
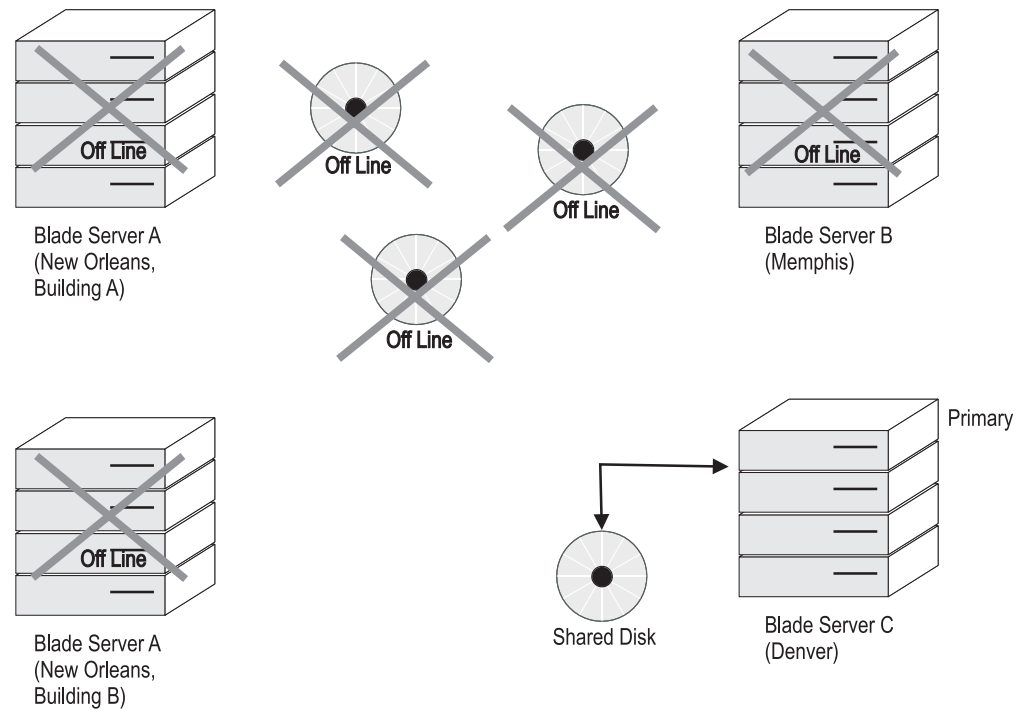
Figure 22-23. Third tier of protection

Table 22-2. Suggested configurations for various requirements

Requirement	Suggested configuration
You periodically must increase reporting capacity	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware availability, but are concerned about server failures	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware mirroring, but also want a second set of servers that are able to be brought online if the main operation is lost (and the limitations of mirrored disks are not a problem)	Consider using two blade centers running SD secondary servers at the two sites
You want to have a backup site some moderate distance away, but cannot tolerate any loss of data during failover	Consider using two blade centers with SD secondary servers on the main blade center and an HDR secondary on the remote.
You want to have a highly available system in which no transaction is ever lost, but must also have a remote system on the other side of the world	Consider using an HDR secondary located nearby running SYNC mode and an RS secondary server on the other side of the world
You want to have a high availability solution, but because of the networks in your region, the best response time from a ping is about 200 ms	Consider using an RS secondary server

Table 22-2. Suggested configurations for various requirements (continued)

Requirement	Suggested configuration
You want a backup site but you do not have any direct communication with the backup site	Consider using Continuous Log Restore with backup and recovery
You can tolerate a delay in the delivery of data as long as the data arrives eventually; however you must have quick failover in any case	Consider using SD secondary servers with hardware disk mirroring in conjunction with ER.
You require additional write processing power, can tolerate some delay in the delivery of those writes, require something highly available, and can partition the workload	Consider using ER with SD secondary servers

Troubleshooting high-availability cluster environments

A high-availability cluster environment requires little or no additional troubleshooting when compared with a stand-alone server environment. This topic explains the terminology used to describe high-availability cluster environments and provides some common troubleshooting procedures.

You use the diagnostic tools to display the configuration of a high-availability environment and to verify that your secondary servers are set up correctly to update data.

Transactions are processed by the servers very quickly. The **onstat** commands display status information only for the instant the command was run.

To update data on secondary servers, IBM Informix creates *proxy distributors* on both the primary and the secondary database servers. Each proxy distributor is assigned an ID that is unique within the cluster. The proxy distributor is responsible for sending DML update requests from secondary servers to the primary server. Secondary servers determine how many instances of the proxy distributors to create based on the UPDATABLE_SECONDARY setting in the secondary server's onconfig file.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the ENCRYPT_SMX configuration parameter on the secondary server. See "Enable SMX encryption" on page 21-12 for more information.

When initializing an updatable secondary server in a high-availability cluster, the server remains in fast recovery mode until all open transactions, including open and prepared XA transactions, are complete. Applications cannot connect to the server while it is in fast recovery mode. The server remains in fast recovery mode until all open transactions are either committed or rolled back.

Use the **onstat -g proxy** command on the primary server to view information about all proxy distributors in the high-availability cluster.

```
onstat -g proxy

Secondary    Proxy      Reference     Transaction  Hot Row
```

```
Node          ID         Count          Count          Total
serv2         392        0              2              112
serv2         393        0              2              150
```

Examining the output from the **onstat** command in the previous example, there are two proxy distributors whose IDs are 392 and 393. The Transaction Count indicates the number of transactions currently being processed by each proxy distributor.

You run **onstat -g proxy** on a secondary server to view information about the proxy distributors that are able to service update requests from the secondary server.

```
onstat -g proxy

Primary    Proxy     Reference      Transaction  Hot Row
Node       ID        Count          Count        Total
serv1      392       0              2            112
serv1      393       0              2            150
```

In this example, the server is a shared disk (SD) secondary server, and is configured to update data. In addition, the server is connected to the primary server named **serv1**, and there are two proxy distributors, each with a transaction count of 2.

Use **onstat -g proxy all** on the primary server to display information about proxy distributors and *proxy agent threads*. One or more proxy agent threads are created by the proxy distributor to handle data updates from the secondary server.

```
onstat -g proxy all

Secondary  Proxy     Reference      Transaction  Hot Row
Node       ID        Count          Count        Total
serv2      392       0              2            1
serv2      393       0              2            0

TID   Flags       Proxy   Source  Proxy   Current  sqlerrno iserrno
                  ID      SessID  TxnID   Seq
63    0x00000024  392     22      1       5        0        0
64    0x00000024  392     19      2       5        0        0
62    0x00000024  393     23      1       5        0        0
65    0x00000024  393     21      2       5        0        0
```

In the output of the **onstat -g proxy all** command, **TID** represents the ID of the proxy agent thread that is running on the primary server. **Source SessID** represents the ID of the user's session on the secondary server. **Proxy TxnID** displays the sequence number of the current transaction. Each **Proxy TxnID** is unique to the proxy distributor. **Current Seq** represents the sequence number of the current operation in the transaction being processed. Each database transaction sent to a secondary server is separated internally into one or more operations that are then sent to the primary server. The last two fields, **sqlerrno** and **iserrno**, display any SQL or ISAM/RSAM errors encountered in the transaction. An error number of 0 indicates completion with no errors.

Running **onstat -g proxy all** on the secondary server displays information about all of the sessions that are currently able to update data on secondary servers.

```
onstat -g proxy all

Primary    Proxy     Reference      Transaction  Hot Row
Node       ID        Count          Count        Total
serv1      3466      0              0            1
serv1      3465      0              1            0
```

```
Session  Proxy   Proxy   Proxy   Current  Pending  Reference
         ID      TID     TxnID   Seq      Ops      Count
19       3465    67      1       23       0        1
```

In the output from the **onstat -g proxy all** command run on the secondary server,
Session represents the ID of a user's session on the secondary server. The **Proxy ID**
and **Proxy TID** are the same as those displayed on the primary server. **Pending
Ops** displays the number of operations that are waiting to be transferred to the
primary server. **Reference Count** displays the number of threads in use for the
transaction. When **Reference Count** displays 0 the transaction processing is
complete.

To display detailed information about the current work being performed by a
given distributor, use:

```
onstat -g proxy <proxy id> [proxy transaction id] [operation number]
```

The proxy transaction ID and operation number are both optional parameters.
When supplied, the first number is considered the proxy transaction ID. If a
secondary number follows it is interpreted as the operation number. If no proxy
transaction ID exists, the command performs the same as: **onstat -**. Similarly, if no
such operation number for the given proxy transaction ID exists, the command
performs the same as: **onstat -**.

Use the following command to display information about whether a server is
configured to allow updates to data. The command can be run either on the
primary or secondary server:

```
onstat -g <server_type>
```

Examples:

```
onstat -g rss
onstat -g sds
onstat -g dri
```

Configuring Connection Manager using settings from the primary server

It is often useful to configure a Connection Manager instance using the same
configuration parameters as those on the primary server. This is useful if you have
existing applications that point to a primary server and you do not want to
recompile those applications to point to Connection Manager. Using the procedure
that follows, configure a Connection Manager instance so that existing applications
connect to Connection Manager instead of the primary server.

For this example, assume that a Connection Manager instance has not been
configured, and you have a primary database server, an HDR secondary server, an
SD secondary server, and an RS secondary server. Assume further that the
DBSERVERNAME of the primary server is **myserv**, and the service name is **5656**.

To migrate all servers to the new DBSERVERNAME for the primary server:

1. Update the `sqlhosts` files on all of the secondary servers with the
   DBSERVERNAME and service name of the new primary server.

   Existing `sqlhosts` file:

   ```
   myserv <protocol> <host> 5656
   ```

   Modified `sqlhosts` file:

   ```
   myserv_pri <protocol> <host> 15656
   ```

2. On the Connection Manager server, edit the `sqlhosts` file as follows:

   ```
   myserv <protocol> <host> 5656
   ```

3. On the Connection Manager server, edit the `cmsm.cfg` file to set the name of the Connection Manager instance:

   ```
   NAME myserv
   SLA ...
   DEBUG ...
   LOGFILE ...
   ```

   Note that the Connection manager NAME and the corresponding `sqlhosts` file value are the same as those for the original primary server. This enables client applications to connect to Connection Manager without recompiling applications. If you are using a version of IBM Informix earlier than version 11.50, a driver upgrade might be required. See the *IBM Informix Migration Guide*.

4. Start Connection Manager: **oncmsm**

5. Shut down the SD secondary server and the primary server.

6. On the primary server, update the `onconfig` file to configure the new DBSERVERNAME:

   Old:

   ```
   DBSERVERNAME  myserv
   ```

   New:

   ```
   DBSERVERNAME  myserv_pri
   ```

7. Start the primary server using the following command: **oninit -SDS=myserv_pri**

8. Start the SD secondary server using the **oninit** command.

9. On the RS Secondary server, shut down the server and then start it in physical recovery mode: **oninit -PHY**

10. On the RS Secondary server, connect to new primary server with the following command: **onmode -d RSS myserv_pri**

For the HDR secondary server, no additional setup is required because the HDR pair are reestablished automatically. However, steps 6 and 7 can be performed in order to register the new primary server name with the HDR secondary server.

Design data replication group clients

This topic explains various design considerations for clients that connect to database servers that are running data replication.

Also see "Isolation levels on secondary servers" on page 22-35 for information about **committed read** and **committed read last committed** isolation levels on secondary servers.

Use of temporary dbspaces for sorting and temporary tables

Even though the secondary database server is in read-only mode, it does write when it must sort or create a temporary table. "Temporary dbspaces" on page 8-11 explains where the database server finds temporary space to use during a sort or for a temporary table.

To prevent the secondary database server from writing to a dbspace that is in logical-recovery mode, you must take the following actions:

1. Ensure that one or more temporary dbspaces exist. For instructions on creating a temporary dbspace, see "Creating a dbspace that uses the default page size" on page 9-7.
2. Perform one of the following actions:
   - Set the DBSPACETEMP parameter in the `onconfig` file of the secondary database server to the temporary dbspace or dbspaces.
   - Set the **DBSPACETEMP** environment variable of the client applications to the temporary dbspace or dbspaces.

Temporary tables created on secondary servers (SD secondary servers, RS secondary servers, and HDR secondary servers) must be created using the WITH NO LOG option. Alternatively, set the TEMPTAB_NOLOG configuration parameter to 1 on the secondary server to change the default logging mode for temporary tables to no logging. Tables created with logging enabled result in ISAM errors.

For SD secondary servers, set the SDS_TEMPDBS configuration parameter for configuring temporary dbspaces to be used by the SD secondary server.

For SD secondary servers, it is not necessary to explicitly add a temporary dbspace because the secondary server allocates the chunk specified by SDS_TEMPDBS when the server is started. It is only necessary to prepare the device that accepts the chunk.

If the primary server in a high-availability cluster fails and an SD secondary server takes over as the primary server, then the value set for the SDS_TEMPDBS configuration parameter on the SD secondary server is used for temporary dbspaces until the server is restarted. You must ensure that the value specified for the SDS_TEMPDBS configuration parameter on the SD secondary server is different than the value specified on the primary server. After the SD secondary server is restarted, the DBSPACETEMP configuration parameter is used.

Performing basic administration tasks

These topics contain instructions on how to perform database server administration tasks when your system is running HDR.

Changing database server configuration parameters

Some of the configuration parameters must be set to the same value on both database servers in the replication pair (as listed under "Database server configuration requirements for clusters" on page 21-3.) Other IBM Informix configuration parameters can be set to different values.

To make changes to `onconfig` files:
1. Bring each database server offline with the **onmode -k** option. If DRAUTO is set to RETAIN_TYPE or REVERSE_TYPE, you can more easily bring the secondary database server offline first.
2. Change the parameters on each database server.
3. Starting with the last database server that you brought offline, bring each database server back online.

   For example, if you brought the secondary database server offline last, bring the secondary database server online first. Table 21-1 on page 21-8 lists the procedures for bringing the primary and secondary database servers back online.

If the configuration parameter is not required to have the same value on each database server in the replication pair, you can change the value on the primary or secondary database server individually.

Back up storage spaces and logical-log files

When you use HDR, you must back up logical-log files and storage spaces only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.

You must use the same backup and restore tool on both database servers.

The block size and tape size used (for both storage-space backups and logical-log backups) must be identical on the primary and secondary database servers.

You can use **ontape** to set the tape size to 0 to automatically use the full physical capacity of a tape.

Changing the logging mode of databases

You cannot turn on transaction logging for databases on the primary database server while you are using HDR. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

To turn on database logging:
1. To turn HDR off, shut down the secondary database server.
2. Turn on database logging.

   After you turn on logging for a database, if you start data replication without performing the level-0 backup on the primary database server and restore on the secondary database server, the database on the primary and secondary database servers might have different data. This situation might cause data-replication problems.
3. Perform a level-0 backup on the primary database server and restore on the secondary database server. This procedure is described in "Starting HDR for the First Time" on page 21-6.

Add and drop chunks and storage spaces

You can perform disk-layout operations, such as adding or dropping chunks and dbspaces, only from the primary database server. The operation is replicated on the secondary database server. This arrangement ensures that the disk layout on both database servers in the replication pair remains consistent.

The directory path name or the actual file for chunks must exist before you create them. Make sure the path names (and offsets, if applicable) exist on the secondary database server before you create a chunk on the primary database server, or else the database server turns off data replication.

Renaming chunks

If you use symbolic links for chunk path names, you can rename chunks while HDR is operating. For instructions on renaming chunks, see the *IBM Informix Backup and Restore Guide*.

If you do not use symbolic links for chunk path names, you must take both database servers offline while renaming the chunks, for the time that it takes to complete a cold restore of the database server.

To rename chunks on a failed HDR server:

1. Change the mode of the undamaged server to standard.
2. Take a level-0 backup of the standard server.
3. Shut down the standard server.
4. Rename the chunks on the standard server during a cold restore from the new level-0 archive (for instructions, see the *IBM Informix Backup and Restore Guide*).
5. Start the standard server.
6. Take another level-0 archive of the standard server. Be sure the server is in standard mode.
7. Restore the failed server with the new level-0 backup and reestablish the HDR pair.

Saving chunk status on the secondary database server

For a data-replication pair, if the status of a chunk (*down*, *online*) is changed on the secondary database server, and that secondary server is restarted before a checkpoint is completed, the updated chunk status is not saved.

To ensure that the new chunk status is flushed to the reserved pages on the secondary database server, force a checkpoint on the primary database server and verify that a checkpoint also completes on the secondary database server. The new chunk status is now retained even if the secondary database server is restarted.

If the primary chunk on the secondary database server is down, you can recover the primary chunk from the mirror chunk.

To recover the primary chunk from the mirror chunk:

1. Run **onspaces -s** on the secondary database server to bring the primary chunk online.

   You also can use IBM Informix Server Administrator to bring the primary chunk online.
2. Run **onmode -c** on the primary database server to force a checkpoint.
3. Run **onmode -m** on the primary database server to verify that a checkpoint was actually performed.
4. Run **onmode -m** on the secondary database server to verify that a checkpoint was also completed on the secondary database server.

After you complete these steps, the primary chunk is online when you restart the secondary database server.

Use and change mirroring of chunks

Before you can add a mirror chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. If you want to mirror a dbspace on one of the database servers in the replication pair, you must create mirror chunks for that dbspace on both database servers. For general information about allocating disk space, see "Allocate disk space" on page 9-1.

Do not set the MIRROR configuration parameter to 1 unless you are using mirroring.

You can perform disk-layout operations from the primary database server only. Thus, you can add or drop a mirror chunk only from the primary database server. A mirror chunk that you add to or drop from the primary database server is added to or dropped from the secondary database server as well. You must perform mirror recovery for the newly added mirror chunk on the secondary database server. For more information, see "Recover a mirror chunk" on page 18-6.

When you drop a chunk from the primary database server, IBM Informix automatically drops the corresponding chunk on the secondary database server. This applies to both primary and mirror chunks.

When you turn mirroring off for a dbspace on the primary database server, Informix does not turn mirroring off for the corresponding dbspace on the secondary database server. To turn off mirroring for a dbspace on the secondary database server independent of the primary server, use **onspaces -r**. For more information about turning off mirroring, see "End mirroring" on page 18-6.

You can take down a mirror chunk or recover a mirror chunk on either the primary or secondary database server. These processes are transparent to HDR.

Manage the physical log

The size of the physical log must be the same on both database servers. If you change the size and location of the physical log on the primary database server, this change is replicated to the secondary database server. ONCONFIG values on secondary are updated automatically.

For information about changing the size and location of the physical log, see Chapter 16, "Manage the physical log," on page 16-1.

Manage the logical log

The size of the logical log must be the same on both database servers. You can add or drop a logical-log file with the **onparams** utility, as described in Chapter 14, "Manage logical-log files," on page 14-1. IBM Informix replicates this change on the secondary database server; however, the LOGFILES parameter on the secondary database server is not updated. After you issue the **onparams** command from the primary database server, therefore, you must manually change the LOGFILES parameter to the appropriate value on the secondary database server. Finally, for the change to take effect, you must perform a level-0 backup of the root dbspace on the primary database server.

If you add a logical-log file to the primary database server, this file is available for use and flagged F as soon as you perform the level-0 backup. The new logical-log file on the secondary database server is still flagged A. However, this condition does not prevent the secondary database server from writing to the file.

Manage virtual processors

The number of virtual processors has no effect on data replication. You can configure and tune each database server in the pair individually.

Manage shared memory

If you make changes to the shared-memory ONCONFIG parameters on one database server, you must make the same changes at the same time to the

shared-memory ONCONFIG parameters on the other database server. For the procedure for making this change, see "Changing database server configuration parameters" on page 22-22.

Set the wait time for a response from the primary server

You can use two environment variables, IFX_SMX_TIMEOUT and IFX_SMX_TIMEOUT_RETRY, to manipulate the amount of time that a high-availability replication (HDR), remote stand-alone (RS) or shared disk (SD) secondary server waits for a response from the primary server.

Use:
- The IFX_SMX_TIMEOUT environment variable to specify the maximum number of seconds for a secondary server to wait for a message from the primary server.
- The IFX_SMX_TIMEOUT_RETRY environment variable to specify the number of times that the secondary server repeats the wait cycle specified by the IFX_SMX_TIMEOUT environment variable if a response from the primary server has not been received.

Related reference:

➡ IFX_SMX_TIMEOUT environment variable (Guide to SQL: Reference)

➡ IFX_SMX_TIMEOUT_RETRY environment variable (Guide to SQL: Reference)

Replicate an index to an HDR secondary database server

If index page logging is enabled, index replication to the HDR secondary database server occurs automatically (see "Index page logging" on page 21-11). If index page logging is disabled, and an index on an HDR secondary database server becomes corrupted and must be rebuilt, you can either:
- Manually replicate the index from the primary server to the secondary server.
- Let the secondary server automatically replicate the index if you enabled the secondary server to do this.

To enable the secondary database server to automatically replicate the index, either:
- Set **onmode -d idxauto** to on.
- Set the value of the DRIDXAUTO configuration parameter to 1.

After you set either of these values, when one of the threads on the secondary database server detects a corrupted index, the index is automatically replicated to the secondary database server. Restarting index replication can take up to the amount of time specified in seconds in the DRTIMEOUT configuration parameter.

Sometimes, you might want to replicate an index manually, for example, when you want to postpone index repair because the table is locked. If you want to be able to manually replicate an index on the HDR secondary server, turn off the automatic replication feature.

To turn off the automatic index replication feature, either:
- Set **onmode -d idxauto** to off.
- Set the DRIDXAUTO configuration parameter to 0.

If **onmode -d idxauto** is set to off or DRIDXAUTO is set to 0 and the secondary server detects a corrupted index, you can manually replicate an index on the HDR secondary server by issuing an **onmode -d index** command in the following format.onmode -d index database:[ownername].table#index

For example:onmode -d index cash_db:user_dx.table_12#index_z

In the case of a fragmented index with one corrupted fragment, the **onmode -d idxauto** option only transfers the single affected fragment, whereas the **onmode -d index** option transfers the whole index.

Important: When turning the automatic index replication feature on or off, you can use either the **onmode** command or the DRIDXAUTO configuration parameter. If you use the **onmode** command, you are not required to stop and restart the database server. When you use the DRIDXAUTO parameter, the database server is restarted with the setting you specify. The **onmode** command does not change the DRIDXAUTO value. If you use the **onmode** command, you must manually change the value of DRIDXAUTO.

The online.log file produced by the secondary server contains information about any replicated index.

Related reference:

⬛➡ onmode -d command: Replicate an index with data-replication (Administrator's Reference)

⬛➡ DRIDXAUTO configuration parameter (Administrator's Reference)

Encrypting data traffic between HDR database servers

To support encrypted HDR connections in conjunction with Communication Support Module (CSM) client/server encryption, two network ports must be configured:

- One network port must be configured for HDR.
- The other network port must be configured for CSM client/server connections.

You can use Informix server encryption options to encrypt the data traffic between the database servers of an HDR pair. Do this when you want to ensure secure transmission of data.

After you enable encryption, the first database server in an HDR pair encrypts the data before sending the data to the other server in the pair. The server that receives the data, decrypts the data as soon as it receives the data.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the ENCRYPT_SMX configuration parameter on the secondary server. See "Enable SMX encryption" on page 21-12 for more information.

Restriction: You cannot start HDR on a network connection that is configured to use CSM encryption for client/server connections.

Additional buffers or larger buffers might be necessary to accommodate the size of encrypted data.

To encrypt data traffic between two HDR database servers:
1. Set the following configuration parameters on the first server in the HDR pair.
    - ENCRYPT_HDR, which enables or disables HDR encryption
    - ENCRYPT_CIPHERS, which specifies the ciphers and modes to use for encryption

- ENCRYPT_MAC, which controls the level of message authentication code (MAC) generation
- ENCRYPT_MACFILE, which specifies a list of the full path names of MAC key files
- ENCRYPT_SWITCH, which specifies the number of minutes between automatic renegotiations of ciphers and keys

To change these parameters, follow the instructions in "Changing database server configuration parameters" on page 22-22.

2. Set the encryption configuration parameters on the secondary server. The ENCRYPT_HDR, ENCRYPT_CIPHERS, ENCRYPT_MAC, and the ENCRYPT_SWITCH configuration parameters must have the same values as the corresponding configuration parameters on the primary server. The ENCRYPT_MACFILE configuration parameter can have a different value on each server, but the files must contain the same MAC keys.

For example, specify the following information about the primary and secondary servers in an HDR pair:

Configuration parameter	Sample setting on primary server	Sample setting on secondary server
ENCRYPT_HDR	1	1
ENCRYPT_CIPHERS	all	all
ENCRYPT_MAC	medium	medium
ENCRYPT_MACFILE	/vobs/tristan/sqldist/etc/mac1.dat	vobs/tristan/sqldist/etc/mac2.dat
ENCRYPT_SWITCH	60,60	60,60

In this example, the file name in the ENCRYPT_MACFILE path for the primary server is `mac1.dat` and the file name in the ENCRYPT_MACFILE path for the secondary server is `mac2.dat`. Otherwise, all settings are the same on both servers.

Only use these configuration parameters to specify encryption information for HDR. You cannot specify HDR encryption information by using the CSM option in the `sqlhosts` file.

HDR encryption works in conjunction with Enterprise Replication encryption and operates whether Enterprise Replication encryption is enabled or not. When working in conjunction with each other, HDR and Enterprise Replication share the same ENCRYPT_CIPHER, ENCRYPT_MAC, ENCRYPT_MACFILE and ENCRYPT_SWITCH configuration parameters.

For more information about these configuration parameters, see the *IBM Informix Administrator's Reference*.

Adjust LRU flushing and automatic tuning in HDR server pairs

When a server is configured for HDR, checkpoints triggered by the secondary database server are nonblocking. These types of checkpoints occur infrequently. If a nonblocking checkpoint is triggered by the secondary server, transactions are blocked on the primary server to make sure that the integrity of the secondary server is not compromised. If nonblocking checkpoints triggered by the secondary server occur on your system, you must tune LRU flushing more aggressively on the primary server to reduce transaction blocking.

To increase LRU flushing, reduce the values of **lru_min_dirty** and **lru_max_dirty** in the BUFFERPOOL configuration parameter.

Automatic LRU tuning can be turned on or off independently on each HDR node. The setting can be different on each HDR database server. For information about turning off automatic LRU tuning, see "Turn automatic LRU tuning on or off" on page 16-5.

For more information about LRU tuning, see the *IBM Informix Performance Guide*.

Cloning a primary server

You can use the **ifxclone** utility to perform one-step server instantiation, allowing a primary server in a high-availability cluster to be cloned with minimum setup or configuration.

You can use the **ifxclone** utility to create a stand-alone Informix server or to create an RS secondary server. Using the **ifxclone** utility, the database administrator can quickly, easily, and securely create a clone server from a running Informix instance without requiring to back up data on the source server, and transfer and restore it to the clone server. The backup and restore processes are started simultaneously using the **ifxclone** utility and there is no requirement to read or write data to disk or tape.

Data is transferred from the source server to the target server over the network using encrypted Server Multiplexer Group (SMX) Connections.

You can automate the creation of clone instances by calling the **ifxclone** utility from a script.

Creating a clone of a primary server

You use the **ifxclone** utility to create a clone of a primary server.

The general steps to create a clone of a server are as follows:
1. Set the following environment variables on the target server:
   - INFORMIXDIR
   - INFORMIXSERVER
   - ONCONFIG
   - INFORMIXSQLHOSTS
2. On the target server, create all of the chunks that exist on the source server. Follow these steps to create the chunks:
   a. On the source server, run the **onstat -d** command to display a list of chunks:
      ```
      onstat -d
      ```
   b. On the target server, log-in as user informix and use the commands **touch**, **chown**, and **chmod** to create the chunks. For example, to create a chunk named /usr/informix/chunks/rootdbs.chunk, follow these steps:
      ```
      $ su informix
      Password:
      $ touch /usr/informix/chunks/rootdbs.chunk
      $ chown informix:informix /usr/informix/chunks/rootdbs.chunk
      $ chmod 660 /usr/informix/chunks/rootdbs.chunk
      ```
   c. Repeat all of the commands in the previous step for each chunk reported by the **onstat -d** command.

3. While still logged in as user informix, run the **ifxclone** utility with the appropriate parameters on the target system on which the clone server is started.

4. Optionally, create **onconfig** and **sqlhosts** files on the target server.

Use the following steps to clone a server using the ONCONFIG and INFORMIXSQLHOSTS configuration files from the source server.

In this example, omitting the -L option causes the **ifxclone** utility to retrieve the necessary configuration information from the source server. The configuration files are used as a template to create the target server configuration. Having the **ifxclone** utility create the configuration files for you saves time and reduces the chance introducing errors in the configuration files.

For this example, assume that the source server (Amsterdam) has an **sqlhosts** file configured as follows:

```
#Server    Protocol HostName       Service Group
Amsterdam onsoctcp 192.168.0.1 123     -
```

You also must have the name, IP address, and port number of the target server. The following information is used for this example:

- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. On the target server, log-in as user informix and use the **touch**, **chown**, and **chmod** commands to create, change the owner, and change the permissions for the chunks. The chunk paths must match the paths of the chunks on the source server.

2. Run the **ifxclone** utility as user informix:

   ```
   ifxclone -T -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
           -i 192.168.0.2 -p 456
   ```

The **ifxclone** utility modifies the **sqlhosts** file on the source server and creates a copy of the file on the new target server. The **sqlhosts** file on the target server is the same as the source server:

```
#Server    Protocol HostName       Service Group
Amsterdam onsoctcp 192.168.0.1 123     -
Berlin    onsoctcp 192.168.0.2 456
```

Use the -L (–useLocal) option to create a clone of a server on a remote host computer: The -L option is used to merge the source onconfig file configuration information with the target onconfig file. This option also copies the source sqlhosts file to the target server.

- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. Create the **onconfig** and **sqlhosts** files and set the environment variables on the target computer.
2. On the target server, log-in as user informix and use the **touch**, **chown**, and **chmod** commands to create, change the owner, and change the permissions for the chunks. The chunk paths must match the paths of the chunks on the source server.
3. Run the **ifxclone** utility as user informix:

```
ifxclone -T -L -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
         -i 192.168.0.2 -p 456
```

To add an RS secondary server to the existing high-availability cluster:
- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. Create the **onconfig** and **sqlhosts** files and set the environment variables on the target computer.
2. On the target server, log-in as user informix and use the **touch**, **chown**, and **chmod** commands to create, change the owner, and change the permissions for the chunks. The chunk paths must match the paths of the chunks on the source server.
3. On the source server (if necessary), enable the LOG_INDEX_BUILDS configuration parameter enabled by running the following command as user informix:

```
onmode -wf LOG_INDEX_BUILDS=1
```

4. On the source server, run the following command as user informix to add the target server as an RS secondary server:

```
onmode -d add RSS Berlin
```

5. Run the **ifxclone** utility as user informix:

```
ifxclone -T -L -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
         -i 192.168.0.2 -p 456 -s medium -d RSS
```

On Windows systems, to clone a server instance using the useLocal (-L) option:
1. Manually create the instance on the target Windows system without initializing the server.
2. Use the **setnet32** utility to enter the source server entry in `sqlhosts` file.
3. Run the **ifxclone** utility.

    You must be a member of the Informix-Admin group to run the ifxclone command on Windows.

If you do not specify the UseLocal (-L) option you can run the **ifxclone** utility without first creating the instance on the target server. If the target server instance does not exist, you are prompted to enter the informix password to create the instance.

Related reference:

➥ The ifxclone utility (Administrator's Reference)

Database updates on secondary servers

You can enable applications connected to secondary servers to update database data. If you enable write operations on a secondary server, DELETE, INSERT, MERGE, and UPDATE operations are propagated to the primary server.

Use the UPDATABLE_SECONDARY configuration parameter to control whether the secondary server can update data and to configure the number of connections that update operations use.

Both data definition language (DDL) statements and data manipulation language (DML) statements are supported on secondary servers.

The **dbimport** utility is supported on all updatable secondary servers.

The **dbexport** is supported on a remote standalone (RS) secondary server only if:
• The STOP_APPLY configuration parameter is set to a valid value other than zero.
• The UPDATABLE_SECONDARY configuration parameter is set to a valid value other than zero.
• The USELASTCOMMITTED configuration parameter is set to COMMITTED READ, DIRTY READ, or ALL.

The USELASTCOMMITTED session environment setting can override the USELASTCOMMITTED configuration parameter setting.

The **dbschema** utility is supported on all updatable secondary servers.

The **dbschema** utility is also supported on read-only secondary servers. However, the **dbschema** utility displays a warning message when running on these servers.

Most applications that use DDL or DML can run on any of the secondary servers in a high-availability cluster; however, the following DDL statements are not supported:
• CREATE DATABASE (with no logging)
• CREATE EXTERNAL TABLE
• CREATE RAW TABLE
• CREATE TEMP TABLE (with logging)
• CREATE XADATASOURCE
• CREATE XADATASOURCE TYPE
• DROP XADATASOURCE
• DROP XADATASOURCE TYPE
• UPDATE STATISTICS

In cluster environments, the SET CONSTRAINTS, SET INDEXES, and SET TRIGGERS statements are not supported on updatable secondary servers. Any session-level index, trigger, or constraint modes that the SET Database Object Mode statement specifies are not redirected for UPDATE operations on table objects in databases of secondary servers.

Client applications can insert, update, and delete rows on a secondary server only if the secondary server image matches that of the primary server. The following data types are supported:

- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BYTE (stored in the table)
- CHAR
- CLOB
- DATE
- DECIMAL
- DATETIME
- FLOAT
- INT
- INT8
- INTERVAL
- MONEY
- NCHAR
- NVCHAR
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT (stored in the table)
- VARCHAR

BYTE and TEXT data types that are stored in blobspaces are not supported because blobspace data is not replicated.

The following data types are also supported if they do not receive a pointer reference to a different partition:

- COLLECTION
- LIST
- LVARCHAR
- MULTISET
- ROW
- SET
- UDTVAR

Any difference between the primary server image and the secondary server image causes an SQL error and the rollback of any changes.

You cannot use the following utilities on HDR secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers:

- **archecker**
- **dbload**
- High-Performance Loader (HPL)

- **ondblog**
- **onload**
- ON-Monitor
- **onparams**
- **onperf**
- **onsnmp**
- **onspaces**
- **onunload**

In addition, you cannot use the **dbexport** utility on HDR secondary servers or shared disk (SD) secondary servers. Among secondary servers, only remote standalone secondary (RS) servers support write operations by **dbexport**, and only with the previously-described STOP_APPLY, UPDATABLE_SECONDARY, and USELASTCOMMITTED configuration parameter settings.

SD secondary servers are not supported in Windows environments.

Byte range locking is not supported on secondary servers configured for updates. Byte range locks on secondary servers are promoted to full object locks.

Replicate smart large objects

You might receive one or more of the following error messages while working with updatable secondary servers:
- 12014
- 12015
- 12233

These errors generally indicate a problem with a smart large object file descriptor. These errors can be caused by any of the following conditions:
- A smart large object identifier is passed to another transaction or process before committing the transaction. Because all objects including smart large objects are uncommitted until the transaction is committed, do not allow other transactions to use the smart large object. In particular, dirty reads can access locked smart large objects.
- Smart large objects are not closed after opening them. At the end of a transaction, all smart large objects must be closed on secondary servers, especially those that are created and then the transaction is rolled back. Leaving smart large object file descriptors open causes memory to remain allocated until the session terminates.
- Another process has deleted the smart large object on the primary server. Share locks are not automatically propagated from secondary servers to the primary server so a different secondary server might access a smart large object that has actually been deleted on the primary. These accesses work until either the log record containing the delete is replayed on the secondary server or the secondary server is updated by the primary server.

Three additional error codes might be returned when processing dirty read information.
- -126 (ISAM error: bad row id)
- -244 (SQL error: Could not do a physical-order read to fetch next row)
- -937

Try your query again if you receive any of the previous codes.

LOCK TABLE statement behavior on secondary servers

You can set an exclusive lock on a table from an updatable secondary server in a high-availability cluster. For exclusive mode locks requested from a secondary server, sessions can read the table but not update it. This behavior is similar to shared access mode on a secondary server; that is, when one session has an exclusive lock on a given table, no other session can obtain a shared or exclusive lock on that table.

On read-only secondary servers, the session issuing the LOCK TABLE statement does not lock the table and the database server does not return an error to the client.

Shared mode locks in a cluster behave the same as for a standalone server. After a LOCK TABLE statement runs successfully, users can read the table but cannot modify it until the lock is released.

Related concepts:

"XA in high-availability clusters" on page 25-3

Isolation levels on secondary servers

The following statements are supported on all types of secondary servers:

```
Set isolation to committed read
Set isolation to committed read last committed
```

Secondary servers on which Committed Read isolation is set can read locally committed data. They can also read data committed on the primary server when it becomes available and committed on the secondary server. Applications connected to a secondary server receive data that is currently committed on the secondary server. See "Design data replication group clients" on page 22-21 for additional information about design considerations for clients that connect to database servers that are running data replication.

The default isolation level on secondary servers is Dirty Read; however, setting an explicit isolation level enables the correct isolation level: Dirty Read, Committed Read, or Committed Read Last Committed.

Repeatable Read and Cursor Stability isolation levels are not supported. Using the SET ISOLATION statement with Cursor Stability and Repeatable Read levels is ignored.

After starting a secondary server, client applications connect to the server only when all transactions open at the startup checkpoint have either committed or rolled back.

If the UPDATABLE_SECONDARY configuration parameter is disabled (by being unset or being set to zero), a secondary data replication server is read-only. In this case, only the DIRTY READ or READ UNCOMMITTED transaction isolation levels are available on secondary servers.

If the UPDATABLE_SECONDARY parameter is enabled (by setting it to a valid number of connections greater than zero), a secondary data replication server can support the COMMITTED READ, COMMITTED READ LAST COMMITTED, or COMMITTED READ transaction isolation level, or the USELASTCOMMITTED session environment variable. Only DML statements of SQL (the DELETE, INSERT,

UPDATE, and MERGE statements), and the **dbexport** utility, can support write operations on an updatable secondary server. (Besides UPDATABLE_SECONDARY, the STOP_APPLY and USELASTCOMMITTED configuration parameters must also be set to enable write operations by **dbexport** on a secondary data replication server.)

Use **onstat -g ses** or **onstat -g sql** to view isolation level settings. See the *IBM Informix Administrator's Reference* for more information.

Set lock mode: Issuing a SET LOCK MODE TO WAIT or SET LOCK MODE TO WAIT *n* statement on a secondary server sets the lock wait timeout value for that session just like on a primary server. The value set by SET LOCK MODE is used by the proxy thread created for the current session on the primary server when it performs updates from a secondary server. If the value for SET LOCK MODE is greater than the ONCONFIG parameter value of DEADLOCK_TIMEOUT, the value of DEADLOCK_TIMEOUT is used instead.

Transient types on high-availability cluster secondary servers

Transient unnamed complex data types (ROW, SET, LIST, and MULTISET) can be used on high-availability cluster secondary servers, whether the secondary servers are read-only or updatable. The following types of operations that use transient types are supported on secondary servers:

- SQL queries that use transient types
- SQL queries that use derived tables, collection subqueries, and XML functions (these statements implicitly use transient types)
- Temporary tables created with the CREATE TEMP statement that uses transient types

See the *IBM Informix Guide to SQL: Reference* and *IBM Informix Guide to SQL: Syntax* for information about complex data types.

Row versioning

Use row versioning to determine whether a row was changed and to detect collisions. With row versioning enabled, each row of a table is configured to contain both a checksum and a version number. When a row is first inserted, the checksum is generated automatically, and the version is set to 1. Every time the row is updated the version is incremented by one, while the checksum value is not changed. With row versioning, if a row is deleted and another row is reinserted in a table, it is possible to recognize that the row is different. By comparing the row checksum and row version between the secondary and the primary servers, it is possible to detect data collisions.

Web applications can use a version column to determine whether information contained in a previously retrieved object is still current. For example, a web application might display items for sale to a customer. When the customer decides to purchase an item, the application can check the version column of the item's row to determine whether any information about the item has changed.

If client applications can update data on the secondary servers in your environment, use row versioning to minimize network use, especially if your tables have many columns. Otherwise, entire rows on the secondary server are compared with entire rows on the primary server to determine whether updates occurred.

To add row versioning to an existing table, use the following syntax:

```
ALTER TABLE tablename add VERCOLS;
```

Similarly, you can delete row versioning from a table with the following syntax:

```
ALTER TABLE tablename drop VERCOLS;
```

To create a new table with row versioning, use the following syntax:

```
CREATE TABLE tablename (
    Column Name    Datatype
    Column Name    Datatype
    Column Name    Datatype
    ) with VERCOLS;
```

When row versioning is enabled, **ifx_row_version** is incremented by one each time the row is updated; however, row updates made by Enterprise Replication do not increment the row version. To update the row version on a server using Enterprise Replication, you must include the **ifx_row_version** column in the replicate participant definition.

Backup and restore with high-availability clusters

You cannot perform most backup and restore operations on secondary servers.

Before you can establish a server as an HDR (high-availability data replication) or RS (remote stand-alone) secondary server, you must perform a cold restore on it.

After you have set up a server as an HDR or RSS secondary server, you can only perform the following backup and restore operations:

- You can perform a logical restore on HDR or RS secondary servers when you are setting up a high-availability cluster.
- You can perform an external backup on an RS secondary server. For more information, see the *IBM Informix Backup and Restore Guide*.

SD secondary servers must be shut down during a cold restore of the primary server, but can be online during a warm restore, after they have been shut down and restarted.

Related tasks:

"Starting an RS secondary server for the first time" on page 21-12

"Starting HDR for the First Time" on page 21-6

"Recovering a shared-disk cluster after data is damaged" on page 24-19

Change the database server mode

If you change the mode of a database server in a high-availability cluster, replication stops.

To change the database server mode, use the **onmode** utility.

The following table summarizes the effects of changing the mode of the primary database server.

Table 22-3. Mode changes on the primary database server

On the primary	On the secondary	To restart HDR
Any mode to offline (**onmode -k**)	Secondary displays: DR: Receive error. HDR is turned off. The mode remains read-only. If DRAUTO is set to 0 (OFF), the mode remains read-only. If DRAUTO is set to 1 (RETAIN_TYPE), the secondary server switches to standard type and can accept updates. (If DRAUTO is set to 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.)	Treat it like a failure of the primary. Two different scenarios are possible, depending on what you do with the secondary database server while the primary database server is offline. See these sections for information: • The secondary server was not changed to the primary server • The secondary server was changed to the primary server automatically See "Restart if the primary server fails" on page 24-17.
to online, quiescent, or administration (**onmode -s** / **onmode -u**) (**onmode -j**)	Secondary does not receive errors. HDR remains on. Mode remains read-only.	Use **onmode -m** on the primary.

The following table summarizes the effects of changing the mode of the secondary database server.

Table 22-4. Mode changes on the secondary database server

On the secondary	On the primary	To restart HDR
Read-only offline (**onmode -k**)	Primary displays: DR: Receive error. HDR is turned off.	Treat it as you would a failure of the secondary. Follow the procedures in "Restarting HDR or RS clusters if the secondary server fails" on page 24-16.

Administration mode operates the same way on an HDR secondary database server as it does on the primary database server.

Changing the database server type

You can change the type of either the primary or the secondary database server.

You can change the type of either the primary or the secondary database server.

You can change the database server type from secondary to standard only if HDR is turned off on the secondary database server. HDR is turned off when the data replication connection to the primary database server breaks or data replication fails on the secondary database server. When you take the standard database server offline and bring it back online, it does not attempt to connect to the other database server in the replication pair.

Use the following commands to switch the type:

- hdrmksec.[sh|bat] and hdrmkpri.[sh|bat] scripts

To switch the database server type using hdrmkpri and hdrmksec scripts:

1. Shut down the primary database server (**ServerA**): **onmode -ky**
2. With the secondary database server (**ServerB**) online, run the hdrmkpri.sh script on UNIX or hdrmkpri.bat script on Windows.Now **ServerB** is a primary database server.
3. For **ServerA**, run the hdrmksec.sh script on UNIX or hdrmksec.bat script on Windows.Now **ServerA** is a secondary database server.
4. Bring **ServerB** (primary database server) online.

The following commands can also be used to switch the server type:

1. Change **ServerA** to the primary server by running the following command:

   onmode -d make primary ServerA

   This command makes **ServerA** the primary server, and redirects any other secondary servers in the cluster to point to the new primary server. The command also shuts down the old HDR primary (**ServerB**) because only a single primary server can exist in a high-availability environment.
2. Initialize **ServerB** as the HDR secondary server by running the following command:
   - On UNIX systems:

     $INFORMIXDIR/bin/hdrmksec.sh ServerB
   - On Windows systems:

     hdrmksec.bat ServerB

Related concepts:

"Manual switchover" on page 24-6

Prevent blocking checkpoints on HDR servers

On an HDR secondary server, checkpoint processing must wait until the flushing of buffer pools is complete. You can configure non-blocking checkpoints on an HDR secondary server so that log data sent from the primary server is stored, or *staged*, in a directory until checkpoint processing is complete.

You configure non-blocking checkpoints on an HDR secondary server by setting the LOG_STAGING_DIR and LOG_INDEX_BUILDS configuration parameters. When non-blocking checkpoints is configured, log data sent from the primary server is staged in a directory specified by the LOG_STAGING_DIR configuration parameter. When the HDR secondary server finishes processing the checkpoint it reads and applies the log data stored in the staging area. When the staging directory is empty the HDR secondary server reads and applies log data as it is received from the primary server.

You enable non-blocking checkpoints by setting the LOG_STAGING_DIR configuration parameter on the HDR secondary server, and LOG_INDEX_BUILDS on both the primary server and the HDR secondary server. The value for LOG_INDEX_BUILDS must be the same on both the primary server and the HDR secondary server.

When the HDR secondary server encounters a checkpoint, it enters a *buffering* mode. While in buffering mode, the secondary server stages any log page data from the primary server into files in the staging directory.

When the HDR secondary server completes checkpoint processing, the server enters a *drain* mode. In this mode, the HDR secondary server reads data from the staging file and also receives new data from the primary server. After the staging area is empty, the HDR secondary server resumes normal operation.

Where log records are stored on the HDR server

The HDR secondary server creates additional directories named `ifmxhdrstage_##` in the directory specified by LOG_STAGING_DIR, where `##` is the instance specified by SERVERNUM. The directories are used to store the logical logs sent from the primary server during checkpoint processing. The files within `ifmxhdrstage_##` are purged when no longer required.

Interaction of non-blocking checkpoints with secondary server updates

You must consider the interaction between secondary server updates and non-blocking checkpoints on HDR secondary servers. If the HDR secondary server receives an update request, the updates are not applied until the HDR secondary server processes the corresponding log records. When non-blocking checkpoints are enabled on the HDR secondary server, a delay in the application of data on the secondary server might occur because log data is staged at the secondary server due to checkpoint processing.

View statistics for nonblocking checkpoints on HDR servers

You use the **onstat** utility to view information about nonblocking checkpoints on primary servers and on HDR secondary servers.

To view information about staged logs, use the **onstat -g dri ckpt** command.

For an example of **onstat -g dri ckpt** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitor HDR status

Monitor the HDR status of a database server to determine the following information:
* The database server type (primary, secondary, or standard)
* The name of the other database server in the pair
* Whether HDR is on
* The values of the HDR parameters

Command-line utilities

The header information displayed every time you run **onstat** has a field to indicate if a database server is operating as a primary or secondary database server.

The following example shows a header for a database server that is the primary database server in a replication pair, and in online mode:

```
IBM Informix Dynamic Server Version 11.50.UC1   -- online(Prim) -- Up 45:08:57
```

This example shows a database server that is the secondary database server in a replication pair, and in read-only mode.

```
IBM Informix Dynamic Server Version 11.50.UC1     -- Read-Only (Sec) -- Up 45:08:57
```

The following example shows a header for a database server that is not involved in HDR. The type for this database server is standard.

```
IBM Informix Dynamic Server Version 11.50.UC1     -- online -- Up 20:10:57
```

The onstat -g dri option:
To obtain full HDR monitoring information, run the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The HDR state (on or off)
- The paired database server
- The last HDR checkpoint
- The values of the HDR configuration parameters

For an example of **onstat -g dri** output, see the *IBM Informix Administrator's Reference*.

The oncheck -pr option:
If your database server is running HDR, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that HDR uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in the following example.

```
Validating Informix Database Server reserved pages - PAGE_1ARCH &
 PAGE_2ARCH
         Using archive page PAGE_1ARCH.

    Archive Level                0
    Real Time Archive Began      01/11/95 16:54:07
    Time Stamp Archive Began     11913
    Logical Log Unique Id        3
    Logical Log Position         b018

    DR Ckpt Logical Log Id       3
    DR Ckpt Logical Log Pos      80018
    DR Last Logical Log Id       3
    DR Last Logical Log Page     128
```

SMI tables

The **sysdri** table provides information about the High-Availability Data-Replication status of the database server.

The **sysdri** table, described in these topics on the **sysmaster** database in the *IBM Informix Administrator's Reference*, contains the following columns.

type   HDR server type

state   HDR server state

name   Database server name

intvl   HDR buffer flush interval

timeout
      Network timeout

lostfound
      HDR lost-and-found path name

Monitor HDR status with ON-Monitor (UNIX)

Choose **Status** > **Replication** to see information about HDR.

This option displays the same information as the **onstat -g dri** option.

Obtain RS secondary server statistics

Use the **onstat** command to display information about the state of RS secondary servers. To display the number of RS secondary servers, information about the data that has been sent to the RS secondary servers, and information about what the RS secondary servers have acknowledged receiving, use the **onstat -g rss** command.

This command has options to show extended information about a single server or multiple secondary servers. For examples of **onstat -g rss** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Remove an RS secondary server

Remove an RS secondary server from a high-availability cluster by issuing the following command:

```
onmode –d delete RSS rss_servername
```

RS secondary server security

RS secondary servers support similar encryption rules as HDR. See "Encrypting data traffic between HDR database servers" on page 22-27 for details.

See "Server Multiplexer Group (SMX) connections" on page 21-12 for additional information about setting up and configuring encryption between servers and RS secondary servers.

Create or change a password on an RS secondary server

You can create a password for an RS secondary server to provide authentication between the primary server and the secondary server when the cluster is established. The password is optional. The password is valid only the first time the primary and secondary connect to each other.

The password prevents an unwanted instance on the network from initializing and accepting transaction data from the primary. The password is independent of the **informix** user password.

You create the password with the same command that you use to specify the RS secondary server name. The name and password of each RS secondary server can be defined either before or after the level-0 backup of the primary server.

To set the RS secondary server name and password, run the **onmode -d add RSS** *rss_servername password* command on the primary server. During secondary server setup, you include the password when you run the **onmode -d RSS** *rss_servername password* command on the secondary server.

You can change the password only if the server is not connected to a high availability environment. Attempting to change the password of an RS secondary server that is already connected returns an error.

To change the password on an RS secondary server before the server is connected, use the **onmode -d change RSS** *password* command on the primary server.

Examples

The following commands establish an RS secondary server named **ServerB** using a password of s#cure.

On the primary server:
```
onmode -d add RSS ServerB s#cure
```

On the secondary server:
```
onmode -d RSS ServerB s#cure
```

The following command changes the password of the RS secondary server named **ServerB** that is not connected to the primary to s@fest:
```
onmode -d change RSS ServerB s@fest
```

Transaction completion during cluster failover

You can configure servers in a high-availability cluster environment to continue processing transactions after failover of the primary server.

Transactions running on any server except the failed primary server continue to run. You configure the cluster environment so that:
- Transactions running on secondary servers are not affected.
- Transactions running on the secondary server that becomes the primary server are not affected.
- Transactions running on the failed primary server are terminated.

Transaction completion after failover is currently not supported for smart large objects, XA transactions, and when running DDL statements on secondary servers.

When a failover occurs, the secondary servers in the cluster temporarily suspend running user transactions until the new primary server is running. After failover, the secondary servers resend the saved transactions to the new primary server. The new primary server resumes execution of the transactions from the surviving secondary servers.

When running distributed transactions (transactions that span multiple database servers), any transaction running on the primary server at the time of failure are terminated.

Whether failover is automatic (using the Connection Manager) or manual (by specifying a server to become the new primary server), the failover server must be the server with the most advanced log replay position of all the surviving servers in the cluster. If the failover server does not have the most advanced log replay position, all transactions in the cluster are terminated and rolled back.

For best performance, use the default Connection Manager failover configuration of: SDS+HDR+RSS,0 (see Chapter 23, "Connection Management," on page 23-1).

Failover to an SD secondary server is recommended because the primary and SD secondary server read from the same physical disk.

If the failover server is an HDR secondary server, SD secondary servers are shut down.

Configuring the server so that transactions complete after failover

Use the FAILOVER_TX_TIMEOUT configuration parameter to configure the servers in a high-availability cluster so that transactions complete after failover.

The value of FAILOVER_TX_TIMEOUT indicates the maximum number of seconds the server waits before rolling back transactions after failure of the primary server. Set FAILOVER_TX_TIMEOUT to the same value on all servers in the cluster. For example, to specify 20 seconds for transaction completion, set the value of the FAILOVER_TX_TIMEOUT configuration parameter in the onconfig file to 20.

To disable transaction completion after failover, set the FAILOVER_TX_TIMEOUT configuration parameter to 0 on all servers in the cluster.

Related reference:

FAILOVER_TX_TIMEOUT configuration parameter (Administrator's Reference)

Chapter 23. Connection Management

Use the Connection Manager to monitor and maintain client connections, and to direct client connection requests to the appropriate server in a connection unit.

The **oncmsm** utility starts the Connection Manager, which manages and directs client connection requests based on service level agreements configured by the system administrator. The Connection Manager provides load balancing and directs client connection requests to one or more connection units. A connection unit is a collection of one or more database servers arranged in a grid, a high-availability cluster, a replicate set, or a server set configuration.

Table 23-1. Collection unit types and descriptions

Connection Unit Type	Description
CLUSTER	A high-availability cluster is a set of database servers consisting of a primary server and one or more secondary servers. The servers in a cluster are homogeneous; that is, all of the servers use the same hardware and software configurations. The data on the primary server is duplicated on all the secondary database server. Minimally, a cluster consists of a primary server and at least one secondary server. A cluster can consist of a primary server, an HDR secondary server, and zero or more shared disk secondary servers (SDS servers) and zero or more remote standalone secondary servers (RSS servers).
REPLSET	A replicate set is a set of database servers linked with Enterprise Replication (ER). ER supports asynchronous data replication over geographically distributed database servers. You can use ER to replicate both entire databases and subsets of databases and tables. Servers linked using ER can be heterogeneous; that is, the servers might use different hardware and software configurations. A replicate defines the replication participants and how to replicate data, while a replicate set combines several replicates to form a set that can be administered together as a unit. A domain is the set of all servers known to ER. A node within an ER domain can consist of a high-availability cluster, making it possible to have a cluster of clusters.
GRID	A grid is any set of interconnected servers, including clusters, replicate sets, and server sets, in an Enterprise Replication (ER) domain. A grid makes it easy to administer large groups of database servers. For example, when you create a table on one server the table is created on all of the servers within the Grid and the data is automatically synchronized.
SERVERSET	A server set is a server or group of servers that are managed by third-party replication application. The database server must have the same database name and schema in order for client applications to connect. The Connection Manager provides only availability and load balancing to server sets.

The Connection Manager is a daemon program that accepts a connection request from a client application and then connects the client to a database server. The Connection Manager gathers workload statistics from each server in the connection unit and directs client connections to the most appropriate server.

When the Connection Manager must choose among multiple servers to connect a client request, it decides which server to connect to based on a pre-configured policy determined by the system administrator. The specified policy directs connection requests based on server data latency, server failure status, or workload capacity.

The Connection Manager program is configured to use an `sqlhosts` file in the same manner as an Informix database server. You can configure more than one Connection Manager instance, which allows the failover from one Connection Manager instance to another if the Connection Manager fails. Configuring multiple Connection Manager instances is especially important to avoid having the Connection Manager become a single point of failure. See "Establish Connection Manager redundancy and failover" on page 23-25 the for an example of configuring multiple Connection Manager instances.

Because connection units might consist of multiple database servers arranged in grids, high-availability clusters, Enterprise Replication replicate sets, and server sets, client applications must have the ability to connect to any of a number of servers. With many servers, it can be difficult to determine which server to connect to. It can also be difficult to determine which server has sufficient free resources to perform a given task. Finally, it is difficult (if not impossible) to know when a server might encounter a problem. You use the Connection Manager to solve these difficulties.

The Connection Manager balances workloads by directing client application connections to the server with the least amount of activity. The Connection Manager utility also performs failover arbitration. You can configure the Connection Manager to ensure that, if a primary server in a high-availability cluster fails, another server automatically takes over the role of the primary server.

The Connection Manager performs three roles:
- Rule-based connection redirection
- Connection unit load balancing
- Cluster failover

Rule-based connection redirection

Applications connect to the Connection Manager as if they are connecting to a database server. When applications connect to the Connection Manager, redirection of the connection takes place in the communication layer and no additional action is required by the application.

In order to configure the Connection Manager, a daemon is started (or on Windows systems, a service) called **oncmsm** with customized redirection rules called Service Level Agreements (SLAs). After the Connection Manager is configured and initialized, it accepts connection requests from client applications and redirects them to the appropriate server based on the SLA (redirection rule).

Connection unit load balancing

The Connection Manager can perform load balancing, in which redirection is based on configurations set in the service level agreement. The Connection Manager connects to each of the servers in each connection unit and gathers statistics regarding the type of server, unused workload capacity, and the current state of the

server. From this information, the Connection Manager is able to redirect the client connection to the server with the highest available capacity.

You use the POLICY parameter in the SLA definition to set load balancing policy for connection units. For grids and replicate sets, quality of data (QOD) must be enabled in order to take advantage of the LATENCY, FAILURE, and WORKLOAD policies. If QOD is not set, or if the policy is not defined, the redirection policy is based on workload only.

For high-availability clusters and server sets, the policy is based on workload only.

Automatic failover

You can configure automatic failover with the Connection Manager. In a high-availability cluster, if the Connection Manager detects that the primary server has failed, and no action is taken by the primary server to reconnect during the ensuing timeout period, then the most appropriate secondary server is converted to the primary server.

You configure Connection Manager failover parameters using the FOC option in the Connection Manager configuration file. If you use multiple Connection Managers to manager failover for a cluster, you can enforce a consistent failover policy by setting the HA_FOC_ORDER configuration parameter on the cluster's primary server. The value of the HA_FOC_ORDER configuration parameter replaces the value of FOC ORDER= in the configuration file of each Connection manager that connects to the primary server.

The Connection Manager monitors the servers in connection units and it helps client applications connect to the most appropriate server. These roles, however, are separate; when the Connection Manager connects a client to a server, it does not redirect the client thereafter. If a database server to which an application is connected experiences a problem, the application must request a connection through the Connection Manager again.

The Connection Manager supports Distributed Relational Database Architecture (DRDA) connections. For more information, see "Overview of DRDA" on page 2-45 and "Configuring Informix for connections to IBM Data Server Clients" on page 2-46.

Related concepts:

"Redirection and connectivity for data-replication clients" on page 24-6

"Failover configuration" on page 24-1

"Overview of DRDA" on page 2-45

Related reference:

⇨ HA_FOC_ORDER configuration parameter (Administrator's Reference)

⇨ The oncmsm utility (Administrator's Reference)

"Service level agreement examples" on page 23-15

"Failover configuration examples" on page 23-21

"Connection information set in the HA_ALIAS configuration parameter" on page 2-44

Configuring the Connection Manager

To configure the Connection Manager, you must set up an encrypted password file, service level agreements, failover parameters, and configure the sqlhosts file.

Prerequisites:

UNIX only: Only user **informix** can run the **oncmsm** command. If user **root** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **root** or that member of the DBSA group can also run **oncmsm**.

Windows only: You must be a member of the **Informix-Admin** group to run the **oncmsm** command. If user **administrator** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **administrator** or that member of the DBSA group can also run the **oncmsm** command.

All operating systems: For proxy mode service level-agreements (SLAs) only, set the maximum number of file descriptors allowed by your operating system (for example, by using ulimit on UNIX systems).

For additional information about UNIX file descriptor parameters, see the *IBM Informix Performance Guide*.

The Connection Manager is packaged with the IBM Informix Client Software Development Kit (Client SDK) version 3.50 and higher. Before you configure and use the Connection Manager, you must install the Client SDK. See the *IBM Informix Client Products Installation Guide*.

To configure and start the Connection Manager:

1. Create and encrypt a password file (see "Creating an encrypted password file" on page 23-5).
2. Set the **INFORMIXDIR** environment variable (see "Configuring the environment for Connection Manager" on page 23-6).
3. Create a Connection Manager configuration file on the computer on which the Connection Manager is to run. The configuration file defines the Connection Manager instance name, the service level agreements, DEBUG parameter, the name of the log file, and the failover configuration. See "Connection Manager setup example" on page 23-23 for an example, or see *IBM Informix Administrator's Reference* for the configuration file layout.

4. Create entries for the primary server and all SLAs by editing the `sqlhosts` file on UNIX, or by editing the SQLHOST registry key with the **setnet32** utility on Windows (see "Modifying the sqlhosts file for Connection Manager" on page 23-6).

Related reference:

➽ The oncmsm utility (Administrator's Reference)

Creating an encrypted password file

If any of the servers within the connection unit are outside of a secure network environment, you must create an encrypted password file so that the Connection Manager can create secure connections to each of the servers in the cluster.

To create an encrypted password file:

1. Use a text editor to create an ASCII text file containing the server names, user names, and passwords for all servers in the cluster.

   For high-availability clusters and Enterprise Replication domains, if you are using a secure port by setting the s=6 parameter in the sqlhosts file and client applications use the DBSERVERALIASES configuration parameter to specify a list of alternative database server names, you must define the alternative server alias for the secure ports in the password file.

2. To protect the encrypted file, specify a key. The key can consist of any sequence of characters or numbers, but must not contain spaces.

   Important: To later decrypt the password file, you must supply the same key that was used to encrypt the file.

3. Encrypt the password file with the **onpassword** utility, specifying the key name and the password file name:

   ```
   onpassword -k key -e ./password_file
   ```

Related reference:

➽ The oncmsm utility (Administrator's Reference)

➽ The onpassword utility (Administrator's Reference)

Modifying an encrypted password file

You might want to modify the encrypted password file if you add or remove servers from a connection unit, change the passwords, or change your password key.

To edit the encrypted password file:

1. Decrypt the password file with the **onpassword** utility, specifying the password key and the password file name:

   ```
   onpassword -k EncryptKey -d ./password_file
   ```

2. If necessary, edit the file with a text editor to make the required changes.

   If necessary, determine a new password key. The password key can consist of any sequence of characters or numbers, but must not contain spaces. In order to later decrypt the password file, you must supply the same password key that was used to encrypt the file.

3. Encrypt the password file with the **onpassword** utility, specifying the password key and the password file name:

   ```
   onpassword -k EncryptKey -e ./password_file
   ```

Related reference:

↪ The onpassword utility (Administrator's Reference)

Configuring the environment for Connection Manager

Before starting the Connection Manager, the `INFORMIXDIR` environment variable must point to the directory in which the Connection Manager was installed.

If you are using the UNIX C shell (csh), use the **setenv** command to set the environment variables. For other shells, use the method appropriate for that shell.

```
setenv INFORMIXDIR path
```

To use a file other than `sqlhosts` to specify Connection Manager settings, set the `INFORMIXSQLHOSTS` environment variable to the name of that file.

For Windows systems, use the **Environment** tab of the **setnet32** utility to set or verify environment variables.

Modifying the sqlhosts file for Connection Manager

You must modify the Connection Manager `sqlhosts` file to define network connectivity information.

The Connection Manager is configured using the `sqlhosts` file in the same way that any server is configured. Each entry in the `sqlhosts` file represents either a service level agreement (SLA) name or a database server.

Every server that the Connection Manager is expected to connect to and monitor must be listed in the Connection Manager `sqlhosts` file. To monitor a high-availability cluster, the Connection Manager `sqlhosts` file must include the primary server and all secondary servers.

You can use the INFORMIXSQLHOSTS environment variable to specify where the database server can find the `sqlhosts` file.

1. Edit the `sqlhosts` file on the server on which Connection Manager is to run.
2. Add a line containing the name, net type, host name and service name of the primary server, and separate lines for each secondary server and each enterprise replication server that the Connection Manager is expected to manage.
3. Add one line for each of the service level agreement names. For example, the following entries create service level agreement names of **oltp** and **report**:

```
oltp      onsoctcp      cmhost1 cmport1
report    onsoctcp      cmhost1 cmport3
```

   In the example, **oltp** and **report** represent the name of Service Level Agreements (SLAs). The SLA name is what clients use to connect to a server through the Connection Manager. **cmhost1** represents the name of the server on which Connection Manager is running. **cmport1** and **cmport3** are the ports to which clients connect.

Related concepts:

"The sqlhosts file and the SQLHOSTS registry key" on page 2-17

Connection Manager configuration file

To set up a Connection Manager, you must create a Connection Manager configuration file.

Most of the Connection Manager options are specified in a configuration file. The configuration file consists of two parts:

- The heading, which contains Connection Manager configuration options that specify the instance name and other optional parameters.
- The body, which consists of one or more connection unit sections that define the type and name of the connection unit, service level agreements, failover configuration, and other options.

The general format of the configuration file is as follows:

```
# *** HEADING ***
NAME cm_name

# Optional configuration file parameters
MACRO name=server_list
LOG value
LOGFILE path_and_filename
DEBUG value
EVENT_TIMEOUT seconds
SECONDARY_EVENT_TIMEOUT seconds


# *** BODY ***
# Connection Unit 1
group_type connection_unit_name
{
    INFORMIXSERVER server_list
    SLA name attribute=value
    FOC ORDER=list|ENABLED|DISABLED TIMEOUT=value RETRY=value
    CMALARMPROGRAM path_and_filename
}
# Connection Unit 2
group_type connection_unit_name
{
    INFORMIXSERVER server_list
    SLA name attribute=value
    FOC ORDER=list|DISABLED TIMEOUT=value RETRY=value
    CMALARMPROGRAM path_and_filename
}
.
.
.
# Connection Unit n
group_type connection_unit_name
{
    INFORMIXSERVER server_list
    SLA name attribute=value
    FOC ORDER=list|ENABLED|DISABLED TIMEOUT=value RETRY=value
    CMALARMPROGRAM path_and_filename
}
```

Tip: For increased readability, you can break long configuration-file lines into smaller ones by using a backslash (\ ) as a line continuation character. The following example shows a macro definition that is separated into four lines:

```
MACRO  srv1list=node1,node2,node3,node4, \
                node6,node7,node8, \
                node100,node101, \
                node500,node501
```

Connection manager configuration file parameters

The following table lists the parameters that you can use in different sections of the Connection Manager configuration file. For more information about each parameter, follow the link to it from the table. The parameters are described in alphabetical order after the table.

Table 23-2. Mandatory and optional parameters for different parts of the Connection Manager configuration file

Part of the configuration file	Mandatory parameters	Optional parameters
Heading configuration file parameters	NAME	LOGLOGFILE DEBUG MACRO Connection unit type and name CM_TIMEOUT EVENT_TIMEOUT SECONDARY_EVENT_TIMEOUT
Body for a high-availability cluster	SLA INFORMIXSERVERFOC	CMALARMPROGRAM
Body for a grid, replicate set, or server set	SLA INFORMIXSERVER	CMALARMPROGRAM SQLHOSTS

CMALARMPROGRAM *path_and_file_name*

Specifies the path and file name of a program or script to run if failover processing encounters an error. If failover processing fails, then the Connection Manager calls the program that is specified by CMALARMPROGRAM.

CM_TIMEOUT *seconds*

Specifies the amount of time, in seconds, for the database server to wait for events from the Connection Manager. If the server does not receive any events from the Connection Manager within the specified amount of time, the server assumes that the Connection Manager is inoperative. If the inoperative Connection Manager is the failover arbitrator, then the next available Connection Manager becomes the failover arbitrator. If CM_TIMEOUT is not specified, the default timeout value is 60 seconds.

DEBUG *value*

Specifies whether to enable or disable debug mode. Set the DEBUG parameter to 1 to log SQL and ESQL/C error messages. Set the DEBUG parameter to 0 or remove the **DEBUG** parameter from the configuration file to disable debug mode. You cannot enable debug mode from the command line.

EVENT_TIMEOUT *seconds*

Specifies the number of seconds that the Connection Manager waits for primary-server events failover processing begins. An *event* in this context is an indication from the primary server that the server is still functioning, such as a performance statistics message, a node change message, or an administration message. If the Connection Manager does not receive any events from the primary server within the specified time, failover processing begins.

- -1 means that Connection Manager waits indefinitely for events from the primary server, or until it receives notification from one of the secondary servers that the primary server is offline.
- Values of 0 to 30 are read as 30.
- 60 is the default value.

FOC ORDER=*list* **| ENABLED | DISABLED TIMEOUT=***seconds* **RETRY=***value*
PRIORITY=*value*

Specifies the failover configuration (FOC) that the Connection Manager uses to determine which server takes over in case the primary server fails.

Important: If the FOC parameter is enabled and failover processing occurs, you must not restart the failed primary server manually until failover processing completes.

To configure multiple Connection Manager instances to manage a high-availability cluster, set the HA_FOC_ORDER configuration parameter in the primary server's `onconfig` file. If your Connection Manager instance manages more than one high-availability cluster and the failover configuration for a given cluster is identical within each Connection Manager instance, each cluster can have a different failover configuration.

The FOC parameter is not valid for connection unit type REPLSET, GRID, and SERVERSET.

If the FOC parameter is not specified, the following default value is used:

```
FOC ORDER=SDS,HDR,RSS TIMEOUT=0 RETRY=1 PRIORITY=0
```

Table 23-3. Attributes of the FOC parameter

Attribute	Description
ORDER	Specifies a comma-separated list of server types or whether the list of server types should come from the HA_FOC_ORDER configuration parameter of a cluster's primary server. • ENABLED specifies that the Connection Manager is considered in failover arbitration, using the primary server's HA_FOC_ORDER configuration parameter value to determine the failover node sequence. • DISABLED specifies that failover processing by the Connection Manager is disabled. If failover processing is disabled, the Connection Manager ignores failover requests. • HDR specifies a high-availability data replication secondary server. • RSS specifies a remote stand-alone secondary server • SDS specifies a shared disk secondary server.
TIMEOUT	Specifies the number of additional seconds to wait, after not receiving any events from the primary server, before the Connection Manager initiates the failover process. The TIMEOUT parameter applies after the EVENT_TIMEOUT value is exceeded. For example, if the EVENT_TIMEOUT parameter is set to **60** and the TIMEOUT value is set to **10**, 70 seconds of no primary server events must elapse before failover processing can begin.
RETRY	Specifies the number of times to go through the ORDER list while the Connection Manager waits for a successful failover to complete. After the RETRY limit is reached without a successful failover, an alarm is generated and the Connection Manager terminates the automated failover attempt. **0** indicates that the Connection Manager continues failover processing indefinitely.

INFORMIXSERVER *server_list*

Specifies a comma-separated list of servers to connect to during initialization of the Connection Manager. If the first server in the list in inoperative, then the Connection Manager attempts to connect to the next server in the list, and so on, until the end of the list is reached.

group_type connection_unit_name

A connection unit is a specific high-availability server configuration. You specify group types and connection units in your Connection Manager configuration file to direct client connection requests to the appropriate database server. Specify the *group_type* as one of the following values:

Chapter 23. Connection Management   **23-9**

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The *connection_unit_name* specifies the name by which the connection unit is known to the Connection Manager. Each name must be unique within the configuration file.

When specifying the name of a CLUSTER connection unit in the configuration file, you should use the same name as the group name for the corresponding cluster in the INFORMIXSQLHOSTS file.

When specifying connection unit names in the configuration file, you can use multibyte characters only for GRID and REPLSET connection unit types.

LOG *value*

Specifies logging for Connection Manager modes.
- 0 specifies no logging (default).
- 1 specifies that connection information about both PROXY and REDIRECT service level agreements is logged.
- 2 logs PROXY mode SLA information only and specifies that data send/receive activities between clients and the Connection Manager is logged.
- 3 logs PROXY mode SLA information only and specifies that data content between clients and the Connection Manager is logged.

LOGFILE *path_and_filename*

Specifies the name and location of the Connection Manager log file. The path and file name of the log file are displayed when the Connection Manager is started, and the log file is continuously updated with status information while the Connection Manager is running. The log file is necessary for monitoring and troubleshooting purposes.

Make sure that the directory for the log file exists, and that access to the file is enabled for the user that starts the Connection Manager. If no log file is specified, the Connection Manager creates a log file in the $INFORMIXDIR/tmp directory, with a name of *connection_manager_name.process_ID*.log.

MACRO *name*=(*server_list*)

Specifies the name of a macro and a list of server names. The macro becomes the list of server names when used in SLA definitions. For example, the following line defines a macro named CA that consists of three servers:

```
MACRO  CA=ca1,ca2,ca3
```

To use the macro, enclose the macro name in braces as shown in the following example:

```
SLA repl1_ca DBSERVERS=${CA}
```

Load balancing, where connections are directed based on server CPU usage, can be defined within a macro by enclosing the list of servers in parentheses:

```
MACRO  CA=(ca1,ca2,ca3)
```

NAME *cm_name*

Specifies the name of the Connection Manager instance. If the NAME parameter is not specified, the name of the first SLA is used as the Connection Manager instance name. Use *cm_name* to monitor the instance and to shut down or reload the Connection Manager. Connection Manager instance names must be unique.

SECONDARY_EVENT_TIMEOUT *seconds*

Specifies the amount of time, in seconds, that the Connection Manager waits for events from a secondary server. If the Connection Manager receives no events from a secondary server during the time that is specified, the Connection Manager disconnect from the secondary server. The default value is 60.

SLA *name*

Client applications use the SLA name to connect to the database servers that are specified by the value of DBSERVERS. The redirection policy specifies the type or the name of the target database server to which the client connects. For each SLA, a listener thread is installed at the specified port on the server to detect incoming client requests. The SLA parameter can be specified multiple times in the same configuration file; however, each SLA name must be unique. Attributes and attribute values are not case-sensitive.

There are optional attributes that can be set in a SLA definition:

- Each SLA should have a corresponding entry in the `sqlhosts` file; however, users can instead specify the `sqlhots` information inside the SLA by setting the `NETTYPE`, `HOST`, `SERVICE`, and `SQLHOSTSOPT` attributes. For example, the values in the following `sqlhosts` entry can be entered in an SLA:

```
#dbserver  nettype  hostname  servicename  options
oltp1      onsoctcp host_1    sales_1      csm=path
```

  The SLA entry:

```
SLA oltp1 DBSERVERS=primary \
   NETTYPE=onsoctcp \
   HOST=host_1 \
   SERVICE=sales_1 \
   SQLHOSTSOPT="cms=path"
```

- The `MODE` attribute specifies whether connection requests go through the Connection Manager or are redirected by the Connection Manager.

- The `POLICY` attribute specifies how the Connection Manager determines which listed database server is chosen to receive a connection request.

- The `USEALIASES` attribute specifies whether the Connection Manager can redirects client connection requests to database servers specified by the DBSERVERALIASES configuration parameter.

Table 23-4. Required attribute and values of the SLA parameter.

Required Attribute	Value
DBSERVERS	Specifies a comma-separated list of servers, server alias names, a server type, or a previously defined macro for directing connection requests. Enclose the list in parentheses to enable load balancing, where connections are directed to the server with the lowest CPU utilization. • `ANY` specifies that connection requests can be sent to any available database server. • `HDR` specifies that connection requests can be sent to the high-availability data replication server. • `PRI` or `primary` specifies that connection requests can be sent to the primary database server. • `SDS` specifies a that connection requests can be sent to shared disk secondary servers. • `RSS` specifies a that connection requests can be sent to remote stand-alone secondary servers. • A specific server name or alias specifies that connection requests can be sent to the database server with that name or alias. • ${*macro*} specifies that connection requests can be sent to the database servers defined in a macro in the heading of the configuration file.

Table 23-5. Optional Attributes and values of the SLA parameter.

Optional Attribute	Value
MODE	Specifies whether connection requests go through the Connection Manager or are redirected by the Connection Manager. • `PROXY` specifies that client connections use the Connection Manager as a proxy server. Use proxy mode when a client application cannot connect to a database server that is behind a firewall. Because a proxy-server Connection Manager handles all client requests, configuring multiple Connection Manager instances, to avoid the Connection Manager becoming a single point of failure. • `REDIRECT` specifies that client connections use redirect mode, which configures the Connection Manager to return the appropriate server node, IP address, and port number to the client application. The client application uses the IP address and port number that is returned by the Connection Manager to connect to a database server.
POLICY	Specifies how the Connection Manager directs client connection requests. The policy function uses a formula to decide which server to direct clients. The policy refers to a relative value rather than an absolute value. For example, specifying LATENCY does not indicate a specific latency period. • `LATENCY` specifies that the Connection Manager directs connection requests to the server with the lowest latency. • `FAILURE` specifies that connection requests are directed to the server with the fewest apply failures. • `WORKLOAD` specifies that connection requests are directed to the server with the lowest workload Policy values can be given relative weights. For example, to specify a policy for redirection to servers with the lowest latency and the fewest apply failures, with the LATENCY option specified as 10 times more important than the FAILURE option, use: `POLICY=10*LATENCY+FAILURE` To use the latency and failure policies, you must: • create a table through the grid, and the table must have replication enabled • enabled quality of data (qod) monitoring

Table 23-5. Optional Attributes and values of the SLA parameter. (continued)

Optional Attribute	Value

USEALIASES	Specifies whether the Connection Manager can redirects client connection requests to database server aliases specified by the DBSERVERALIASES configuration parameter.
	
	`OFF` specifies that the Connection Manager does not direct client connection requests to server aliases specified by the `DBSERVERS` attribute.
	
	For example, the database server srv1 has aliases `srv1_alias1` and `svr1_alias2`, and your Connection Manager contains the following SLA definitions:
	
	`SLA  sla1  DBSERVERS=srv1`
	`SLA  sla2  DBSERVERS=srv1          USEALIASES=OFF`
	`SLA  sla3  DBSERVERS=srv1_alias1`
	`SLA  sla4  DBSERVERS=srv1_alias1  USEALIASES=OFF`
	
	Given the previous configuration, the Connection Manager directs clients according to the following list:
	• Clients running connect to @sla1 can be directed to **srv1**, **srv1_alias1**, and **srv1_alias2**.
	• Clients running connect to @sla2 are directed to **srv1**.
	• Clients running connect to @sla3 can be directed to **srv1**, **srv1_alias1**, and **srv1_alias2**.
	• Clients running connect to @sla4 are directed to **srv1_alias1**.

WORKERS	Specifies the number of worker threads allocated to the SLA. When a service level agreement is specified, the Connection Manager creates an SLA listener process to intercept client connection requests. The SLA listener process can have one or more threads, called worker threads. The default is 4.

Table 23-6. Optional SQLHOSTS Attributes and values of the SLA parameter.

Optional SQLHOSTS Attribute	Value
HOST	Specifies a database server's host. The value in the SLA is used, rather than the value in the `sqlhosts` file. • A *hostname* value specifies a host name or host alias for a database server. • A *TCP/IP_address* value specifies a TCP/IP address for a database server.
NETTYPE	Specifies the network protocol of a database server. The value in the SLA is used, rather than the value in the `sqlhosts` file. • `onsoctcp` specify sockets with TCP/IP protocol • `drsoctcp` specifies Distributed Relational Database Architecture (DRDA) - connection for IBM data server client • `onsocssl` specifies Secured Sockets Layer (SSL) protocol for DRDA
SERVICE	Specifies a database server's port number or service name. The value in the SLA is used, rather than the value in the `sqlhosts` file. • A *port_number* value specifies a port number. • A *servicename* value specifies a service name.
SQLHOSTSOPT	Specifies connectivity options for a database server that is specified in a SLA. Enclose all connectivity options in a single pair of quotation marks. The value in the SLA is used, rather than the value in the `sqlhosts` file.

SQLHOSTS LOCAL|REMOTE|LOCAL+REMOTE

When the Connection Manager receives a client connection request, it searches for the requested server in the list of server instances that are defined in the Connection Manager's `sqlhosts` file. You can use the

SQLHOSTS parameter to cause the Connection Manager to search for database instances on servers other than the ones listed in the Connection Manager's sqlhosts file.

For example, you have a web server located on an external network and an application server located on an internal network. The database server defines TCP/IP **port1** and **port2** for external network connections and TCP/IP **port3** and **port4** for internal connections. Specifying **SQLHOSTS local** causes the Connection Manager to find server instances defined only in the sqlhosts file of the Connection Manager instance. In this case, the Connection Manager directs client connection requests only to servers on the external network.

Specifying **SQLHOSTS REMOTE** causes the Connection Manager to search for server instances that are defined in the sqlhosts file of the primary server instance. Specifying **SQLHOSTS LOCAL+REMOTE** causes the Connection Manager to direct connection requests to servers defined in the sqlhosts files of both the Connection Manager instance and the primary server instance.

The SQLHOSTS option is useful when you want to prevent users from accessing one or more servers in the cluster.

Table 23-7. Values for the SQLHOSTS Connection Manager configuration parameter

SQLHOSTS parameter value	Description
LOCAL	The Connection Manager searches for the requested instance in the sqlhosts file that is located on the computer on which the Connection Manager is running. The Connection Manager uses the value that is set by the INFORMIXSQLHOSTS environment variable to determine which sqlhosts file to use; Informix instances are specified by the DBSERVERNAME or DBSERVERALIASES configuration parameters.
REMOTE	The Connection Manager searches for the requested instance in the sqlhosts file that is located on a remote instance. The Connection Manager uses the value that is set by the **INFORMIXSQLHOSTS** environment variable to determine which sqlhosts file to use. Informix instances are specified by the DBSERVERNAME or DBSERVERALIASES configuration parameters.
LOCAL+REMOTE	The Connection Manager first checks the local sqlhosts file for Informix instances; if the requested server entry is not found then the DBSERVERNAME or DBSERVERALIASES configuration parameters on the remote server are used.

Related tasks:

→ Routing client connections in a grid (Enterprise Replication Guide)

Related reference:

"Determine the status of Connection Manager" on page 23-31

→ HA_FOC_ORDER configuration parameter (Administrator's Reference)

→ The oncmsm utility (Administrator's Reference)

→ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

→ DBSERVERALIASES configuration parameter (Administrator's Reference)

"sqlhosts Connectivity information" on page 2-21

→ cdr start qod (Enterprise Replication Guide)

→ cdr define qod (Enterprise Replication Guide)

Service level agreement examples

The examples in this section show how to configure Service Level Agreements (SLAs) for high-availability clusters, replicate sets, grids, and server sets.

In each of the following examples, it is possible to use specific server alias names or generic server types. For example, specifying the keyword RSS indicates a generic server type, which means that the Connection Manager evaluates each RS secondary server in the cluster to determine the best candidate for client application requests. You can also specify a server's network alias, such as sds1, or rss1.

The following configuration file examples show various options:

Example 1: A high-availability cluster with two service level agreements

```
NAME      cm1
LOG       1
LOGFILE   ${INFORMIXDIR}/tmp/cm1.log

CLUSTER west # cluster name
{
 INFORMIXSERVER  ids_w1,ids_w2
 SLA  oltp DBSERVERS=primary
 SLA  report DBSERVERS=HDR,SDS
 FOC  ORDER=ENABLED TIMEOUT=5 RETRY=2
 CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
```

This example configures the Connection Manager for a high-availability cluster and defines two SLAs.

The heading of the configuration file defines:
- **cm1** as the name of the Connection Manager instance
- Logging is enabled
- cm1.log as the name of the log file
- ${INFORMIXDIR}/tmp as the log directory

The body of the configuration file defines the two SLAs, failover parameters, and the program that is called if failover processing fails:

- Direct client `connect to @oltp` connection requests such to the primary server
- Direct client `connect to @report` connection requests to the HDR secondary server. If the HDR secondary server is not available, direct `connect to @report` connection requests to any available SD secondary server.
- Use the value of the primary server's HA_FOC_ORDER configuration parameter for the failover rule.
- The Connection Manager waits an additional 5 seconds for events with the connection unit's primary server before failover processing begins.
- The Connection Manager cycles through the list specified by the primary server's HA_FOC_ORDER configuration parameter 3 times before automated failover is terminated and an alarm is generated.
- If failover processing encounters an error, ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh is called.

Example 2: Connecting to SD secondary servers

```
NAME       cm2
LOG        1
LOGFILE    ${INFORMIXDIR}/tmp/cm2.log

CLUSTER east # cluster name
{
 INFORMIXSERVER  ids_e1,ids_e2
 SLA   secondaryNodes DBSERVERS=SDS,HDR,PRI
 FOC   ORDER=ENABLED TIMEOUT=5 RETRY=1
 CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
```

Connection Manager **cm2** directs client `connect to @secondaryNodes` connection requests as follows:

1. Connect to any available SD secondary servers.
2. If SD secondary servers are unavailable, connect to the HDR secondary server.
3. If the HDR secondary server is unavailable, connect to the primary server.

Example 3: Load balancing

To enable load balancing among servers, enclose the list of server types with parentheses.

```
NAME       cm3
LOG        1
LOGFILE    ${INFORMIXDIR}/tmp/cm3.log

CLUSTER south # cluster name
{
 INFORMIXSERVER  ids_s1,ids_s2
 SLA   secondary DBSERVERS=(SDS,HDR)
 FOC   ORDER=ENABLED TIMEOUT=5 RETRY=1
 CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
```

Connection Manager **cm3** directs client `connect to @secondary` connection requests to the SD or HDR secondary server with the lowest CPU utilization.

Example 4: PROXY and REDIRECT mode

In redirect mode, the Connection Manager sends the IP address and port number of the specified database server to the client application, which uses the information to connect to the database server. In proxy mode, client requests are routed through the Connection Manager. Redirect mode is the default if no SLA mode is specified.

```
NAME       cm4
LOG        1
LOGFILE    ${INFORMIXDIR}/tmp/cm4.log

CLUSTER north # cluster name
{
   INFORMIXSERVER  ids_n1,ids_n2
   SLA  sla1 DBSERVERS=alpha MODE=REDIRECT
   SLA  sla2 DBSERVERS=beta MODE=PROXY
   SLA  sla3 DBSERVERS=SDS MODE=REDIRECT
   FOC  ORDER=ENABLED TIMEOUT=5 RETRY=1
   CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
```

For this example:
- The primary server's alias name is **alpha**
- The HDR secondary server's alias is **beta**
- There are two SD secondary servers with aliases of **gamma1** and **gamma2**

Connection Manager **cm4** directs client connection requests as follows:
- `connect to @sla1` connection requests are directed to the primary server (**alpha**).
- `connect to @sla2` connection requests are directed through the Connection Manager, which acts as a proxy server, to the HDR secondary server (**beta**).
- `connect to @sla3` connection requests are directed to the SD secondary server which has the most available resources.

Example 5: Database server and Connection Manager timeout

```
NAME        cm5
LOG         1
LOGFILE     ${INFORMIXDIR}/tmp/cm5.log
CM_TIMEOUT 60
EVENT_TIMEOUT 45
SECONDARY_EVENT_TIMEOUT 50

CLUSTER southwest # cluster name
{
  INFORMIXSERVER  ids_sw1,ids_sw2
  SLA oltp DBSERVERS=primary MODE=PROXY
  SLA report DBSERVERS=SDS,RSS,HDR
  SLA primary DBSERVERS=primary
  SLA secondary DBSERVERS=(SDS,RSS,HDR)
  FOC ORDER=ENABLED RETRY=1
  CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
```

If a database server does not receive any events from the Connection Manager within 60 seconds, that database server assumes that the Connection Manager is inoperative, and the next available Connection Manager becomes failover arbiter.

If the Connection Manager does not receive any events from the primary server within 45 seconds, the Connection Manager begins failover processing. If the FOC parameter's TIMEOUT attribute is defined for a connection unit, the value of that parameter is added to the value of the EVENT_TIMEOUT parameter, for the total wait time before that connection unit's failover processing begins.

If the Connection Manager does not receive any events from a secondary server within 50 seconds, the Connection Manager disconnects from the secondary server.

- `connect to @oltp` connection requests are directed through the Connection Manager, which acts as a proxy server, to the primary server.
- `connect to @report` connection requests are directed to the first available SD secondary server. If SD secondary servers are unavailable, `connect to @report` connection requests are sent to any available RS secondary server. If RS secondary servers are unavailable, then `connect to @report` connection requests are sent to the HDR secondary server.
- `connect to @primary` connection requests are directed to the primary server.
- `connect to @secondary` connection requests are directed, by load balancing, to the secondary server that has the most resources available.
-

Example 6: Configuration for a replicate set

Macros can be defined for groups of servers. Load balancing can be defined within the macro, or within an SLA.

```
NAME       cm6
MACRO      NY=(ny1,ny2,ny3)
MACRO      CA=(ca1,ca2,ca3)
LOG        1
LOGFILE    ${INFORMIXDIR}/tmp/cm6.log

REPLSET erset #replset name
{
  INFORMIXSERVER    g_er1,g_er2
  SLA  repl1_any    DBSERVERS=ANY  # node discovery
  SLA  repl1_ca     DBSERVERS=${CA}
  SLA  repl1_ny     DBSERVERS=${NY}
}
```

In this example, two macros are defined:
- NY, which uses workload balancing and is composed of **ny1**, **ny2**, and **ny3**.
- CA, which is uses workload balancing and is composed of **ca1**, **ca2**, and **ca3**.

Connection Manager **cm6** directs client connection requests as follows:
- `connect to @repl1_any` connection requests are directed to any available server.
- `connect to @repl1_ca` connection requests are directed to whichever of **ny1**, **ny2**, or **ny3** has the lowest CPU utilization.
- `connect to @repl1_ny` connection requests are directed to whichever of **ca1**, **ca2**, or **ca3** has the lowest CPU utilization.

Example 7: Configuration for a grid that has a latency policy

```
NAME       cm7
MACRO      SF=(sf1,sf2,sf3)
MACRO      LA=(la1,la2,la3)
LOG        1
LOGFILE    ${INFORMIXDIR}/tmp/cm7.log

GRID grid1 # grid name
{
  INFORMIXSERVER  node1,node2,node3
  SLA  grid1_any    DBSERVERS=ANY POLICY=LATENCY
  SLA  grid1_avail  DBSERVERS=${SF},${LA}
}
```

In this example, two macros are defined:

- SF, which uses workload balancing and is composed of **sf1**, **sf2**, and **sf3**.
- LA, which is uses workload balancing and is composed of **la1**, **la2**, and **la3**.

Connection Manager **cm7** directs client connection requests as follows:

- connect to @grid1_any connection requests can be directed to any available server. Connection requests are directed to the server with the lowest latency.
- connect to @grid1_avail connection requests are directed to whichever of **sf1**, **sf2**, or **sf3** has the lowest CPU utilization. If **sf1**, **sf2**, and **sf3** are unavailable, connection requests are sent to whichever of **la1**, **la2**, or **la3** has the lowest CPU utilization.

Example 8: Configuration for a server set

```
NAME      cm8
LOG       1
LOGFILE   ${INFORMIXDIR}/tmp/cm8.log

SERVERSET ss # serverset name
{
  INFORMIXSERVER ids1,ids2,ids3
  SLA ssavail  DBSERVERS=ids1,ids2,ids3 \
     HOST=apollo \
     SERVICE=9600 \
     NETTYPE=onsoctcp
  SLA ssany  DBSERVERS=(ids1,ids2,ids3) \
     HOST=apollo \
     SERVICE=9610 \
     NETTYPE=onsoctcp
}
```

This example shows an example configuration to support a server set that is supported by the Connection Manager. The servers' sqlhosts values are defined in the SLA

- For the ssavail SLA, **ids1**, **ids2**, and **ids3** have the following sqlhosts information:

```
#dbservername    nettype       hostname      servicename    options
ids1             onsoctcp      apollo        9600
ids2             onsoctcp      apollo        9600
ids3             onsoctcp      apollo        9600
```

- For the ssany SLA, **ids1**, **ids2**, and **ids3** have the following sqlhosts information:

```
#dbservername    nettype       hostname      servicename    options
ids1             onsoctcp      apollo        9610
ids2             onsoctcp      apollo        9610
ids3             onsoctcp      apollo        9610
```

Connection Manager **cm8** directs client connection requests as follows:

- connect to @ssavail connection requests are directed to **ids1**. If **ids1** is unavailable, connect to @ssavail connection requests are sent to **ids2**. If **ids2** is unavailable, then connect to @ssavail connection requests are sent to **ids3**.
- connect to @ssany connection requests are directed to whichever of **ids1**, **ids2**, or **ids3** has the lowest CPU utilization.

Example 8: Configuration for multiple connection unit types

```
NAME      cm1
MACRO     NY=(ny1,ny2,ny3)
MACRO     CA=(ca1,ca2,ca3)
LOG       1
```

```
LOGFILE  ${INFORMIXDIR}/tmp/cm1.log

CLUSTER west
{
  INFORMIXSERVER ids_w1,ids_w2
  SLA oltpw DBSERVERS=primary
  SLA reportw DBSERVERS=(HDR,SDS)
  FOC ORDER=ENABLED TIMEOUT=5 RETRY=1
  CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
CLUSTER east
{
  INFORMIXSERVER ids_e1,ids_e2
  SLA oltpe DBSERVERS=primary
  SLA reporte DBSERVERS=RSS,HDR
  FOC ORDER=ENABLED TIMEOUT=5 RETRY=1
  CMALARMPROGRAM ${INFORMIXDIR}/etc/CMALARMPROGRAM.sh
}
REPLSET erset
{
  INFORMIXSERVER g_er1,g_er2
  SLA repl1_any  DBSERVERS=ANY # node discovery
  SLA repl1_ca DBSERVERS=${CA} POLICY=WORKLOAD
  SLA repl1_ny DBSERVERS=${NY}
}
GRID grid1
{
  INFORMIXSERVER node1,node2,node3
  SLA grid1_any DBSERVERS=ANY POLICY=LATENCY
  SLA grid1_avail DBSERVERS=${NY},${CA}
}
GRID grid2
{
  INFORMIXSERVER node4,node5
  SLA grid2_any DBSERVERS=ANY POLICY=LATENCY
  SLA grid2_avail DBSERVERS=${CA},${NY}
}
SERVERSET ss # name of serverset (unique in CM)
{
  INFORMIXSERVER ids1,ids2,ids3
  SLA ssavail DBSERVERS=ids1,ids2,ids3 \
     HOST=apollo SERVICE=9600 NETTYPE=onsoctcp
  SLA ssany  DBSERVERS=(ids1,ids2,ids3) \
     HOST=apollo SERVICE=9610 NETTYPE=onsoctcp
}
```

This example configures the Connection Manager to support two high-availability clusters, a replicate set, two grids, and one server set. Each connection-unit name (**west**, **east**, **erset**, **grid1**, **grid2**, and **ss**) must be unique in the Connection Manager configuration file.

Related concepts:

"Establish Connection Manager redundancy and failover" on page 23-25

"Connection Manager setup example" on page 23-23

Chapter 23, "Connection Management," on page 23-1

Related reference:

⯈ HA_FOC_ORDER configuration parameter (Administrator's Reference)

⯈ The oncmsm utility (Administrator's Reference)

⯈ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

Failover configuration examples

These examples show configuration examples for failover performed by a Connection Manager.

The following examples show several ways to configure Connection Manager to promote a secondary server to the primary server in the event the original primary server encounters a problem. Each service level agreement (SLA) should have a failover configuration defined.

If you use multiple Connection Managers to manager failover for a cluster, set the HA_FOC_ORDER configuration parameter on the cluster's primary server. The value of the HA_FOC_ORDER configuration parameter replaces the value of FOC ORDER= in the configuration file of each Connection manager that connects to the primary server.

For the following examples, the HA_FOC_ORDER configuration parameter on all servers is the default of SDS,HDR,RSS (shared-disk secondary server, high-availability data replication secondary server, and remote stand-alone secondary server).

Example 1: Immediate failover

```
FOC ORDER=ENABLED TIMEOUT=0
```

In this example, if the Connection Manager detects that the primary server is offline because the amount of time specified by the CM_TIMEOUT value has elapsed with no primary server events. Failover processing starts immediately because the TIMEOUT value is set to 0. The Connection Manager first attempts to convert the most suitable SDS server to the primary server. If no SDS server is online, Connection Manager attempts to convert the HDR server into the primary server. If the HDR server is not online, Connection manager attempts to convert the most suitable RSS server to the primary server.

Example 2: Failover with extended timeout

```
FOC ORDER=ENABLED TIMEOUT=10
```

In this example, setting the timeout value to 10 causes the Connection Manager to wait an addition 10 seconds for the primary server to come back on line before failover processing can begin. If the primary server does not come back online within 10 seconds, the Connection Manager first attempts to convert the most suitable SDS server to the primary server. If no SDS server is online, Connection Manager attempts to convert the HDR server into the primary server. If the HDR

server is not online, Connection manager attempts to convert the most suitable RSS server to the primary server.

Example 3: Failover with extended timeout and retry.

```
FOC ORDER=ENABLED TIMEOUT=20 RETRY=2
```

In this example, if the Connection Manager cannot determine within the amount of time specified by the CM_TIMEOUT parameter plus 20 seconds that the primary server is online, then failover processing begins. The Connection Manager first attempts to convert the most suitable SDS server to the primary server. If no SDS server is online, Connection Manager attempts to convert the HDR server into the primary server. If the HDR server is not online, Connection manager attempts to convert the most suitable RSS server to the primary server. The RETRY parameter indicates that the Connection Manager will go through the ORDER list of SDS,HDR,RSS a maximum of two times while waiting for a successful failover to complete. If failover does not complete after going through the ORDER list 2 times, an alarm is generated and the Connection Manager terminates failover.

Related concepts:

"Establish Connection Manager redundancy and failover" on page 23-25

"Connection Manager setup example" on page 23-23

Chapter 23, "Connection Management," on page 23-1

Related reference:

➥ HA_FOC_ORDER configuration parameter (Administrator's Reference)

➥ The oncmsm utility (Administrator's Reference)

➥ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

Proxy mode and redirect mode

Use the **MODE=REDIRECT** or **MODE=PROXY** after the SLA definition to specify how the Connection Manager redirects client connection requests.

Specifying **MODE=REDIRECT** configures the Connection Manager to return the appropriate server node, IP address, and port number to the client application (depending on the SLA definition). The client application connects to the server using the IP address and port number that the Connection Manager provides.
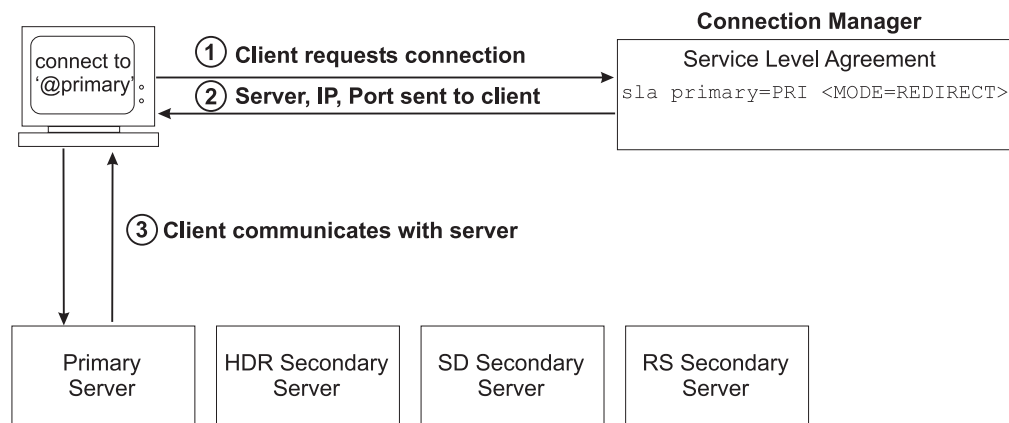
Figure 23-1. MODE=REDIRECT Service Level Agreement

Specifying **MODE=PROXY** configures the Connection Manager as a proxy server. Use **MODE=PROXY** when clients connect to Informix data servers from outside a firewall. Setting **MODE=PROXY** for a service level agreement causes the Connection Manager to manage client communication with the appropriate server. Use **MODE=PROXY** when the server is situated behind a firewall that the client would not ordinarily be able to connect to. Because the Connection Manager handles all client requests, consider configuring multiple Connection Manager instances when using **MODE=PROXY**.
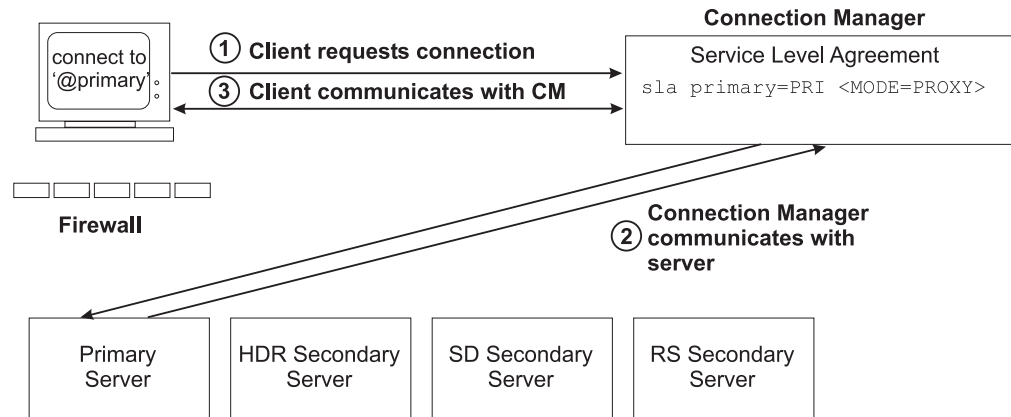
Figure 23-2. MODE=PROXY Service Level Agreement

Related reference:

➥ HA_FOC_ORDER configuration parameter (Administrator's Reference)

➥ The oncmsm utility (Administrator's Reference)

➥ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

Connection Manager setup example

This example shows how to set up the Connection Manager for a small high-availability cluster.

Suppose you had a configuration consisting of three servers: a primary server, an HDR secondary server, and an SD (shared disk) secondary server. You want the cluster to provide three different services: OnLine Transaction Processing (OLTP), PAYROLL, and REPORTING to support different applications. You decide that all OLTP services can be run on the primary server; PAYROLL services can be run on either the HDR secondary or the primary server; and REPORT services can be run on the HDR secondary or on the shared disk secondary servers.

The following figure illustrates the configuration. The illustration shows client applications issuing connection requests to the Connection Manager. Based on predefined service level agreements, the Connection Manager routes the connections to the appropriate server. Notice that the `sqlhosts` file for the Connection Manager and client computers defines a server group named **cluster_1**. The server group is configured to ensure that the Connection Manager can reconnect to the cluster in case of failure of the primary server and the Connection Manager is restarted. The server group must include all server nodes that are possible failover targets.
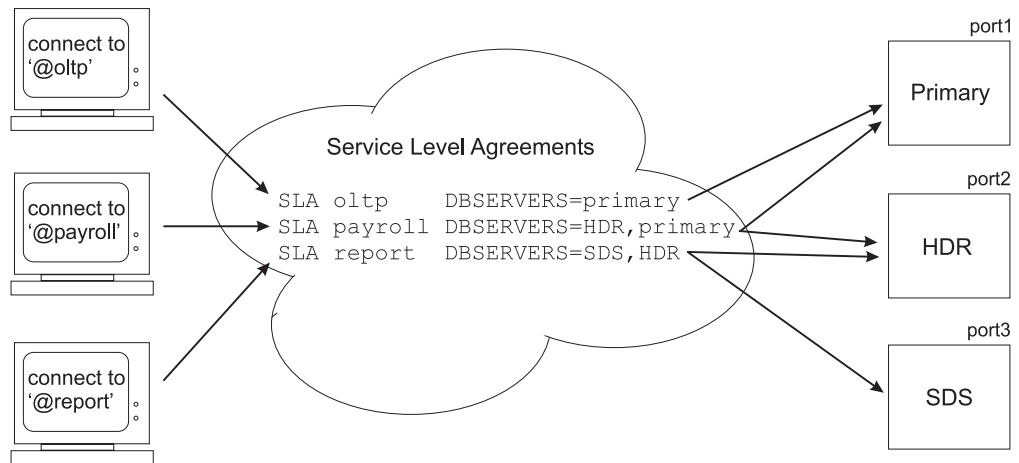
Figure 23-3. Connection Manager configuration

The `sqlhosts` file on the Connection Manager computer and the client computers is configured as follows:

```
#dbservername   nettype     hostname    servicename    options
cluster_1       group       -           -              i=10
ifx             onsoctcp    host1       port1          g=cluster_1
ifx_hdr         onsoctcp    host2       port2          g=cluster_1
ifx_sds         onsoctcp    host3       port3          g=cluster_1
oltp            onsoctcp    cmhost1     cmport1
report          onsoctcp    cmhost1     cmport2
payroll         onsoctcp    cmhost1     cmport3
```

The `sqlhosts` file on each server is configured as follows:

```
#dbservername   nettype     hostname    servicename    options
ifx             onsoctcp    host1       port1
ifx_hdr         onsoctcp    host2       port2
Ifx_sds         onsoctcp    host3       port3
```

To configure and start the Connection Manager:

1. Create and encrypt a password file. For this example, create a file called `passwords.txt` containing the following entries:

   ```
   ifx      ifx        informix password1
   ifx_hdr  ifx_hdr    informix password2
   ifx_sds  ifx_sds    informix password3
   ```

   Encrypt the file using the command:

   ```
   onpassword -k SecretKey -e ./passwords.txt
   ```

2. Set the **INFORMIXDIR** environment variable to point to the directory in which Client SDK was installed.

3. Create the Connection Manager configuration file. For this example, on the computer on which Connection Manager is to run, create a file called `cmconfig` that contains the following entries:

   ```
   NAME      cm_example
   LOG       1
   LOGFILE   cmlog

   CLUSTER cluster_1 # cluster name
   {
     INFORMIXSERVER ifx
     SLA  oltp    DBSERVERS=primary
   ```

```
    SLA  payroll DBSERVERS=HDR,primary
    SLA  report  DBSERVERS=SDS,HDR
    FOC  ORDER=ifx_sds,ifx_hdr TIMEOUT=10 RETRY=1
}
```

The configuration file names the Connection Manager instance **cm_example**, describes the service level agreements, turns debug off, defines the name of the log file, and defines a failover configuration.

4. Edit the `sqlhosts` file on the primary server, the HDR secondary server and the SD secondary server as follows:

```
#Server Protocol HostName Service
ifx     onsoctcp host1    port1
ifx_hdr onsoctcp host2    port2
ifx_sds onsoctcp host3    port3
```

5. On the Connection Manager computer, and on each of the client computers, edit the `sqlhosts` file as follows:

```
#dbservername    nettype     hostname    servicename    options
cluster_1        group       -           -              i=10
ifx              onsoctcp    host1       port1          g=cluster_1
ifx_hdr          onsoctcp    host2       port2          g=cluster_1
ifx_sds          onsoctcp    host3       port3          g=cluster_1
oltp             onsoctcp    cmhost1     cmport1
report           onsoctcp    cmhost1     cmport2
payroll          onsoctcp    cmhost1     cmport3
```

6. On the Connection Manager computer, start the Connection Manager by using the **oncmsm** command. For example:

```
oncmsm -c cmconfig
```

7. Examine the log file to verify that Connection Manager started correctly. A message indicates that the Connection Manager has started successfully.

Related tasks:

"Configuring Informix for connections to IBM Data Server Clients" on page 2-46

Related reference:

➡ CMCONFIG environment variable (Guide to SQL: Reference)

➡ The oncmsm utility (Administrator's Reference)

"Service level agreement examples" on page 23-15

"Failover configuration examples" on page 23-21

Establish Connection Manager redundancy and failover

You must configure multiple instances of the Connection Manager on different computers in order to avoid having the Connection Manager become a single point of failure.

You can install and run the Connection Manager on an IBM Informix server instance, or, to isolate the Connection Manager from database server failures, you can install it on a separate computer that is not running Informix. In addition, you can configure the Connection Manager to run on a hardware platform other than the computers that make up the connection unit. All that is required is the correct configuration of the `sqlhosts` file to indicate the connection host name and port number.

To further increase availability, you can configure multiple Connection Manager instances and use the failover feature of server groups within the `sqlhosts` file. Configuring multiple Connection Manager instances is especially important to avoid having the Connection Manager become a single point of failure. Using redirect mode, clients remain connected to servers after a failure of the Connection

Manager; however, new client connections cannot connect to servers unless another Connection Manager instance is configured to serve as a backup. Using proxy mode, the Connection Manager actively routes all client communications to and from the database servers. If the Connection Manager fails, all client connections are lost.

The following figure shows an example of a high-availability cluster that contains two Connection Manager instances.

Figure 23-4. Connection Manager group configuration

The HA_FOC_ORDER configuration parameter on the primary server is set to SDS,HDR,RSS.

The sqlhosts file on the primary and secondary servers is configured as follows:

```
#dbservername   nettype    hostname      servicename   options
ifx_cluster     group      -             -             i=30
ifx_pri         onsoctcp   p_machine     ifxsrv0pri    g=ifx_cluster
ifx_sds         onsoctcp   sds_machine   ifxsrv0sds    g=ifx_cluster
ifx_hdr         onsoctcp   hdr_machine   ifxsrv0hdr    g=ifx_cluster
ifx_rss         onsoctcp   rss_machine   ifxsrv0rss    g=ifx_cluster
```

The sqlhosts file configures a server group named **ifx_cluster** that includes each server in the cluster.

The sqlhosts file on the servers running the Connection Manager and on the client computers is configured as follows:

```
#dbservername       nettype    hostname      servicename   options
ifx_pri             onsoctcp   p_machine     ifxsrv0pri
# ifx_cm group
ifx_cm              group      -             -             c=1
ifx_cm1             onsoctcp   cm1_machine   cm1_port1     g=ifx_cm
ifx_cm2             onsoctcp   cm2_machine   cm2_port2     g=ifx_cm
# ifx_proxy group
```

```
ifx_proxy       group    -            -           c=1
ifx_proxy1      onsoctcp cm1_machine  cm1_proxy1  g=ifx_proxy
ifx_proxy2      onsoctcp cm2_machine  cm2_proxy2  g=ifx_proxy
```

Each entry (each line) in the sqlhosts file associates a name with a database server, a server group, or a service level agreement. In the example, **ifx_pri** is the name assigned to the primary server, so client connection requests sent to **ifx_pri** are directed to the primary server.

The **ifx_cm** entry defines a group that consists of two service level agreements, **ifx_cm1** and **ifx_cm2**. The **c=1** option causes the client application to choose a random starting point from which to connect to the list of servers or service level agreements. Clients connection requests sent to **ifx_cm** are directed either to **ifx_cm1** or to **ifx_cm2**. If one of the Connection Manager instances is offline, client requests are automatically routed to the online instance.

The **ifx_proxy** entry also defines a group that consists of two service level agreements, **ifx_proxy1** and **ifx_proxy2**. Clients connection requests sent to **ifx_proxy** are directed either to **ifx_proxy1** or to **ifx_proxy2**. If one of the Connection Manager instances is offline, client requests are automatically routed to the online instance.

Suppose that a client connection request sent to **ifx_cm** was directed to **ifx_cm1** either because **ifx_cm2** was offline or because it was chosen randomly. In this case, the sqlhosts file directs the request to the Connection Manager instance running on **cm1_machine**. Similarly, requests to **ifx_cm2** are directed to the Connection Manager instance running on **cm2_machine**. In the same manner, connection requests sent to **ifx_proxy** are sent to the Connection Manager instance on either **cm1_machine** or **cm2_machine**.

Because **ifx_cm1**, **ifx_cm2**, **ifx_proxy1**, and **ifx_proxy2** are service level agreements rather than database servers, client requests are directed according to the service level agreement's redirection policy defined in the Connection Manager configuration file.

The configuration file for the first Connection Manager instance, **cm1_machine**, is shown with line numbers in the following table. The line numbers are not included in the configuration file.

Table 23-8. Connection Manager configuration file 1

Line	Configuration file
1	NAME ifx_cm1
2	LOG 1
3	LOGFILE /usr2/logs/cmsm1.log
4	CLUSTER my_cluster1
5	{
6	INFORMIXSERVER=ifx_pri,ifx_sds_ifx_rss,ifx_hdr
7	SLA ifx_cm1 DBSERVERS=(SDS,HDR),RSS,primary
8	SLA ifx_proxy1 DBSERVERS=HDR,RSS,primary MODE=PROXY
9	FOC ORDER=ENABLED TIMEOUT=5
10	}

Line 1 Sets the name of the Connection Manager instance to ifx_cm1.

Line 2 Specifies that Connection Manager logging is enabled. Set LOG 1 to log information about both PROXY and REDIRECT service level agreements.

Line 3 Specifies the name and location of the Connection Manager log file.

Line 4 Specifies that the connection unit is a high-availability cluster and that its name is my_cluster1.

Line 5 The left brace indicates the start of the body of the connection unit definition.

Line 6 Specifies one or more Informix servers to connect with during initialization of the Connection Manager.

Line 7 Creates a service level agreement that directs client connection requests sent to **ifx_cm1** to the SD secondary server or to the HDR secondary server, depending on which server has the most available resources. If no SD secondary servers are available and the HDR secondary server is not available, then the request is directed to the RS secondary server. If the RS secondary server is not available, requests are routed to the primary server.

Line 8 Creates an SLA that configures client connections to use proxy mode. Client connection requests sent to **ifx_proxy1** are directed to the HDR secondary server. If the HDR secondary server is not available, then requests are directed to the RS secondary server. If more than one RS secondary servers are included in the cluster, then client requests are directed to the RS secondary server with the most available resources. If no RS secondary servers are available, then client requests are routed to the primary server.

Using proxy mode, the Connection Manager serves as a proxy server. Clients can use proxy mode to connect to data servers that are protected by a firewall. If your database servers are protected by a firewall, configure the Connection Manager computers so clients can access them without going through the firewall. Configure the Connection Manager computers with the appropriate security settings to communicate through the firewall to the database servers.

Line 9 Specifies that the value of the HA_FOC_ORDER configuration parameter on the primary server is used for failover order configuration, and specifies a timeout value that the Connection Manager uses to promote a secondary server to the primary server in the event the primary server fails. With this failover configuration, the first available SD secondary server takes over the role of the primary server if the primary server fails. If no SD secondary server is available, the HDR secondary server takes over the role of the primary server. If no SD of HDR secondary servers are available, the RS secondary server takes over the role of primary server. The timeout value specifies the amount of time (in seconds) to wait before performing a failover; in this case, five seconds.

Line 10

The right brace indicates the end of the body of the connection unit definition.

The configuration file for the second Connection Manager instance, **cm2_machine**, is shown with line numbers in the following table. The line numbers are not included in the configuration file.

Table 23-9. Connection Manager configuration file 2

Line	Configuration file
1	`NAME     ifx_cm2`
2	`LOG      1`
3	`LOGFILE  /usr2/logs/cmsm2.log`
4	`CLUSTER  my_cluster2`
5	`{`
6	`  INFORMIXSERVER=ifx_pri,ifx_sds_ifx_rss,ifx_hdr`
7	`  SLA ifx_cm2 DBSERVERS=(SDS,HDR),RSS,primary`
8	`  SLA ifx_proxy2 DBSERVERS=HDR,RSS,primary MODE=PROXY`
9	`  FOC ORDER=ENABLED TIMEOUT=5`
	`}`

The configuration files are identical except for the SLA names, and the name of the cluster and the log file. The `sqlhosts` file ensures that requests sent to **ifx_cm** and **ifx_proxy** are directed to the correct Connection Manager instance. It is not strictly necessary to use different log file names if the Connection Manager instances exist on different computers as in this example. Use different log file names or directories, however, if the Connection Manager instances exist on the same computer.

- If the Connection Manager configuration file is located in $INFORMIXDIR/etc and is named cm1_machine.cfg, run the following command on **cm1_machine** to start the Connection Manager instance:

  `oncmsm -c $INFORMIXDIR/etc/cm1_machine.cfg`

- If the Connection Manager configuration file is located in $INFORMIXDIR/etc and is named cm2_machine.cfg, run the following command on **cm2_machine** to start the Connection Manager instance:

  `oncmsm -c $INFORMIXDIR/etc/cm2_machine.cfg`

Client applications must send connection requests to the Connection Manager group name, **ifx_cm** or **ifx_proxy**. Using the group name ensures that clients connect to a valid Connection Manager instance even if one of the Connection Manager instances is offline.

When configuring multiple Connection Manager instances, ensure that the service level agreements defined for each Connection Manager instance are identical. Having identical service level agreements ensures that clients use the same redirection policies regardless of which Connection Manager instance is used.

For more information about creating and configuring groups, see "Client/server architecture" on page 2-1.

Related reference:

➡ HA_FOC_ORDER configuration parameter (Administrator's Reference)

➡ The oncmsm utility (Administrator's Reference)

"Service level agreement examples" on page 23-15

"Failover configuration examples" on page 23-21

Starting the Connection Manager

You use the **oncmsm** utility to start the Connection Manager.

To start the Connection Manager:

- UNIX

  Start the Connection Manager on UNIX systems with the following command:

  ```
  oncmsm -c configuration_file
  ```

- Windows

  Starting the Connection Manager on Windows systems is a two-step process. You install the service first then start the Connection Manager. To install the Connection Manager service:

  ```
  oncmsm -i -c configuration_file
  ```

  To start the Connection Manager:

  ```
  oncmsm Connection_Manager_Name
  ```

  Connection_Manager_Name is the name of the Connection Manager instance.

  The following message is displayed when starting the Connection Manager on Windows systems:

  ```
  Specify the user and password to run this service.
  Press <ENTER> to run Connection Manager as 'localsystem'
  ```

  Either enter the **informix** user ID and password, or press **Enter** to automatically create an **informix-admin** group and assign it the necessary access rights.

Related reference:

➡ The oncmsm utility (Administrator's Reference)

Monitoring the Connection Manager

Tools are available to help diagnose problems that arise using the Connection Manager.

The Connection Manager log file contains information about service level agreements, failover configuration, and status information. The location of the log file is displayed when the Connection Manager is started.

In addition to examining the log file, you can use tools provided with the server. You can monitor the status of the Connection Manager in the following ways:

- Use the **onstat** command to display statistics that you can use to diagnose problems.
- Use the IBM OpenAdmin Tool (OAT) for Informix.

Determine the status of Connection Manager

You can use the **onstat** utility to display information about active Connection
Manager instances.

Use the **onstat -g cmsm** command to display the Connection Manager daemons
that are attached to a server instance, and also to display the number of
connections that the daemon has processed. See the *IBM Informix Administrator's
Reference* for more information and examples.

The following code is example output from the **onstat -g cmsm** command with a
single active Connection Manager instance.

```
Connection Manager Name: argo
Hostname: argo

SLA       Connections  Service/Protocol  Rule
oltp      9            9593/onsoctcp     primary
oltp_ssl  4            9596/onsocssl     primary MODE=PROXY
report    0            9594/onsoctcp     SDS,HDR,RSS

Failover Configuration:

Connection Manager name  Rule         Timeout  State
argo                     SDS,HDR,RSS  10       Active Arbitrator, Primary is up
```

The following code is example output from the **onstat -g cmsm** command with
two active Connection Manager instances.

```
Connection Manager Name: reynolds
Hostname: argo

SLA       Connections  Service/Protocol  Rule
oltp      9            9593/onsoctcp     primary
oltp_ssl  4            9596/onsocssl     primary mode=proxy
report    0            9594/onsoctcp     SDS,HDR,RSS

Connection Manager Name: stimpson
Hostname: argo

SLA       Connections  Service/Protocol  Rule
oltp2     9            19903/onsoctcp    primary
report2   0            19904/onsoctcp    SDS,HDR,RSS

Failover Configuration:
Connection Manager name  Rule         Timeout  State
argo
                         SDS,HDR,RSS  10       Active Arbitrator, Primary is up
argo2
                         SDS,HDR,RSS  0        Primary is up
```

Related reference:

⇨ The oncmsm utility (Administrator's Reference)

"Connection Manager configuration file" on page 23-6

⇨ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

Stop the Connection Manager

You stop a Connection Manager instance when you no longer want connection redirection to occur.

Prerequisites:

UNIX only: Only user **informix** can run the **oncmsm** command. If user **root** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **root** or that member of the DBSA group can also run **oncmsm**.

Windows only: You must be a member of the **Informix-Admin** group to run the **oncmsm** command. If user **administrator** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **administrator** or that member of the DBSA group can also run the **oncmsm** command.

To stop a Connection Manager instance:
1. Log on to the computer on which the Connection Manager instance is running.
2. Use the **oncmsm** utility with the **-k** option.

   ```
   oncmsm -k connection_manager_name
   ```
Related reference:

⇨ The oncmsm utility (Administrator's Reference)

Dynamically reconfiguring the Connection Manager

You use the Connection Manager reload option to add or change service level agreements, failover parameters, log file name, debug, or other options.

Prerequisites:

UNIX only: Only user **informix** can run the **oncmsm** command. If user **root** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **root** or that member of the DBSA group can also run **oncmsm**.

Windows only: You must be a member of the **Informix-Admin** group to run the **oncmsm** command. If user **administrator** or a member of the DBSA group is granted privileges to connect to the sysadmin database, the user **administrator** or that member of the DBSA group can also run the **oncmsm** command.

You can edit the existing configuration file or replace the original file with one that contains the new parameters. The new options take effect immediately.

To reload configuration options, use the **-r** parameter:

```
oncmsm -r connection_manager_name
```

The *connection_manager_name* is the name of the Connection Manager instance to reload. Because multiple instances of the Connection Manager can be active at the same time, it is necessary to specify the name of the Connection Manager instance.

Tip: Use the **onstat** utility to determine the names of Connection Manager instances: **onstat -g cmsm**

For example, to reload the configuration file of a Connection Manager instance named **cm_1**:

1. Modify the existing Connection Manager configuration file with the new service level agreements, failover parameters, log file name, or debug option.
2. On the computer on which Connection Manager is running: **oncmsm -r cm_1**

   The Connection Manager loads the new parameters.

Related reference:

➡ The oncmsm utility (Administrator's Reference)

➡ onstat -g cmsm command: Print Connection Manager information (Administrator's Reference)

Converting the Connection Manager configuration file

The Connection Manager configuration file in versions of IBM Informix Client Software Development Kit (Client SDK) prior to version 3.70.xC3 are incompatible with the current version of the Connection Manager. You must convert configuration files from versions prior to 3.70.xC3 using command line options.

Prerequisites:
- On UNIX, you must be logged in as user informix or root to convert a configuration file. On Windows, you must be a user in the Informix Admin group.
- You must have read permission on the configuration file you want to convert.
- You must have write permission on the configuration file you want to create.

To convert a Connection Manager file to the current format:

1. Log on to the computer on which the Connection Manager instance is running.
2. Run the **oncmsm** utility with the **-n** option, specifying the old and new configuration file paths and file names.

To use the new configuration file, you must start the Connection Manager. (You can convert a configuration file whether the Connection Manager is running or not.)

If you convert multiple configuration files, you must combine SLA definitions from each file into a single configuration file.

For example, to convert a configuration file named: `$INFORMIXDIR/etc/cmsm.cfg` to the current Connection Manager format and store the result in `new_configuration_file`, run the following command:

`oncmsm -n new_configuration_file`

To convert a configuration file named `old_configuration_file` to the latest format and name the file `new_configuration_file`, use the following command:

`oncmsm -c old_configuration_file -n new_configuration_file`

Related reference:

The oncmsm utility (Administrator's Reference)

Chapter 24. Cluster failover, redirection, and restoration

To maintain availability, you must plan for the failover of primary servers, redirecting client connections from unavailable servers, and restoring the cluster to its original configuration after a failure.

Failover configuration

A failure in a high-availability cluster means that one of the servers is no longer available. Because the purpose of a high-availability cluster is to maintain availability, you must plan that if a primary server becomes unavailable, a secondary server in the cluster becomes the primary server. The best way to control failover is to configure Connection Manager to perform failover to the servers that you specify.

You set the DRAUTO configuration parameter to specify how failover is performed, and then use one of the following methods to create a new primary server when the existing primary server fails:

- Connection Manager
- ISV cluster management software
- Manual switchover

Related concepts:

Chapter 23, "Connection Management," on page 23-1

Related reference:

➥ DRAUTO configuration parameter (Administrator's Reference)

Failover with ISV cluster management software

You can use independent software vendor (ISV) cluster management software instead of the Connection Manager to manage failover processing in high-availability cluster environments.

If the primary server in a high-availability cluster encounters a problem that requires a secondary server to assume the role of the primary, it is important that, before performing the actual failover, disk I/O is prohibited on the failed primary server and is allowed on the new primary server. In addition, network access to the failed primary server must be prevented. This is especially true for SD secondary servers, where disk corruption can occur if these steps are not done correctly.

The mechanism for enabling disk I/O operations from a server in a high-availability cluster environment is known as *I/O Fencing*. I/O Fencing is configured using a callback script. When a failure of the primary server occurs, the failover process executes a callback script on the secondary server before the secondary server assumes the role of the primary server. The script calls any I/O specific commands to enable or disable disk access. The script enables write access to the shared disk on the server that is to become the primary server, and disables write access to the shared disk on the failed server.

Use the FAILOVER_CALLBACK configuration parameter to specify the name of the script to run when a database server transitions from a secondary server to a

primary server, or from a secondary server to a standard server. A template script named **ifx_failover_callback.sh** (UNIX) or **ifx_failover_callback.bat** (Windows) is provided in the `$INFORMIXDIR/etc` directory. When configured, the script specified by FAILOVER_CALLBACK is executed before the secondary server is switched to a primary or standard server.

You can test the failover script by performing one of the following actions, depending on your type of high-availability cluster:
- Converting an SD secondary server into a primary server.
- If the DRAUTO configuration parameter is set to 0, shutting down the primary server and convert the HDR secondary server to standard mode.
- If the DRAUTO configuration parameter is set to 1, shutting down the primary server in an HDR pair.
- Shutting down the primary server in a remote stand-alone cluster and converting the RS secondary server to standard mode.

An `Invoking Failover Callback` message is in the `online.log` listing the path and file name of the failover script after it is run.

See the information about the FAILOVER_CALLBACK configuration parameter in the *IBM Informix Administrator's Reference*.

If the script specified by FAILOVER_CALLBACK fails (that is, if it returns a non-zero exit code), the failover of the secondary to the primary (or standard) server also fails. In this case, the DBA must manually perform the failover.

I/O fencing for shared file systems

You can configure I/O fencing to protect shared resources in a high-availability cluster environment.

A software or hardware malfunction might cause unresolved operations to be written to shared storage devices. I/O fencing can be used to isolate a server and prevent it from accessing shared storage. I/O fencing must be used if maintenance or testing is being performed on a server in a high-availability cluster. If a problem is detected on a server or within an application, the cluster manager can detect the problem and prevent the server from connecting to the shared data.

You can configure a script to run when a primary server fails. Fencing commands are called by setting the FAILOVER_CALLBACK configuration parameter, which runs a script when a failover is initiated.

Although I/O fencing is not required in order to use Informix database software, configuring I/O fencing must be used to protect shared-disk systems from inadvertent loss

Types of I/O fencing

Several types of I/O fencing are available, including:
- Power fencing - Powers off nodes if a problem is detected.
- Fibre Channel Switch Fencing (requires SCSI-3 persistent group reservation) - Blocks a port on the fibre channel device by removing the problem node's reservation.

Perhaps the most common method of implementing I/O fencing is to use fibre channel fencing. The fibre channel switch supports the industry-standard SCSI-3 persistent group reservation (PR) technology. PR technology allows a set of systems to have temporary registrations with the disk and to coordinate a write-exclusive reservation with the disk containing the data.

In most cases, cluster manager software must be installed. The cluster manager software provides the drivers and utilities required to issue commands to the fibre channel switch. For example, the Linux Cluster Suite provides a script named **fence_scsi**. Sun Cluster provides a command named **scdidadm**.

Other fencing methods are also available depending on the cluster manger software used and different hardware capabilities.

The General Parallel File System (GPFS) is a high performance shared disk clustered file system developed by IBM. The file system is available for use with AIX, Linux, and Windows platforms. The file system can be used with the IBM HACMP™ Cluster Management Software. GPFS uses the SCSI PR fencing mechanism to support I/O fencing. Fencing issues must be resolved by the IBM GPFS support team.

Implementing I/O fencing

I/O fencing can be configured on several platforms, including:
- Linux
- Solaris
- AIX
- Windows

see the documentation provided by the manufacturer of your equipment for specific information about configuring I/O fencing.

Related reference:

↪ FAILOVER_CALLBACK configuration parameter (Administrator's Reference)

Cluster failures

A high-availability cluster failure is a loss of connection between the database servers in a cluster that can be caused by several different situations.

Any of the following situations might cause a cluster failure:
- A catastrophic failure (such as a fire or large earthquake) at the site of one of the database servers
- A disruption of the networking cables that join the database servers
- An excessive delay in processing on one of the database servers
- A disk failure on a secondary database server that is not resolved by a mirror chunk

Tip: A cluster failure does not necessarily mean that one of the database servers has failed, only that the connection between database servers is lost.

The database server interprets either of the following conditions as a cluster failure:
- A specified timeout value was exceeded.

During normal cluster operation, a database server expects confirmation of communication from the other database servers in the cluster. Each database server in the cluster has an ONCONFIG parameter, DRTIMEOUT, that specifies a number of seconds. If confirmation from the other database server in a cluster does not return within the number of seconds that DRTIMEOUT specifies, the database server assumes that a cluster failure has occurred.

- A database server in the cluster does not respond to the periodic messaging (*pinging*) attempts over the network.

  The database servers ping each other regardless of whether the primary database server sends any records to the secondary database servers. If one database server does not respond to four sequential ping attempts, the other database server assumes that a cluster failure has occurred.

  Each database server in the cluster sends a ping to the other database servers in the cluster when the number of seconds specified by the DRTIMEOUT parameter on that database server has passed.

After a database server detects a cluster failure, it writes a message to its message log (for example, `DR: receive error`) and turns data replication off. If a cluster failure occurs, the connection between the two database servers is dropped and the secondary database server remains in read-only mode.

If a secondary database server remains online after a cluster failure, and the DRAUTO configuration parameter is set to 1 (RETAIN_TYPE), the type of that database server changes automatically to standard. If DRAUTO is set to 0 (off), the secondary database server periodically attempts to reestablish communication with the primary database server. If DRAUTO is set to 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the original primary server fails, rather than when the original primary server is restarted.

If Connection Manager is enabled, setting DRAUTO to 3 prevents the possibility of having multiple primary servers within a high-availability cluster. If an attempt is made to bring a server online as a primary server and DRAUTO=3, then the Connection Manager verifies that there are no other active primary servers in the cluster. If another primary server is active, then the Connection Manager rejects the request.

If the primary database server fails, the secondary database server can operate in the following ways:

- The secondary database server can remain in logical-recovery mode. In this case, no action is taken. This is desirable if you expect the HDR connection to be restored very soon.
- The secondary database server can automatically become a standard database server. This action is called *automatic switchover*.
- The secondary database server can become a standard database server if you use *manual switchover* to change the database server mode to standard.

Related concepts:

"Checkpoints between database servers" on page 22-6

Automatic switchover

Automatic switchover means that after the failure of the primary server in a cluster, a secondary database server automatically becomes a standard database server if the DRAUTO configuration parameter is set to 1, or a primary database server if the DRAUTO configuration parameter is set to 2.

Automatic switchover first rolls back any open transactions and then comes into online mode as a primary database server. Automatic switchover occurs only if the parameter DRAUTO in the onconfig file of the secondary database server is set to 1 (RETAIN_TYPE) or 2 (REVERSE_TYPE).

Because the secondary database server becomes a standard or primary database server, you must be sure that either:
- The secondary database server has enough logical-log disk space to allow processing to continue without backing up logical-log files.
- The logical-log files are backed up.

The automatic switchover changes only the type of the database server. It does not redirect client applications to the secondary database server.

Automatic switchover has the following advantages over manual switchover:
- Clients that you redirect from the primary database server to the secondary database server can continue to write and update data.
- The switchover does not depend on an operator monitoring the message log to see when high-availability data-replication failures occur and then manually switching the secondary database server to a standard database server.

Automatic switchover requires a very stable network to function appropriately.

When you succeed in bringing the original primary database server back online, the cluster connection is automatically established.
- If DRAUTO is set to RETAIN_TYPE, the secondary-turned-standard database server goes through a graceful shutdown (to ensure that all clients that might potentially write to the database server are not connected) and then switches back to a secondary database server.
- If DRAUTO is set to REVERSE_TYPE, the secondary-turned-primary database server switches directly to the primary type. No shutdown occurs. Any applications connected to this database server can stay connected. The original primary database server is switched to a secondary database server.

Related concepts:

"Redirection and connectivity for data-replication clients" on page 24-6

"Automatic switchover without a reliable network"

Related reference:

"Restart if the primary server fails" on page 24-17

Automatic switchover without a reliable network

Although automatic switchover might seem to be the best solution, it is not appropriate for all environments.

Consider what might happen if the primary database server does not actually fail, but the secondary database server registers that the primary has failed. For example, if the secondary database server does not receive responses when it signals (pings) the primary database server because of a slow or unstable network, the secondary server assumes that the primary database server failed and switches automatically to the standard type. If the primary database server also does not receive responses when it signals the secondary database server, it assumes that the secondary database server failed and turns off data replication but remains in online mode. Now the primary database server and the secondary database server (switched to the standard type) are both in online mode.

If clients can update the data on both database servers independently, the database servers in the pair reach a state in which each database server has the logical-log records that are required by the other. In this situation, you must start again and perform initial data replication with a level-0 dbspace backup of one entire database server, as described in "Starting HDR for the First Time" on page 21-6. Therefore, if your network is not entirely stable, you might not want to use automatic switchover. HDR cannot be reinstated without the risk of losing transactions on the previous secondary server.

Related concepts:

"Automatic switchover" on page 24-4

Manual switchover

Manual switchover means that the administrator of the secondary database server changes the type of the secondary database server to standard.

The secondary database server rolls back any open transactions and then comes into online mode as a standard database server, so that it can accept updates from client applications.

Related concepts:

"Changing the database server type" on page 22-38

Connecting offline servers to the new primary server

After failover, you must reconnect any offline secondary servers to the new primary server.

If the primary server in a high-availability cluster fails, all online secondary servers are notified of the failure. The secondary servers connect to the new primary server and continue to operate. However, RS secondary servers and HDR secondary servers that are not online at the time of the failover do not receive the failover notification, and attempts to connect to the original (failed) primary server when the servers are back online. In this case, the primary server must be manually reset on those secondary servers.

Run the following commands on secondary servers that were offline when the failover occurred.

- For HDR secondary servers:
  ```
  oninit -PHY
  onmode -d secondary new_primary
  ```

  new_primary indicates the name of the current primary server.
- For RS secondary servers:
  ```
  oninit -PHY
  onmode -d RSS new_primary
  ```

  new_primary indicates the name of the current primary server.

Redirection and connectivity for data-replication clients

When any server in a high-availability cluster becomes unavailable, any client connections to it should be redirected to an available server. The best way to handle connection redirection is to use Connection Manager to automatically reconnect client connections to the servers that you specify in service level agreements. Alternatively, you can control connection redirection with environment variables or connection information.

If you do not use Connection Manager, you can automatically redirect clients to different database servers in a cluster by configuring applications to connect to the server group to which the servers belong. When you create a connection to a server group, by default the connection is made to the current primary server in the group. If replication is down because one of the servers failed, the connection is made to the server which is online (in Standard mode or Primary mode without a secondary server). You can also automate this action from within the application. Some of the client connectivity drivers included in the IBM Informix Client Software Development Kit (Client SDK) have specific mechanisms for automating redirection. For details, see the IBM Informix Client Software Development Kit (Client SDK) documentation.

When you design client applications, you must make some decisions on redirection strategies. Specifically, you must decide whether to handle redirection within the application and which redirection mechanism to use. The three different redirection mechanisms are as follows:

- Automatic redirection with the **DBPATH** environment variable
- Administrator-controlled redirection with the connectivity information
- User-controlled redirection with the **INFORMIXSERVER** environment variable

The mechanism that you employ determines which CONNECT syntax you can use in your application.

Related concepts:

Redirecting clients automatically with the DBPATH environment variable

You can use the DBPATH environment variable in applications to redirect connections.

What the administrator must do

Administrators take no action to redirect clients, but they might be required to attend to the type of the database server.

What the user must do

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities.

If your applications do not include such code, users who are running clients must quit and restart all applications.

How the DBPATH redirection method works

When an application does not explicitly specify a database server in the CONNECT statement, and the database server that the **INFORMIXSERVER** environment variable specifies is unavailable, the client uses the **DBPATH** environment variable to locate the database (and database server).

If one of the database servers in a replication pair is unusable, applications that use that database server are not required to reset their **INFORMIXSERVER** environment variable if their **DBPATH** environment variable is set to the other database server in

the pair. Their **INFORMIXSERVER** environment variable must always contain the name of the database server that they use regularly, and their **DBPATH** environment variable must always contain the name of the alternative database server in the pair.

For example, if applications normally use a database server called **cliff_ol**, and the database server paired with **cliff_ol** in a replication pair is called **beach_ol**, the environment variables for those applications would be as follows:

```
INFORMIXSERVER  cliff_ol
DBPATH          //beach_ol
```

Because the **DBPATH** environment variable is read only (if required) when an application issues a CONNECT statement, applications must restart in order for redirection to occur.

An application can contain code that tests whether a connection has failed and, if so, attempts to reconnect. If an application has this code, you are not required to restart it.

You can use the CONNECT TO *database* statement with this method of redirection. For this method to work, you cannot use any of the following statements:
- CONNECT TO DEFAULT
- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

The reason for this restriction is that an application does not use **DBPATH** if a CONNECT statement specifies a database server. For more information about **DBPATH**, see the *IBM Informix Guide to SQL: Reference*.

Redirecting clients with the connectivity information

You can redirect client connections to the new primary server by using `sqlhosts` information.

The connectivity information-redirection method relies on the fact that when an application connects to a database server, it uses the connectivity information to find that database server.

If one of the database servers in a replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in "Changing client connectivity information" on page 24-9, the fields of the unavailable database server (except for the **dbservername** field) are changed to point to the remaining database server in the replication pair.

Because the connectivity information is read when a CONNECT statement is issued, applications might be required to restart for redirection to occur. Applications can contain code that tests whether a connection failed and that issues a reconnect statement, if necessary. If a connection failed, redirection is automatic, and you are not required to restart applications for redirection to occur.

Applications can use the following connectivity statements to support this method of redirection:
- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

Applications can also use the following connectivity statements, provided that the **INFORMIXSERVER** environment variable always remains set to the same database server name and the **DBPATH** environment variable is not set:

- CONNECT TO DEFAULT
- CONNECT TO *database*

On UNIX, the **INFORMIXSQLHOSTS** environment variable specifies the full path name and file name of the connection information in $INFORMIXDIR/etc/sqlhosts. For more information about **INFORMIXSQLHOSTS**, see the *IBM Informix Guide to SQL: Reference*.

On Windows, the connectivity information is in a key in the Windows registry.

Related concepts:

"The sqlhosts file and the SQLHOSTS registry key" on page 2-17

"Trusted-host information" on page 2-13

"Trusted-user information" on page 2-14

Changing client connectivity information

To use the connectivity information to redirect clients, you must change the connectivity information for the clients and change other connectivity files, if necessary.

To change the connectivity information about the client computer:

1. In the sqlhosts file or registry, comment out the entry for the failed database server.
2. Add an entry that specifies the dbservername of the failed database server in the **servername** field and information for the database server to which you are redirecting clients in the **nettype**, **hostname**, and **servicename** fields.
3. Use the following options in the sqlhosts file or registry to redirect applications to another database server if a failure occurs:
   a. Connection-redirection option
   b. End-of-group option
   c. Group option
4. Edit the /etc/hosts file on UNIX or hosts file on Windows to add an entry, if necessary, for the **hostname** of the computer that is running the database server to which you are redirecting clients.
5. Edit the /etc/services file on UNIX or services file on Windows to add an entry, if necessary, for the **servicename** of the database server to which you are redirecting clients.

The following figure shows how connectivity values might be modified to redirect clients.

You are not required to change entries in the connectivity information on either of the computers that is running the database servers.

marsh

Client

delta

Client

Client

river

Client

cliff_ol   onsoctc   cliff   ol_cl

cliff_ol   onsoctcp cliff   ol_cl
beach_   onsoctcp beach   ol_bc

beach_o   onsoctcp   beach   ol_bc

cliff

cliff_ol

beach

beach_ol

Before failure of **cliff_ol**

After failure of **cliff_ol**

marsh

Client

delta

Client

Client

river

Client

#cliff_ol   onsoctcp   cliff   ol_cl
cliff_ol   onsoctcp   beach   ol_bc

#cliff_ol   onsoctcp cliff   ol_cl
beach_ol onsoctcp beach   ol_bc
cliff_ol   onsoctcp beach   ol_bc

cliff

cliff_ol

beach

beach_ol

/etc/hosts
beach

/etc/services
ol_bc

Figure 24-1. Connectivity values before and after a failure of the cliff_ol database server

Related concepts:

Chapter 2, "Client/server communication," on page 2-1

Related tasks:

"Configuring secure connections for clusters" on page 21-5

Related reference:

"sqlhosts Connectivity information" on page 2-21

Connecting to the database server

After the administrator changes the connectivity information and other
connectivity files (if required), clients connect to the database server to which the
administrator redirects them when they issue their next CONNECT statement.

If your applications contain code that tests if a connection has failed and issues a
reconnect statement if necessary, redirection is handled automatically. The user has
no responsibilities. If your applications do not include such code, users who are
running clients must quit and restart all applications.

Automatic redirection with server groups

You can use the group option in the `sqlhosts` file to specify a server group to which applications connect instead of an individual database server. To make connection redirection automatic, add a database server definition for both the primary and secondary servers to the server group definition. By default, when a connection request is made to an HDR server group, the connection is routed to the primary server. If the primary server is unavailable, then the connection request is routed to the secondary server that gets promoted to the primary server after failover processing.

For example, the following `sqlhosts` entries represent an HDR server group, **g_hdr**, with a primary server definition, **hdr_prim**, and a secondary server definition, **hdr_sec**.

```
#dbservername   nettype   hostname      servicename   options
g_hdr           group     -             -             i=1
hdr_prim        ontlitcp  machine1pri   port1         g=g_hdr
hdr_sec         ontlitcp  machine1sec   port1         g=g_hdr
```

Applications can use the following connectivity statements to support this method of redirection:

*   CONNECT TO *database@dbserver_group*
*   CONNECT TO *@dbserver_group*

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Redirecting clients with the INFORMIXSERVER environment variable

The **INFORMIXSERVER** environment variable redirection method can be used when an application does not explicitly specify a database server in the CONNECT statement, so that the database server connects to the client that the **INFORMIXSERVER** environment variable specifies.

If one of the database servers in a cluster is unusable, applications that use that database server can reset their **INFORMIXSERVER** environment variable to the another database server in the cluster to access the same data.

Applications read the value of the **INFORMIXSERVER** environment variable only when they start. Therefore, applications must be restarted to recognize a change in the environment variable.

To support this method of redirection, you can use the following connectivity statements:

*   CONNECT TO DEFAULT
*   CONNECT TO *database*

You cannot use the CONNECT TO *database@dbserver* or CONNECT TO *@dbserver* statements for this method. When a database server is explicitly named, the CONNECT statement does not use the **INFORMIXSERVER** environment variable to find a database server.

Administrators take no action to redirect the clients, but they might be required to change the type of the database server.

Users who are running client applications must perform the following three steps when they decide to redirect clients with the **INFORMIXSERVER** environment variable.

To redirect clients with the **INFORMIXSERVER** environment variable:

1. Quit their applications.
2. Change their **INFORMIXSERVER** environment variable to hold the name of the other database server in the replication pair.
3. Restart their applications.

Redirecting clients with application code

If you use the **DBPATH** environment variable or connectivity information to redirect connections, you can include in your clients a routine that handles errors when clients encounter a cluster failure. The routine can call another function that contains a loop that tries repeatedly to connect to the other database server in the cluster. This routine redirects clients without requiring the user to exit the application and restart it.

The following example shows an example of a function in a client application using the **DBPATH** redirection mechanism that loops as it attempts to reconnect. After it establishes a connection, it also tests the type of the database server to make sure it is not a secondary database server. If the database server is still a secondary type, it calls another function to alert the user (or database server administrator) that the database server cannot accept updates.

```
/* The routine assumes that the INFORMIXSERVER environment
 * variable is set to the database server that the client
 * normally uses and that the DBPATH environment variable
 * is set to the other database server in the pair.
 */

#define SLEEPTIME 15
#define MAXTRIES 10

main()
{
   int connected = 0;
   int tries;
   for (tries = 0;tries < MAXTRIES && connected == 0;tries++)
   {
      EXEC SQL CONNECT TO "superstores";
      if (strcmp(SQLSTATE,"00000"))
      {
         if (sqlca.sqlwarn.sqlwarn6 != 'W')
         {
            notify_admin();
            if (tries < MAXTRIES - 1)
               sleep(SLEEPTIME);
         }
         else connected =1;
      }
   }
   return ((tries == MAXTRIES)? -1:0);
   }
```

This example assumes the **DBPATH** redirection mechanism and uses a form of the CONNECT statement that supports the **DBPATH** redirection method. If you used the connectivity information to redirect, you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "superstores@cliff_ol";
```

In this example, superstores@cliff_ol is a database on a database server that the client computer recognizes. For redirection to occur, the administrator must change the connectivity information to make that name refer to a different database server. You might be required to adjust the amount of time that the client waits before it tries to connect or the number of tries that the function makes. Provide enough time for an administrative action on the database server (to change the connectivity information or change the type of the secondary database server to standard).

Comparison of redirection methods

The different redirection methods have different requirements.

The following table summarizes the differences among redirection mechanisms.

Table 24-1. Comparison of redirection methods for different connectivity strategies

	Automatic redirection	User redirection
DBPATH		
When is a client redirected?	When the client next tries to connect with a specified database	When the client next tries to connect with a specified database
Do clients require being restarted to be redirected?	No	Yes
What is the scope of the redirection?	Individual clients redirected	Individual clients redirected
Are changes to environment variables required?	No	No
Connectivity information		
When is a client redirected?	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server
Do clients require being restarted to be redirected?	No	Yes
What is the scope of the redirection?	All clients that use a given database server redirected	Individual clients redirected
Are changes to environment variables required?	No	No
INFORMIXDIR		
When is a client redirected?		When the client restarts and reads a new value for the **INFORMIXSERVER** environment variable
Do clients require being restarted to be redirected?		Yes
What is the scope of the redirection?		Individual clients redirected
Are changes to environment variables required?		Yes
Connection Manager		

Table 24-1. Comparison of redirection methods for different connectivity strategies (continued)

	Automatic redirection	User redirection
When is a client redirected?		When the configured service level agreement is attained.
Do clients require being restarted to be redirected?		No
What is the scope of the redirection?		Individual clients redirected
Are changes to environment variables required?		No

Recover HDR and RS clusters after failure

When you restore the HDR or RS cluster back to the original configuration, you might need to restore critical media, as well as restart and configure servers in the cluster.

The result of a disk failure depends on whether the disk failure occurs on the primary or the secondary database server, whether the chunks on the disk contain critical media (the root dbspace, a logical-log file, or the physical log), and whether the chunks are mirrored.

If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.

If chunks are not mirrored, the procedure for restoring a primary database server depends on whether the disk that failed contains critical media:

- If the disk contains critical media, the primary database server fails. You must perform a full restore using the primary dbspace backups (or the secondary dbspace backups if a secondary database server was switched to standard mode and activity redirected).
- If the disk does not contain critical media, you can restore the affected dbspaces individually with a warm restore. A warm restore consists of two parts: first a restore of the failed dbspace from a backup and next a logical restore of all logical-log records written since that dbspace backup. You must back up all logical-log files before you perform the warm restore.

If chunks are not mirrored, a secondary database server fails if the disk contains critical media but remains online if the disk does not contain critical media. In both cases, you must perform a full restore using the dbspace backups on the primary database server. In the second case, you cannot restore selected dbspaces from the secondary dbspace backup because they might now deviate from the corresponding dbspaces on the primary database server. You must perform a full restore.

Related reference:

"Recovering a cluster after critical data is damaged"

Recovering a cluster after critical data is damaged

If one of the database servers in an HDR or RS cluster experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as

if it has no data on the disks, and you are starting an HDR or RS cluster for the first time. Use the functioning database server with the intact disks as the database server with the data.

Primary server failure

For the following steps, assume that the configuration consists of a primary server named **srv_A** and an HDR secondary server named **srv_B**. The steps for restarting an RS cluster are the similar.

To restart HDR after a critical media failure:

1. The DRAUTO configuration parameter on **srv_B** affects what you do next
   - If it is set to 0, then you must convert the server to the primary server by running the **onmode -d make primary** command.
   - If it is set to 1 (RETAIN_TYPE), then convert the server to the primary server by running the **onmode -d make primary** command.
   - If it is set to 2 (REVERSE_TYPE), the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.
2. Restore **srv_A** (the primary database server) from the last dbspace backup.
3. Use the **onmode -d** command to set **srv_A** to an HDR secondary database server and to start HDR.

   The **onmode -d** command starts a logical recovery from the logical-log files on **srv_B**. If logical recovery cannot complete because you backed up and freed logical-log files on **srv_B**, HDR does not start until you perform the next step.
4. Apply the logical-log files from **srv_B** (the new primary database server), which were backed up to tape. The HDR pair is now operational; however the roles of **srv_A** and **srv_B** are swapped. To swap **srv_A** and **srv_B** back to their original roles, follow the instructions: "Recovering an HDR cluster after the secondary server became the primary server" on page 24-17.

Table 24-2. Steps for restarting HDR after a critical media failure on the primary database server

Step	On the primary database server (svr_A)	On the secondary database server (svr_B)
1.		**onmode** command **onmode -d make primary srv_A**
2.	**ontape** command **ontape -p** ON-Bar command **onbar -r -p**	
3.	**onmode** command **onmode -d secondary srv_B**	
4.	**ontape** command **ontape -l** ON-Bar command **onbar -r -l**	

Secondary server failure

If the secondary database server suffers a critical media failure, recover the cluster by following the steps for starting a cluster for the first time.

Primary and secondary server failure

In the unfortunate event that both of the computers that are running database servers in a replication pair experience a failure that damages the root dbspace, the dbspaces that contain logical-log files or the physical log, you must restart the cluster.

To restart an HDR or RS cluster after a critical media failure on both database servers:

1. Restore the primary database server from the storage space and logical-log backup.
2. After you restore the primary database server, treat the other failed database server as if it had no data on the disks and you were starting the HDR or RS cluster for the first time.

Related concepts:

"Recover HDR and RS clusters after failure" on page 24-14

Related tasks:

"Starting HDR for the First Time" on page 21-6

Restarting HDR or RS clusters after a network failure

After a network failure, the HDR or RS cluster might need to be restarted. After a network failure, the primary database server is in online mode, and the secondary database server is in read-only mode. Replication is turned off on both database servers (state = off).

Restarting the cluster might not be necessary because the primary database server attempts to reconnect every 10 seconds and displays a message regarding the inability to connect every 2 minutes.

If the cluster does not restart automatically, when the connection is reestablished, you can restart the cluster by running the following commands on the secondary server:

- **onmode -d secondary** *primary_name* to make it the primary server.

Restarting HDR or RS clusters if the secondary server fails

If you must restart HDR or an RS cluster after a failure of a secondary server, in addition to starting the secondary server, you might need to perform a logical log restore on the secondary server.

The steps assume that you have been backing up logical-log files on the primary database server as necessary since the failure of the secondary database server.

Table 24-3. Steps in restarting after a failure on the secondary database server

Step	On the primary	On the secondary
1.	The primary database server must be in online mode.	**oninit** If you receive the following message in the message log, continue with step 2:`DR: Start Failure recovery from tape`
2.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l**

Recovering an HDR cluster after the secondary server became the primary server

If a secondary server in an HDR cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.

Suppose the primary server, named **srv_pri**, has encountered an error that has caused it to fail over to an HDR secondary server named **srv_hdr_sec**. At this point, the primary server is **srv_hdr_sec**, and any other secondary servers in the cluster are now pointing to **srv_hdr_sec**.

To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. Initialize **srv_pri** as the HDR secondary server by running the appropriate command:

   UNIX systems:

   `$INFORMIXDIR/bin/hdrmksec.sh srv_hdr_sec`

   Windows systems:

   `hdrmksec.bat srv_hdr_sec`

2. Change **srv_pri** to the primary server by running:

   `onmode -d make primary srv_pri`

   This command makes **srv_pri** the primary server, and redirects any other secondary servers in the cluster to point to the new primary server. The command also shuts down the old HDR primary (**srv_hdr_sec**) because only a single primary server can exist in a high-availability environment.

3. Initialize **srv_hdr_sec** as the HDR secondary server by running the following command:

   On UNIX systems:

   `$INFORMIXDIR/bin/hdrmksec.sh srv_pri`

   On Windows systems:

   `hdrmksec.bat srv_pri`

Restart if the primary server fails

The process for restarting an HDR or RS cluster after the primary server fails depends on whether a secondary server became the primary server, and the method by which the secondary server became the primary server.

The secondary database server was not changed to a standard database server

If you must restart an HDR or RS cluster after a failure of the primary database server if the secondary database server is not changed to standard, start the primary database server by using the **oninit** command.

The secondary database server was changed to a standard database server manually

If you must restart an HDR or RS cluster after a failure of the primary database server, and you have manually changed the secondary database server to be a standard database server, complete the steps in the following table.

Table 24-4. Steps to restart if you changed the secondary database server to standard

Step	On the primary database server	On the secondary database server
1.		**onmode -s** This step takes the secondary database server (now standard) to quiescent mode. All clients that are connected to this database server must disconnect. Applications that perform updates must be redirected to the primary.
2.		**onmode -d secondary** *prim_name*
3.	**oninit** If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If you have backed up and freed the logical-log files on the secondary, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 4). For **ontape** users: If you want to read the logical-log records over the network, set the logical-log tape device to a device on the computer that is running the secondary database server.	
4.	If you are prompted to recover logical-log records from tape, perform this step. **ontape** command **ontape -l** ON-Bar command **onbar -r -l**	

The secondary database server was changed to a standard database server automatically

If you must restart an HDR or RS cluster after a failure of the primary database server, and the secondary database server was automatically changed to a standard database server, complete the steps shown in the following table.

Table 24-5. Steps to restart if you changed the secondary database server to standard automatically

Step	On the primary database server	On the secondary database server
1.	**% oninit** If DRAUTO = 1, the type of this database server is set to primary. If DRAUTO = 2, the type of this database server is set to secondary when it is restarted. If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If logical-log files that you have backed up and freed are on the secondary database server, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 2). For **ontape** users: • Set the logical-log tape device to a device on the computer running the secondary database server.	If DRAUTO = 1, the secondary database server automatically goes through graceful shutdown when you bring the primary back up. This ensures that all clients are disconnected. The type is then switched back to secondary. Any applications that perform updates must be redirected back to the primary database server. If DRAUTO = 2, the secondary database server switches automatically to primary. The old primary database server becomes a secondary database server after it restarts and connects to the other server and determines that it is now a primary database server.
2.	If you are prompted to recover logical-log records from tape, perform this step. **ontape** command **% ontape -l** ON-Bar command **onbar -r -l**	

Related concepts:
"Automatic switchover" on page 24-4

Recovering a shared-disk cluster after data is damaged

If a shared-disk cluster fails, you must perform a restore of affected dbspaces. The type of restore that you need to perform depends on whether critical data is damaged.

Related concepts:
"Backup and restore with high-availability clusters" on page 22-37

Critical data is damaged

If the primary server experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks. You must perform a full restore of the primary server. In this situation, the primary server and the SD secondary servers are offline.

To recover a shared-disk cluster after critical media failure:

1. Perform a full restore of the primary server. Run one of the following commands, depending on whether the backup was performed with ON-Bar or the **ontape** utility:

   - **onbar -r**
   - **ontape -r**

   The primary server restarts after the restore is complete.

2. Restart the SD secondary servers.

Alternatively, you can perform a cold restore of the critical dbspaces on the primary server, restart the SD secondary servers, and then perform a warm restore of non-critical dbspaces.

Critical data is not damaged

If a disk that does not contain critical media fails, you can restore the affected dbspaces with a warm restore. In this situation the primary server and the SD secondary servers are online.

To recover non-critical data in a shared-disk cluster:

1. Shut down and restart the SD secondary servers.

2. Perform a warm restore of the affected dbspaces. Run one of the following commands, depending on whether the backup was performed with ON-Bar or the **ontape** utility:

   - **onbar -r** with the names of the dbspaces to restore
   - **ontape -r -D** with the names of the dbspaces to restore

Recovering an SD cluster after the secondary server became the primary server

If a secondary server in an SD cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.

In this example, the primary server, named **srv_pri**, has failed over to an SD secondary server named **srv_sds_sec**. At this point, the primary server is **srv_sds_sec**, and any other secondary servers in the cluster are now pointing to **srv_sds_sec**. To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. If necessary, set the following parameters in the onconfig file of **srv_pri**:
   ```
   SDS_ENABLE 1
   SDS_PAGING <path 1>,<path 2>
   SDS_TEMPDBS <dbsname>,<dbspath>,<pagesize>,<offset>,<size>
   ```
   The *dbsname* value must be unique. In addition, the *dbsname* must be unique among all existing dbspaces, blobspaces, and sbspaces, including those (possibly disabled) temporary spaces that are inherited from a primary server. If you have multiple SD secondary servers, the *dbsname* value must be unique for each server and not shared with any other SD secondary server or the primary server. See "Setting up a shared disk secondary server" on page 21-19 for more information about setting these parameters.

2. Initialize **srv_pri** as an SD secondary server by running the **oninit** command on **srv_pri**.

3. Perform a manual failover of **srv_pri** to make it the primary server:

```
onmode -d make primary srv_pri
```

The previous command removes **srv_sds_sec** from the cluster and makes **srv_pri** the primary server.

4. Restore **srv_sds_sec** as an SD secondary server by running the **oninit** command on **srv_sds_sec**.

Part 6. Distributed data

Chapter 25. Multiphase commit protocols

A *two-phase commit protocol* ensures that transactions are uniformly committed or rolled back across multiple database servers. You can use IBM Informix database servers with IBM Informix Enterprise Gateway products or transaction managers to manipulate data in non-Informix databases. Distributed queries across IBM Informix database servers support two-phase commit.

The *heterogeneous commit protocol* ensures that updates to one or more IBM Informix databases and one non-Informix database in a single transaction are uniformly committed or rolled back.

These topics contain information about the use of the two-phase commit protocol. For information about recovering manually from a failed two-phase commit transaction, see Chapter 26, "Manually recovering from failed two-phase commit," on page 26-1.

These topics also contain information about using transaction support for XA-compliant, external data sources, which can participate in two-phase commit transactions. See "Informix transaction support for XA-compliant, external data sources" on page 25-2.

Transaction managers

Transaction managers support two-phase commit and roll back. For example, if your database is IBM Informix, your accounting system is Oracle, and your remittance system is Sybase, you can use a transaction manager to communicate between the different databases. You also can use transaction managers to ensure data consistency between IBM Informix or non-Informix databases by using distributed transactions instead of Enterprise Replication or High-Availability Data Replication.

TP/XA Library with a transaction manager

A *global transaction* is a distributed query where more than one database server is involved in the query. A global transaction environment has the following parts:
- The client application
- The resource manager (IBM Informix database server)
- The transaction manager (vendor software)

TP/XA is a library of functions that lets the database server act as a resource manager in the X/Open DTP environment. Install the TP/XA library as part of IBM Informix ESQL/C to enable communication between a third-party transaction manager and the database server. The X/Open environment supports large-scale, high-performance OLTP applications.

Use TP/XA when your database has the following characteristics:
- Data is distributed across multivendor databases
- Transactions include IBM Informix and non-Informix data

Microsoft Transaction Server (MTS/XA)

The database server supports the Microsoft Transaction Server (MTS/XA) as a transaction manager in the XA environment. To use MTS/XA, install IBM Informix Client Software Development Kit, the latest version of IBM Informix ODBC Driver, and MTS/XA. MTS/XA works on Windows. For more information, contact IBM Informix Technical Support, and see the *IBM Informix Client Products Installation Guide* and the MTS/XA documentation.

Informix transaction support for XA-compliant, external data sources

The IBM Informix Transaction Manager, which is an integral part of Informix, not a separate module, recognizes XA-compliant, external data sources. These data sources can participate in two-phase commit transactions.

The transaction manager runs support routines for each XA-compliant, external data source that participates in a distributed transaction at a particular transactional event, such as prepare, commit, or rollback. This interaction conforms to X/Open XA interface standards.

Transaction support for XA-compliant, external data sources, which are also called *resource managers*, enables you to:

- Create XA-compliant, external data source types and instances of XA-compliant, external data sources.
- Create or modify a user-defined routine (UDR), virtual table interface, or virtual index interface to enable XA-compliant data sources to provide data access mechanisms for external data from XA-compliant data sources.

  The MQ DataBlade Module is an example of a set of UDRs that provide this type of external data access.
- Register XA-compliant, external data sources with Informix.
- Unregister XA-compliant, external data sources.
- Use multiple XA-compliant, external data sources within the same global transaction.

The transaction coordination with an XA-compliant, external data source is supported only in IBM Informix logged databases and ANSI-compliant databases, since these databases support transactions. Transaction coordination with an XA-compliant, external data source is not supported in non-logged databases.

You can use the following DDL statements, which are extensions to SQL statements to manage XA data source types and data sources:

Statement	Description
CREATE XADATASOURCE TYPE	Creates a type of XA-compliant, external data source
CREATE XADATASOURCE	Creates an instance of an XA-compliant, external data source

Statement	Description
DROP XADATASOURCE	Deletes an instance of an XA-compliant, external data source
DROP XADATASOURCE TYPE	Deletes a type of XA-compliant, external data source

For more information about these statements, see the *IBM Informix Guide to SQL: Syntax*.

The interaction between IBM Informix and an XA-compliant, external data source occurs through a set of user-defined XA-support routines, such as **xa_open**, **xa_end**, **xa_commit**, and **xa_prepare**. You create these support routines before using the CREATE XADATASOURCE TYPE statement. For more information, see the *IBM Informix DataBlade API Programmer's Guide*.

After you create an external XA-compliant data source, you can register the data source to a current transaction and you can unregister the data source using the **mi_xa_register_xadatasource()** or **ax_reg()** and **mi_xa_unregister_xadatasource()** or **ax_unreg()** functions. In a distributed environment, you must register a data source at the local, coordinator server. Registration is transient, lasting only for the duration of the transaction. For more information about using these functions, see the *IBM Informix DataBlade API Function Reference* and the *IBM Informix DataBlade API Programmer's Guide*.

Use the following **onstat** options to display information about transactions involving XA-compliant data sources:

The onstat option	What XA-compliant data source information this command displays
onstat -x	Displays information about XA participants in a transaction.
onstat -G	Displays information about XA participants in a global transaction.
onstat -g ses *session id*	Displays session information, including information about XA data sources participating in a transaction.

The IBM Informix MQ DataBlade Module provides external data access mechanisms for XA data sources. For information about this DataBlade Module, see the *IBM Informix Database Extensions User's Guide*.

Related concepts:

"XA in high-availability clusters"

XA in high-availability clusters

The X/Open Distributed Transaction Processing (DTP) Model allows an updatable secondary server in a high-availability cluster to serve as a resource manager in a distributed transaction.

There are three types of participants in any XA global transaction:

- Application Program (AP): Defines transaction boundaries and specifies actions that constitute the transaction branch.
- Resource Manager (RM): Provides access to the resources, such as databases.

- Transaction Manager (TM): Assigns identifier (XID) to transaction branches, monitors transaction progress, coordinates the transaction branches into completion and failure recovery.

In a high-availability cluster, sessions are able to create or attach to a transaction branch from any server in the cluster. For example, a transaction branch detached from server_1 can be attached from server_2. The application can connect to the cluster using the Connection Manager without tracking the server on which the transaction began. The transaction manager can also connect to the cluster using the Connection Manager to complete an XA transaction by committing the transaction, rolling it back, or forgetting it, for both loosely and tightly coupled transactions (see "Loosely-coupled and tightly-coupled modes" on page 25-5).

All global transaction branches started from a secondary server are redirected to the primary server using the existing proxy interface. The primary server starts and maintains all transaction branches and performs all requested work associated with the branches.

When an XA transaction is started on an updatable secondary server, a corresponding XA transaction is started on the primary server. The XA transaction on the primary server executes the full life cycle of an XA transaction (start, end, prepare, and commit or roll back). XA transactions on secondary servers are used to support queries that are not redirected to the primary server. When a call to the **xa_end()** function is issued, the XA transaction is freed and the user session is detached from the XA transaction. All XA transaction requests and all write operations issued within the XA transaction are redirected to the primary server.

The following features are specific to the Informix XA implementation:
- All XA interface requests are available on updatable secondary servers (see "Database updates on secondary servers" on page 22-32.
- Starting, preparing, and committing or rolling back XA transactions from an updatable secondary server is supported.
- The **xa_recover()** function, which obtains a list of prepared transaction branches from a resource manager, is supported.
- XA transaction branch migration among high-availability cluster servers is supported. Any server in a cluster can attach to an XA transaction branch irrespective of whether the transaction branch was originated from it.
- XA clients and the Transaction Manager can connect to any high-availability cluster server using the Connection Manager (see Chapter 23, "Connection Management," on page 23-1).
- Redirection of XA requests from secondary servers to the primary server is supported.
- Transaction survival is supported for XA transactions (see "Transaction completion during cluster failover" on page 22-43).
- If a secondary server on which a redirected XA transaction is running fails, the transaction is rolled back.
- Support is provided for SQL transactions that run within the XA environment but which are outside the XA transaction

The following restrictions exist for XA transactions running on secondary servers:
- Resuming a suspended global transaction branch from a different user session (on the same or different secondary server) is not supported.

- A user session cannot attach to a global transaction branch that is associated with a different user session on another secondary server.
- XA transactions have the same restrictions as other data on secondary servers. See "Database updates on secondary servers" on page 22-32.
- XA transactions cannot be started on read-only secondary servers. If an application attempts to create a new XA transaction on a read-only secondary server, it receives XA error code XAER_RMERR. In addition, running xa_prepare(), xa_commit(), or xa_rollback() on a read-only secondary server returns error code XA_NOTA (-4).
- The following XA APIs are supported on read-only secondary servers:
  - xa_open()
  - xa_close()

Related concepts:

Part 5, "High availability and scalability"

"Database updates on secondary servers" on page 22-32

Related reference:

"TP/XA Library with a transaction manager" on page 25-1

"Microsoft Transaction Server (MTS/XA)" on page 25-2

"Informix transaction support for XA-compliant, external data sources" on page 25-2

"Loosely-coupled and tightly-coupled modes"

Loosely-coupled and tightly-coupled modes

The database server supports XA global transactions in loosely coupled and tightly coupled modes:

- *Loosely coupled mode* means that the different database servers coordinate transactions, but do not share resources. The records from all branches of the transactions display as separate transactions in the logical log.
- *Tightly coupled mode* means that the different database servers coordinate transactions and share resources such as locking and logging. The records from all branches of the transactions display as a single transaction in the logical log.

The Tuxedo Transaction Manager, provided by BEA systems, supports loosely coupled mode. Tuxedo operates on both UNIX and Windows.

Windows only: The MTS/XA Transaction Manager, which operates only on Windows, supports the tightly coupled mode. MTS tightly coupled transaction support on the database server includes:

- Support for application programs with two tiers (a business-logic layer and a data-access layer).
- Connection pooling and session pooling.

MTS tightly coupled transaction support does not affect existing loosely coupled-transaction support. The same database server can use both loosely coupled and tightly coupled transaction support at the same time.

MTS tightly coupled transaction support has the following restrictions:

- Temporary tables are limited to one transaction branch. Different transaction branches within one global transaction cannot share a temporary table.
- Different transaction branches within one global transaction cannot share cursors.

- Different transaction branches within one global transaction cannot share an isolation level or lock-wait mode. The isolation level and lock-wait mode of each transaction branch must be set individually or set to the default level. If you want the same isolation level for all transaction branches, you must use SQL to specify this information for each transaction branch.

\

For a complete list of supported transaction managers, contact your marketing representative.

Related concepts:

"XA in high-availability clusters" on page 25-3

Two-phase commit protocol

The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction. The two-phase commit protocol ensures that all participating database servers receive and implement the same action (either to commit or to roll back a transaction), regardless of local or network failure.

If any database server is unable to commit its portion of the transaction, all database servers participating in the transaction must be prevented from committing their work.

When the two-phase commit protocol is used

A database server automatically uses the two-phase commit protocol for any transaction that modifies data on multiple database servers.

For example, suppose three database servers that have the names **australia**, **italy**, and **france**, are connected, as shown in the following figure.
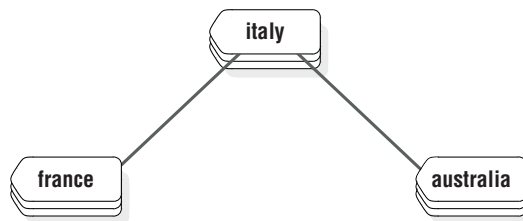
Figure 25-1. Connected database servers

If you run the commands shown in the following example, the result is one update and two inserts at three different database servers.

```
CONNECT TO stores_demo@italy
BEGIN WORK
    UPDATE stores_demo:manufact SET manu_code = 'SHM' WHERE manu_name = 'Shimara'
    INSERT INTO stores_demo@france:manufact VALUES ('SHM', 'Shimara', '30')
    INSERT INTO stores_demo@australia:manufact VALUES ('SHM', 'Shimara', '30')
COMMIT WORK
```

Two-phase commit concepts

Every global transaction has a *coordinator* and one or more *participants*, defined as follows:

- The coordinator directs the resolution of the global transaction. It decides whether the global transaction must be committed or stopped.

  The two-phase commit protocol always assigns the role of coordinator to the current database server. The role of coordinator cannot change during a single transaction. In the sample transaction in "When the two-phase commit protocol is used" on page 25-6, the coordinator is **italy**. If you change the first line in this example to the following statement, the two-phase commit protocol assigns the role of coordinator to **france**:

  ```
  CONNECT TO stores_demo@france
  ```

  Use the **onstat -x** option to display the coordinator for a distributed transaction. For more information, see "Monitor a global transaction" on page 25-15.

- Each participant directs the execution of one *transaction branch*, which is the part of the global transaction involving a single local database. A global transaction includes several transaction branches when:

  – An application uses multiple processes to work for a global transaction

  – Multiple remote applications work for the same global transaction

  In "When the two-phase commit protocol is used" on page 25-6, the participants are **france** and **australia**. The coordinator database server, **italy**, also functions as a participant because it is also doing an update.

The two-phase commit protocol relies on two kinds of communication, *messages* and *logical-log records*:

- Messages pass between the coordinator and each participant. Messages from the coordinator include a transaction identification number and instructions (such as `prepare to commit`, `commit`, or `roll back`). Messages from each participant include the transaction status and reports of action taken (such as `can commit` or `cannot commit`, `committed`, or `rolled back`).

- Logical-log records of the transaction are kept on disk or tape to ensure data integrity and consistency, even if a failure occurs at a participating database server (participant or coordinator).

  For more details, see "Two-phase commit and logical-log records" on page 25-18.

Phases of the two-phase commit protocol

In a two-phase commit transaction, the coordinator sends all the data modification instructions (for example, inserts) to all the participants. Then, the coordinator starts the two-phase commit protocol. The two-phase commit protocol has two parts, the *precommit phase* and the *postdecision phase*.

Precommit phase

During the precommit phase, the coordinator and participants perform the following dialog:

Coordinator

    The coordinator directs each participant database server to prepare to commit the transaction.

Participants

    Every participant notifies the coordinator whether it can commit its transaction branch.

Coordinator

    The coordinator, based on the response from each participant, decides whether to commit or roll back the transaction. It decides to commit only if all participants indicate that they can commit their transaction branches. If

any participant indicates that it is not ready to commit its transaction branch (or if it does not respond), the coordinator decides to end the global transaction.

Postdecision phase

During the postdecision phase, the coordinator and participants perform the following dialog:

Coordinator
> The coordinator writes the commit record or rollback record to the coordinator's logical log and then directs each participant database server to either commit or roll back the transaction.

Participants
> If the coordinator issued a commit message, the participants commit the transaction by writing the commit record to the logical log and then sending a message to the coordinator acknowledging that the transaction was committed. If the coordinator issued a rollback message, the participants roll back the transaction but do not send an acknowledgment to the coordinator.

Coordinator
> If the coordinator issued a message to commit the transaction, it waits to receive acknowledgment from each participant before it ends the global transaction. If the coordinator issued a message to roll back the transaction, it does not wait for acknowledgments from the participants.

How the two-phase commit protocol handles failures

The two-phase commit protocol is designed to handle system and media failures in such a way that data integrity is preserved across all the participating database servers. The two-phase commit protocol performs an automatic recovery if a failure occurs.

Types of failures that automatic recovery handles

The following events can cause the coordinating thread or the participant thread to terminate or hang, thereby requiring automatic recovery:

- System failure of the coordinator
- System failure of a participant
- Network failure
- Termination of the coordinating thread by the administrator
- Termination of the participant thread by the administrator

Administrator's role in automatic recovery

The only role of the administrator in automatic recovery is to bring the coordinator or participant (or both) back online after a system or network failure.

Important: A slow network cannot trigger automatic recovery. None of the recovery mechanisms described here go into effect unless a coordinator system fails, a network fails, or the administrator terminates the coordinating thread.

Automatic-recovery mechanisms for coordinator failure

If the coordinating thread fails, each participant database server must decide whether to initiate automatic recovery before it commits or rolls back the transaction or after it rolls back a transaction. This responsibility is part of the presumed-end optimization. (See "Presumed-end optimization" on page 25-9.)

Automatic-recovery mechanisms for participant failure

Participant recovery occurs whenever a participant thread precommits an item of work that is terminated before the two-phase commit protocol can be completed. The goal of participant recovery is to complete the two-phase commit protocol according to the decision reached by the coordinator.

Participant recovery is driven by either the coordinator or the participant, depending on whether the coordinator decided to commit or to roll back the global transaction.

Important: To support automatic recovery after a subordinate server is shut down or restarted while a cross-server transaction is open, the `sqlhosts` file must include an entry for every database server from which distributed operations might be initiated. During automatic recovery, the name of the coordinator is recovered from the logical logs, and the subordinate server reconnects with the coordinator to complete the transaction. Because the coordinator always identifies itself to the participants using the name that is in the DBSERVERNAME configuration parameter in its own `onconfig` file, the DBSERVERNAME setting of the coordinator must be an Internet protocol connection name known to the participants, but you can also define at least one DBSERVERALIASES setting with the correct connection protocol for connectivity between the coordinator and the subordinate servers. The subordinate server must be able to connect to the coordinator using either the DBSERVERNAME setting or a DBSERVERALIASES setting of the coordinator.

Presumed-end optimization

Presumed-end optimization is a term that describes how the two-phase commit protocol handles the rollback of a transaction.

Rollback is handled in the following manner. When the coordinator determines that the transaction must be rolled back, it sends a message to all the participants to roll back their piece of work. The coordinator does not wait for an acknowledgment of this message, but proceeds to close the transaction and remove it from shared memory. If a participant tries to determine the status of this transaction—that is, find out whether the transaction was committed or rolled back (during participant recovery, for example)—it does not find any transaction status in shared memory. The participant must interpret this as meaning that the transaction was rolled back.

Independent actions

An independent action in the context of two-phase commit is an action that occurs independently of the two-phase commit protocol. Independent actions might or might not be in opposition to the actions that the two-phase commit protocol specifies. If the action is in opposition to the two-phase commit protocol, the action results in an error or a *heuristic decision*. Heuristic decisions can result in an inconsistent database and require manual two-phase commit recovery. Manual recovery is an extremely complicated administrative procedure that you must try to avoid. (For an explanation of the manual-recovery process, see Chapter 26, "Manually recovering from failed two-phase commit," on page 26-1.)

Situations that initiate independent action

Independent action during a two-phase commit protocol is rare, but it can occur in the following situations:

- The participant's piece of work develops into a long-transaction error and is rolled back.
- An administrator stops a participant thread during the postdecision phase of the protocol with **onmode -z**.
- An administrator ends a participant transaction (piece of work) during the postdecision phase of the protocol with **onmode -Z**.
- An administrator ends a global transaction at the coordinator database server with **onmode -z** or **onmode -Z** after the coordinator issued a commit decision and became aware of a participant failure. This action always results in an error, specifically error -716.

Possible results of independent action

As mentioned earlier, not all independent actions are in opposition to the two-phase commit protocol. Independent actions can yield the following three possible results:

- Successful completion of the two-phase commit protocol
- An error condition
- A heuristic decision

If the action is not in opposition to the two-phase protocol, the transaction either commits or rolls back normally. If the action ends the global transaction prematurely, an error condition results. Ending the global transaction at the coordinator is not considered a heuristic decision. If the action is in opposition to the two-phase commit protocol, a heuristic decision results. All these situations are explained in the sections that follow.

Independent actions that allow transactions to complete successfully

Independent actions are not necessarily in opposition to the two-phase commit protocol. For example, if a piece of work at a participant database server is rolled back because it developed into a long transaction, and the coordinator issues a decision to roll back the global transaction, the database remains consistent.

Independent actions that result in an error condition

If you, as administrator at the coordinator database server, run either **onmode -z** (stop the coordinator thread) or **onmode -Z** (stop the global transaction) after the coordinator issues its final *commit* decision, you are removing all knowledge of the transaction from shared memory at the coordinator database server.

This action is not considered a heuristic decision because it does not interfere with the two-phase protocol; it is either acceptable, or it interferes with participant recovery and causes an error.

The action is acceptable any time that all participants are able to commit the transaction without difficulty. In this case, your action to end the transaction forcibly is superfluous. The indication that you ran **onmode -Z** reaches the coordinator only when the coordinator is preparing to terminate the transaction.

In practice, however, you would probably consider running **onmode -z** or **onmode -Z** at the coordinator database server only if you were attempting to hasten the conclusion of a global transaction that has remained open for an unusually long period. In this scenario, the source of the problem is probably a failure at some participant database server. The coordinator has not received acknowledgment that

the participant committed its piece of work, and the coordinator is attempting to establish communication with the participant to investigate.

If you run either **onmode -z** or **onmode -Z** while the coordinator is actively trying to reestablish communication, the coordinating thread obeys your instruction to die, but not before it writes error -716 into the database server message log. The action is considered an error because the two-phase commit protocol was forcibly broken, preventing the coordinator from determining whether the database is consistent.

Stopping a global transaction at a coordinator database server is not considered a heuristic decision, but it can result in an inconsistent database. For example, if the participant eventually comes back online and does not find the global transaction in the coordinator shared memory, it rolls back its piece of work, thereby causing a database inconsistency.

Independent actions that result in heuristic decisions

Some independent actions can develop into heuristic decisions when *both* of the following conditions are true:

* The participant database server already sent a `can commit` message to the coordinator and then rolls back.
* The coordinator's decision is to commit the transaction.

When both conditions are true, the net result is a global transaction that is inconsistently implemented (committed by one or more database servers and rolled back by another). The database becomes inconsistent.

The following two heuristic decisions are possible:

* Heuristic rollback (described in "The heuristic rollback scenario")
* Heuristic end transaction (described in "The heuristic end-transaction scenario" on page 25-14)

After a heuristic rollback or end transaction occurs, you might be required to perform manual recovery, a complex and time-consuming process. you must understand heuristic decisions fully in order to avoid them. Always be wary of running **onmode -z** or **onmode -Z** within the context of two-phase commit.

The heuristic rollback scenario

In a *heuristic rollback*, either the database server or the administrator rolls back a piece of work that has already sent a `can commit` message.

Conditions that result in a heuristic rollback

The following two conditions can cause a heuristic rollback:

* The logical log fills to the point defined by the LTXEHWM configuration parameter. (See the topics about configuration parameters in the *IBM Informix Administrator's Reference*.) The source of the long-transaction condition is a piece of work being performed on behalf of a global transaction.
* An administrator executes **onmode -z** *session_id* to stop a database server thread that is executing a piece of work being performed on behalf of a global transaction.

In either case, if the piece of work has already sent a `can commit` message to its coordinator, the action is considered a heuristic decision.

Condition 1: Logical log fills to a high-watermark:
Under two-phase commit, a participant database server that is waiting for
instructions from the coordinator is blocked from completing its transaction.
Because the transaction remains open, the logical-log files that contain records
associated with this transaction cannot be freed. The result is that the logical log
continues to fill because of the activity of concurrent users.

If the logical log fills to the value of the long-transaction high-watermark
(LTXHWM) while the participant is waiting, the database server directs all
database server threads that own long transactions to begin rolling them back. If a
piece of work that is precommitted is the offending long transaction, the database
server has initiated a heuristic rollback. That is, this database server is rolling back
a precommitted piece of work without the instruction or knowledge of the
coordinator.

Under two-phase commit, the logical-log files that contain records associated with
the piece of work are considered open until an ENDTRANS logical-log record is
written. This type of transaction differs from a transaction involving a single
database server where a rollback actually closes the transaction.

The logical log might continue to fill until the exclusive high-watermark is reached
(LTXEHWM). If this happens, all user threads are suspended except those that are
currently rolling back or currently committing. In the two-phase commit scenario,
the open transaction prevents you from backing up the logical-log files and freeing
space in the logical log. Under these specific circumstances, the logical log can fill
completely. If this happens, the participant database server shuts down, and you
must perform a data restore.

Condition 2: System administrator executes onmode -z:
You, as administrator, can decide to initiate a heuristic rollback of a precommitted
piece of work by running **onmode -z**. You might make this decision because you
want to free the resources that are held by the piece of work. (If you stop the
participant thread by running **onmode -z**, you free all locks and shared-memory
resources that are held by the participant thread even though you do not end the
transaction.)

Results of a heuristic rollback
These topics describe what happens at both the coordinator and participant when a
heuristic rollback occurs and how this process can result in an inconsistent
database:

1. At the participant database server where the rollback occurred, a record is
   placed in the database server logical log (type HEURTX). Locks and resources
   held by the transaction are freed. The participant thread writes the following
   message in the database server message log, indicating that a long-transaction
   condition and rollback occurred:

   ```
   Transaction Completed Abnormally (rollback):
   tx=address flags=0xnn
   ```

2. The coordinator issues postdecision phase instructions to commit the
   transaction.

   The participant thread at the database server where the heuristic rollback
   occurred returns error message -699 to the coordinator as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

   This error message is not returned to the application at this point; it is an
   internal notification to the coordinator. The coordinator waits until all

participants respond to the commit instruction. The coordinator does not determine database consistency until all participants report.

3. The next steps depend on the actions that occur at the other participants. Two situations are possible.

Situation 1: Coordinator issues a commit and all participants report heuristic rollbacks:
The coordinator gathers all responses from participants. If every participant reports a heuristic rollback, the following events occur as a consequence:

1. The coordinator writes the following message to its own database-server message log:

   `Transaction heuristically rolled back.`

2. The coordinator sends a message to all participants to end the transaction.

3. Each participant writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the transaction table.)

4. The coordinator returns error -699 to the application, as follows:

   `-699 Transaction heuristically rolled back.`

5. In this situation, all databases remain consistent.

Situation 2: Coordinator issued a commit; one participant commits and one reports a heuristic rollback:
The coordinator gathers all responses from participants. If at least one participant reports a heuristic rollback and at least one reports an acknowledgment of a commit, the result is called a *mixed-transaction result*. The following events occur as a consequence:

1. The coordinator writes the following message to its own database server message log:

   `Mixed transaction result. (pid=`*nn* `user=`*userid*`)`

   The `pid` value is the user-process identification number of the coordinator process. The `user` value is the user ID associated with the coordinator process. Associated with this message are additional messages that list each of the participant database servers that reported a heuristic rollback. The additional messages take the following form:

   `Participant database server` *dbservername* `heuristically rolled back.`

2. The coordinator sends a message to each participant that heuristically rolled back its piece of work, directing each one to end the transaction.

3. Each participant writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the transaction table.)

4. The coordinator writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)

5. The coordinator returns error -698 to the application, as follows:

   `-698 Inconsistent transaction. Number` *and* `names` *of servers rolled back.*

6. Associated with this error message is the list of participant database servers that reported a heuristic rollback. If many database servers rolled back the transaction, this list might be truncated. The complete list is always included in the message log for the coordinator database server.

In this situation, examine the logical log at each participant database server site and determine whether your database system is consistent. (See "Determine if a transaction was implemented inconsistently" on page 26-1.)

The heuristic end-transaction scenario

A *heuristic end transaction* is an independent action taken by the administrator to roll back a piece of work and remove all information about the transaction from the transaction table. The heuristic end-transaction process is initiated when the administrator executes the **onmode -Z** *address* command.

Whenever you initiate a heuristic end transaction by running **onmode -Z**, you remove critical information required by the database server to support the two-phase commit protocol and its automatic-recovery features. If you run **onmode -Z**, it becomes your responsibility to determine whether your networked database system is consistent.

When to perform a heuristic end transaction

You must run the **onmode -Z** option to initiate a heuristic end transaction in only one, rare, situation. This situation occurs when a piece of work that has been heuristically rolled back remains open, preventing your logical-log files from becoming free. As a result, the logical log is dangerously close to full.

In general, the coordinator issues its commit-or-rollback decision within a reasonable period of time. However, if the coordinator fails and does not return online to end a transaction that was heuristically rolled back at your participant database server, you might face a serious problem.

The problem scenario begins in this way:
1. The participant thread that is executing a piece of work on behalf of a global transaction has sent a `can commit` response to the coordinator.
2. The piece of work waits for instructions from the coordinator.
3. While the piece of work is waiting, the logical log fills past the long-transaction high-watermark.
4. The piece of work that is waiting for instructions is the source of the long transaction. The participant database server directs the executing thread to roll back the piece of work. This action is a heuristic rollback.
5. The participant continues to wait for the coordinator to direct it to end the transaction. The transaction remains open. The logical log continues to fill.

If the coordinator contacts the participant and directs it to end the transaction in a reasonable period of time, no problem develops. The serious problem arises if the heuristic rollback occurs at a participant database server and subsequently the coordinator fails, preventing the coordinator from directing the participant to end the transaction.

As a consequence, the transaction remains open. The open transaction prevents you from backing up logical-log files and freeing space in the logical log. As the logical log continues to fill, it might reach the point specified by the exclusive-access, long-transaction high-watermark (LTXEHWM). If this point is reached, normal processing is suspended. At some point after the high-watermark is reached, you must decide if the open transaction is endangering your logical log. The danger is that if the logical log fills completely, the database server shuts down, and you must perform a data restore.

You must decide whether to end the transaction and protect your system against the possibility of filling the logical log, despite all the problems associated with running **onmode -Z**, or to wait and see if communication with the coordinator can be reestablished in time to end the transaction before the logical log fills.

How to use onmode -Z

The **onmode -Z** *address* command is intended for use only if communication between the coordinator and the participant is broken. To ensure that communication is really broken, the **onmode -Z** command does not run unless the thread that was executing the piece of work has been dead for the amount of time specified by TXTIMEOUT. For more information about this option, see the *IBM Informix Administrator's Reference*.

The *address* parameter is obtained from **onstat -x** output. For more information about the **onstat -x** option, see the *IBM Informix Administrator's Reference*.

Action when the transaction is ended heuristically

When you run **onmode -Z**, you direct the **onmode** utility to remove the participant transaction entry, which is located at the specified address, from the transaction table.

Two records are written in the logical log to document the action. The records are type ROLLBACK and ENDTRANS, or if the transaction was already heuristically rolled back, ENDTRANS only. The following message is written to the participant database server message log:

```
(time_stamp) Transaction Completed Abnormally (endtx): tx=address flags:0xnn user username tty ttyid
```

The coordinator receives an error message from the participant where the **onmode -Z** occurred, in response to its COMMIT instruction. The coordinator queries the participant database server, which no longer has information about the transaction. The lack of a transaction-table entry at the participant database server indicates that the transaction committed. The coordinator assumes that the acknowledgment message was sent from the participant, but somehow it was not received. Because the coordinator does not know that this participant's piece of work did not commit, it does not generate messages indicating that the global transaction was inconsistently implemented. Only the administrator who ran the **onmode -Z** command is aware of the inconsistent implementation.

Monitor a global transaction

Use the **onstat -x** command to track open transactions and determine whether they have been heuristically rolled back.

For example, in the output, an H flag in the **flags** field identifies a heuristic rollback, the G flag identifies a global transaction, the L flag indicates loosely coupled mode, and the T flag indicates tightly coupled mode.

The **curlog** and **logposit** fields provide the exact position of a logical-log record. If a transaction is not rolling back, **curlog** and **logposit** describe the position of the most recently written log record. When a transaction is rolling back, these fields describe the position of the most recently "undone" log record. As the transaction rolls back, the **curlog** and **logposit** values decrease. In a long transaction, the rate at which the **logposit** and **beginlg** values converge can help you estimate how much longer the rollback is going to take.

For more information about and an example of **onstat -x** output, see the *IBM Informix Administrator's Reference*.

You also can use the **onstat -u** and **onstat -k** commands to track transactions and the locks that they hold. For details, see the monitoring transactions topics in your

IBM Informix Performance Guide. For a description of the fields that **onstat -x**
displays, see the *IBM Informix Administrator's Reference*.

On a secondary server, when transaction completion after failover is enabled (by
setting the FAILOVER_TX_TIMEOUT configuration parameter), it is possible that
two global transactions might have the same global transaction identifier: one is a
local temporary global transaction, and the other is the global transaction that
belongs to the recovery thread. A quick way to tell the real global transaction from
the temporary transaction is that the real transaction has a B flag if the transaction
has performed any operations. You can also check the owner of the transaction by
using the **onstat -g ath** command. The temporary global transaction on the
secondary server is deleted after the xa_end() function is called.

The following onstat utility example output illustrates XA transaction support on
both primary and secondary servers in a high-availability cluster environment. The
onstat -x, **onstat -G**, and **onstat -ath** commands are separately documented, but
output from the combined **onstat -xG** command is of special interest for global
transactions. The examples show each state of a redirected transaction.

In the examples, the global transaction shown running on the secondary server is a
temporary transaction. The temporary transaction is used to support the SQL
statements performed on the secondary server (not transactions redirected to the
primary server). The temporary transaction is only shown when a user thread is
actively associated with the global transaction branch.

The following example shows output from the **onstat -xG** command run on a
secondary server after an xa_start() function:

```
Transactions
                                                               est.
address         flags userthread     locks begin_logpos      current logpos    isol    rb_time retrys coord
7000000104d4190 AT--G 7000000104a7b68 0     -                 -                 LC      -       0
7000000104d8bd0 ALB-G 7000000104a5aa8 1     180:0x0           180:0x4eb018      DIRTY   0:00    0

Global Transaction Identifiers
address         flags isol   timeout fID       gtl bql data
7000000104d4190 AT--G COMMIT 0       5067085   15  4   000102030405060708090A0B0C0D0E0F000000
7000000104d8bd0 ALB-G DIRTY  0       5067085   15  4   000102030405060708090A0B0C0D0E0F000000
```

Output from **onstat -g ath** run on the secondary server:

```
Threads:
 tid    tcb             rstcb           prty status              vp-class     name
 317    7000001500902c8 7000000104a7b68 1    cond wait  netnorm   1cpu        sqlexec
 84     7000001403a7dc0 7000000104a5aa8 3    sleeping secs: 1      5cpu        xchg_2.0
```

Output from **onstat -xG** run on the primary server:

```
Transactions
                                                               est.
address         flags userthread     locks begin_logpos      current logpos    isol    rb_time retrys coord
7000000104d9e60 ATB-M 7000000104a8bc8 2     180:0x4ea018      180:0x4eb018      COMMIT  0:00    0

Global Transaction Identifiers
address         flags isol   timeout fID       gtl bql data
7000000104d9e60 AT--M COMMIT 0       5067085   15  4   000102030405060708090A0B0C0D0E0F000000
```

The M flag in the previous example indicates that the global transaction was
started from a secondary server. Global transactions started on the primary server
display a G flag. The M flag is displayed only on the primary server.

Output from the **onstat -g ath | grep 7000000104a8bc8** command:

```
 196    70000013012d3a8 7000000104a8bc8 1    sleeping secs: 1      4cpu        proxyTh
```

The following example shows output from the **onstat -xG** command run on
secondary server after the xa_end() function:

```
Transactions
                                                                    est.
address         flags userthread       locks begin_logpos    current logpos   isol    rb_time retrys coord
7000000104d8bd0 ALB-G 7000000104a5aa8 1      180:0x0         180:0x4ee018     DIRTY   0:00    0

Global Transaction Identifiers
address         flags  isol    timeout fID      gtl bql data
7000000104d8bd0 ALB-G  DIRTY   0       5067085  15  4   000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the primary server:

```
Transactions
                                                                    est.
address         flags userthread       locks begin_logpos    current logpos   isol    rb_time retrys coord
7000000104d9e60 -TB-M 0               2     180:0x4ea018    180:0x4ee018     COMMIT  0:00    0


Global Transaction Identifiers
address         flags  isol    timeout fID      gtl bql data
7000000104d9e60 -T--M  COMMIT  -1      5067085  15  4   000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the secondary server after running the xa_prepare() function:

```
Transactions
                                                                    est.
address         flags userthread       locks begin_logpos    current logpos   isol    rb_time retrys coord
7000000104d8bd0 ALX-G 7000000104a5aa8 1      180:0x0         180:0x4ef018     DIRTY   0:00    0

Global Transaction Identifiers
address         flags  isol    timeout fID      gtl bql data
7000000104d8bd0 ALX-G  DIRTY   0       5067085  15  4   000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the primary server:

```
Transactions
                                                                    est.
address         flags userthread       locks begin_logpos    current logpos   isol    rb_time retrys coord
7000000104d9e60 -TX-M 0               2     180:0x4ea018    180:0x4ef018     COMMIT  0:00    0

Global Transaction Identifiers
address         flags  isol    timeout fID      gtl bql data
7000000104d9e60 -TX-M  COMMIT  -1      5067085  15  4   0
```

Two-phase commit protocol errors

The following two-phase commit protocol errors require special attention from the administrator.

Error number
   Description

-698   If you receive error -698, a heuristic rollback has occurred and has caused an inconsistently implemented transaction. The circumstances leading up to this event are described in "Results of a heuristic rollback" on page 25-12. For an explanation of how the inconsistent transaction developed and to learn the options available to you, see this information.

-699   If you receive error -699, a heuristic rollback has occurred. The circumstances leading up to this event are described in "Results of a heuristic rollback" on page 25-12. For an explanation of how the inconsistent transaction developed, see this information.

-716   If you receive error -716, the coordinating thread has been terminated by administrator action after it issued its final decision. This scenario is described under "Independent actions that result in an error condition" on page 25-10.

Two-phase commit and logical-log records

The database server uses logical-log records to implement the two-phase commit protocol. You can use these logical-log records to detect heuristic decisions and, if necessary, to help you perform a manual recovery. (See Chapter 26, "Manually recovering from failed two-phase commit," on page 26-1.)

The following logical-log records are involved in distributed transactions:
- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

For information about these logical-log records, see the chapter on interpreting the logical log in the *IBM Informix Administrator's Reference*.

This section examines the sequence of logical-log records that are written during the following database server scenarios:

Logical-log records when the transaction commits

The following figure illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.

Start protocol

Coordinator:
Writes log record: BEGPREP.
Sends message: *precommit.*

All participants:
Write log record: TABLOCKS.
Write and flush log record: PREPARE.
Send message: *can commit.*

Coordinator:
Writes log record: COMMIT.
Flushes logical-log buffer.
Sends message: *commit*

All participants:
Writes log record: COMMIT.
Flushes logical-log buffer.
Send message: *committed.*

Coordinator:
Writes log record: ENDTRANS.

End protocol

Figure 25-2. Logical-log records written during a committed transaction

Some of the logical-log records must be flushed from the logical-log buffer immediately; for others, flushing is not critical.

The coordinator's commit-work record (COMMIT record) contains all information required to initiate the two-phase commit protocol. It also serves as the starting point for automatic recovery in the event of a failure on the coordinator's host computer. Because this record is critical to recovery, it is not allowed to remain in the logical-log buffer. The coordinator must immediately flush the COMMIT logical-log record.

The participants in the preceding figure must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. As part of recovery, the participant might query the coordinator to learn the final disposition of this transaction.

Flushing the participant's COMMIT record ensures that, if the participant's host computer fails, the participant has a record of what action it took regarding the transaction. To understand why this information is crucial, consider the situation in which a participant crashes after the PREPARE record is written but before the COMMIT record flushes. After fast recovery, the PREPARE record is restored, but

the COMMIT record is lost (because it was in the logical-log buffer at the time of the failure). The existence of the PREPARE record would initiate a query to the coordinator about the transaction. However, the coordinator would know nothing of the transaction, because it ended the transaction after it received the participant's acknowledgment that the commit occurred. In this situation, the participant would interpret the lack of information as a final direction to roll back the transaction. The two-phase commit protocol requires the participant's COMMIT record to be flushed immediately to prevent this kind of misunderstanding.

Logical-log records written during a heuristic rollback

The following figure illustrates the sequence in which the database server writes the logical-log records during a heuristic rollback. Because a heuristic rollback only occurs after the participant sends a message that it can commit and the coordinator sends a message to commit, the first phase of this protocol is the same as that shown in Figure 25-2 on page 25-19. When a heuristic rollback occurs, the rollback is assumed to be the consequence of a long-transaction condition that occurs at the Participant 1 (P1) database server. The end result is a transaction that is inconsistently implemented. See "The heuristic rollback scenario" on page 25-11.
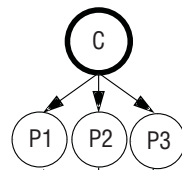
```
        C
      /  |  \
    P1  P2  P3
```

Coordinator:
Writes log record: BEGPREP.
Sends message: *precommit.*

All Participants:
Write log record: TABLOCKS.
Write log record: PREPARE.
Flush logical log.
Send message: *commit ok.*

Within P1 participant's environment:
The database server detects long
transaction condition. Rollback starts.
Writes log record: HEURTX.
Writes log record: ROLLBACK.
Message written in message log.

```
        C
      /  |  \
    P1  P2  P3
```
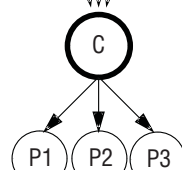
Coordinator:
Writes log record: COMMIT.
Flushes log record.
Sends message: *commit.*

Participant 1:
Sends message: *Transaction heuristically rolled back. Cannot commit.*
Participants 2 and 3:
Write and flush log record: COMMIT.
Send message: *committed.*

```
        C
      /
    P1  P2  P3
```
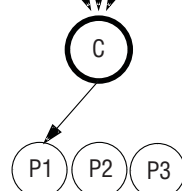
Coordinator:
Writes message in message log (-698).
Sends message to Participant 1: *end-transaction.*

Participant 1:
Writes log record: ENDTRANS.
Sends message: *Transaction ended.*

```
        C
```

Coordinator:
Writes log record: ENDTRANS.
Returns error message to user: Error -698.

End protocol

Figure 25-3. Logical-log records written during a heuristic rollback

Logical-log records written after a heuristic end transaction

The following figure illustrates the writing sequence of the logical-log records
during a heuristic end transaction. The event is always the result of a database
server administrator ending a transaction (see information about the **onmode**
utility in the *IBM Informix Administrator's Reference*) at a participant database server
after the participant has sent a `can commit` message. In the following figure, the
heuristic end transaction is assumed to have occurred at the Participant 1 (P1)
database server. The result is an inconsistently implemented transaction. See "The
heuristic end-transaction scenario" on page 25-14.

Start protocol
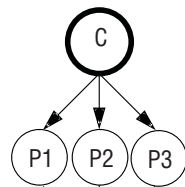
Coordinator:
Writes log record: BEGPREP.
Sends message: *precommit.*

All participants:
Write log record: TABLOCKS.
Write and flush log record: PREPARE.
Send message: *can commit.*
P1 participant's environment:
Transaction is killed.
Writes log record: ROLLBACK.
Writes log record: ENDTRANS.
Message is written in the database server message log.
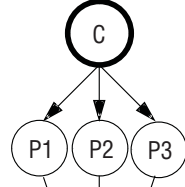
Coordinator:
Writes log record: COMMIT.
Flushes logical-log buffer.
Sends message: *commit.*

Participant 1:
Returns error message.

Participants 2 and 3:
Write log record: COMMIT.
Flush logical-log buffer.
Send message: *committed.*
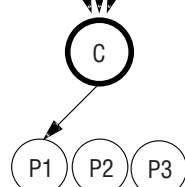
Coordinator:
Receives error message from P1.
Establishes new connection to P1 and sends *TX Inquire* message to P1.

Participant 1:
Sends *transaction status unknown* message back to the coordinator.
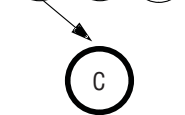
Coordinator:
Assumes *unknown status* means *committed.*
Writes log record: ENDTRANS.

End protocol

Figure 25-4. Logical-log records written during a heuristic end transaction

Configuration parameters used in two-phase commits

The following two configuration-file parameters are specific to distributed environments:

- DEADLOCK_TIMEOUT
- TXTIMEOUT

Although both parameters specify timeout periods, the two are independent. For more information about these configuration parameters, see the *IBM Informix Administrator's Reference*.

Function of the DEADLOCK_TIMEOUT parameter

If a distributed transaction is forced to wait longer than the number of seconds specified by DEADLOCK_TIMEOUT for a shared-memory resource, the thread that owns the transaction assumes that a multiserver deadlock exists. The following error message is returned:

```
-154 ISAM error: deadlock timeout expired - Possible deadlock.
```

The default value of DEADLOCK_TIMEOUT is 60 seconds. Adjust this value carefully. If you set it too low, individual database servers end transactions that are not deadlocks. If you set it too high, multiserver deadlocks might reduce concurrency.

Function of the TXTIMEOUT parameter

The TXTIMEOUT configuration parameter is specific to the two-phase commit protocol. It is used only if communication between a transaction coordinator and participant has been interrupted and must be reestablished.

The TXTIMEOUT parameter specifies a period of time that a participant database server waits to receive a commit instruction from a coordinator database server during a distributed transaction. If the period of time specified by TXTIMEOUT elapses, the participant database server checks the status of the transaction to determine if the participant must initiate automatic participant recovery.

TXTIMEOUT is specified in seconds. The default value is 300 (five minutes). The optimal value for this parameter varies, depending on your specific environment and application. Before you modify this parameter, read the explanation "How the two-phase commit protocol handles failures" on page 25-8.

Heterogeneous commit protocol

Used in the context of IBM Informix database servers, the term heterogeneous environment is a group of database servers in which at least one of the database servers is not an IBM Informix database server. Heterogeneous commit ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment.

Unlike the two-phase commit protocol, the heterogeneous commit protocol supports the participation of a non-Informix participant. The non-Informix participant, called a gateway participant, must communicate with the coordinator through an IBM Informix gateway.

The database server uses heterogeneous commit protocol when the following criteria are met:
- Heterogeneous commit is enabled. (That is, the HETERO_COMMIT configuration parameter is set to 1.)
- The coordinator of the commit is a Version 7.2 or later IBM Informix.
- The non-Informix participant communicates with the IBM Informix database server through an IBM Informix gateway.
- At most, one non-Informix participant performs an update within a single transaction.

The following figure illustrates this scenario.
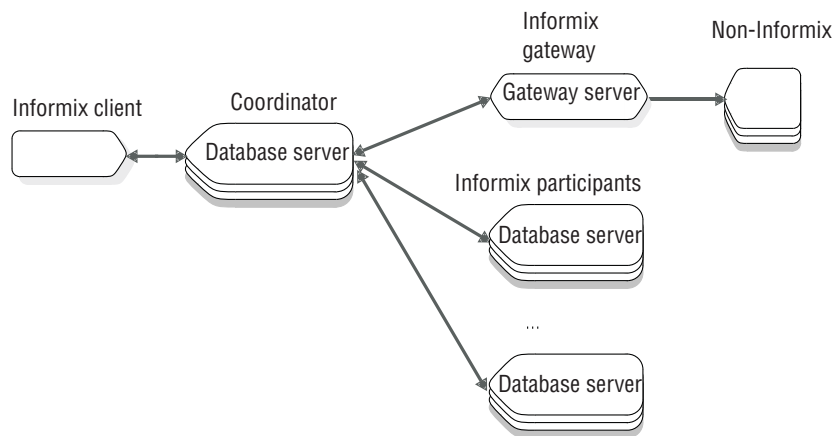
Figure 25-5. Configuration that requires heterogeneous commit for distributed transactions

Gateways that can participate in a heterogeneous commit transaction

A gateway acts as a bridge between an IBM Informix application (in this case, a database server) and a non-Informix database server. You can use a gateway to use an IBM Informix application to access and modify data that is stored in a database that is not IBM Informix.

The following table lists the gateways and corresponding database servers that can participate in a transaction in which the database server uses the heterogeneous commit protocol.

Table 25-1. Gateways and corresponding database servers/heterogeneous commit transaction

Gateway	Database servers
IBM Informix Enterprise Gateway with DRDA	IBM DB2®, OS/400®, SQL/DS
IBM Informix Enterprise Gateway for EDA/SQL	EDA/SQL
IBM Informix Enterprise Gateway Manager	Any database server with ODBC connectivity

Enable and disable of heterogeneous commit

Use a text editor to change the HETERO_COMMIT configuration parameter which enables or disables heterogeneous commit: The change takes effect when you shut down and restart the database server.

Use a text editor or ISA to change the HETERO_COMMIT configuration parameter which enables or disables heterogeneous commit: The change takes effect when you shut down and restart the database server.

When you set HETERO_COMMIT to 1, the transaction coordinator checks for distributed transactions that require the use of heterogeneous commit. When the coordinator detects such a transaction, it automatically executes the heterogeneous commit protocol.

If you set HETERO_COMMIT to 0 or any number other than 1, the transaction coordinator disables the heterogeneous commit protocol. The following table summarizes which protocol the transaction coordinator uses, heterogeneous commit or two-phase commit, to ensure the integrity of a distributed transaction.

HETERO_COMMIT setting	Gateway participant updated	Database server protocol
Disabled	No	Two-phase commit
Disabled	Yes	Two-phase commit
Enabled	No	Two-phase commit
Enabled	Yes	Heterogeneous commit

How heterogeneous commit works

The heterogeneous commit protocol is a modified version of the standard two-phase commit protocol. The postdecision phase in the heterogeneous commit protocol is identical to the postdecision phases in the two-phase commit protocol. The precommit phase contains a minor modification, and a new phase, called the gateway commit phase, is added to the heterogeneous commit protocol.

The following topics explain the modification to the precommit phase and the gateway commit phase. For a detailed explanation of the postdecision phases, see "Postdecision phase" on page 25-8.

Precommit phase

The coordinator directs each update participant (except the gateway participant) to prepare to commit the transaction.

If the updates satisfy all deferred constraints, all participants (except the gateway participant) return messages to the coordinator indicating that they can commit their piece of work.

Gateway commit phase

If all participants successfully return a message indicating that they are prepared to commit, the coordinator sends a commit message to the gateway. The gateway in turn sends a response to the coordinator indicating whether the gateway committed its piece of the transaction. If the gateway commits the transaction, the coordinator decides to commit the entire transaction. The following figure illustrates this process.

Coordinator:
Sends a *commit* message to the gateway participant.

Gateway

Gateway Participant:
Commits the work. Returns message: *committed.*

Gateway

Coordinator:
Receives gateway *committed* message.

End gateway commit phase
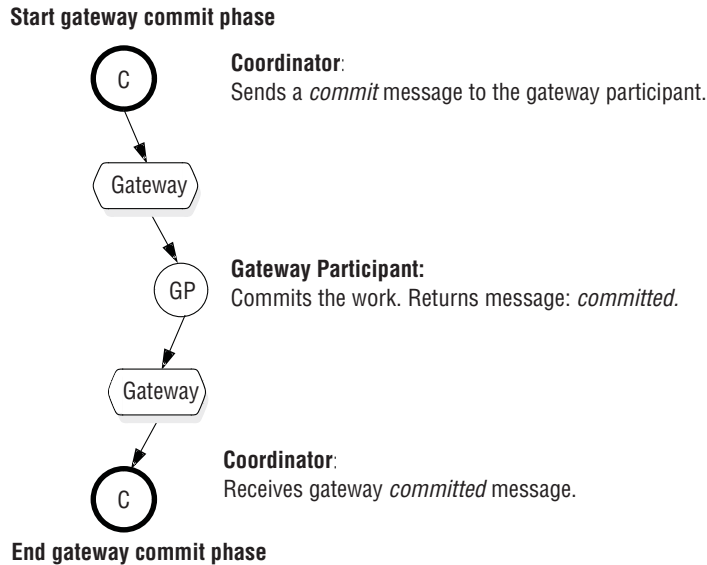
Figure 25-6. Heterogeneous commit phase that results in a committed transaction

If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as the previous figure illustrates.

Heterogeneous commit optimization

The database server optimizes the heterogeneous commit protocol when the only participant that receives an update is a non-Informix database. In this case, the coordinator sends a single commit message to all participants without invoking the heterogeneous commit protocol.

Implications of a failed heterogeneous commit

At any time during a distributed transaction that the database server processes using heterogeneous commit, the coordinator or any number of participants can fail. The database server handles these failures in the same way as in the two-phase commit protocol, except in certain instances. The following topics examine these special instances in detail.

Database server coordinator failure

The consistency of data after a coordinator failure depends on the point in the heterogeneous commit process at which the coordinator fails. If the coordinator fails before sending the commit message to the gateway, the entire transaction is stopped upon recovery, as is the case with two-phase commit.

If the coordinator fails after it writes the commit log record, the entire transaction is committed successfully upon recovery, as is the case with two-phase commit.

If the coordinator fails after it sends the commit message to the gateway but before it writes the commit log record, the remote IBM Informix database server sites in the transaction are stopped upon recovery. This can result in inconsistencies if the gateway received the commit message and committed the transaction.

The following table summarizes these scenarios.

Point of database server coordinator failure	Expected result
After the coordinator writes the PREPARE log record and before the gateway commit phase	Data consistency is maintained.
After the coordinator sends a commit message to the gateway but before it receives a reply	Data is probably inconsistent. No indication of probable data inconsistency from the coordinator.
After gateway commit phase but before the coordinator writes a COMMIT record to the logical log	Data consistency is lost. No indication of data inconsistency from the coordinator.

Participant failure

Whenever a participant in a distributed transaction that uses the heterogeneous protocol fails, the coordinator sends the following error message:

```
-441    Possible inconsistent data at the target DBMS name due to an
aborted commit.
```

In addition, the database server sends the following message to the message log:

```
Data source accessed using gateway name might be in an inconsistent state.
```

A participant failure is not limited to the failure of a database server or gateway. In addition, a failure of the communication link between the coordinator and the gateway is considered a gateway failure. The gateway terminates if a link failure occurs. The gateway must terminate because it does not maintain a transaction log and therefore cannot reestablish a connection with the coordinator and resume the transaction. Because of this restriction, some scenarios exist in which a gateway failure might leave data in an inconsistent state. The following table summarizes these scenarios.

Point of participant failure	Expected result
After participant receives `commit transaction` message from coordinator, but before participant performs commit	Data consistency is maintained.
After participant receives `commit transaction` message from coordinator and commits the transaction, but before the participant replies to coordinator	Data is inconsistent.
After participant commits the transaction and sends a reply to coordinator	If the communications link fails before the coordinator receives the reply, then data is inconsistent. If the coordinator receives the reply, then data is consistent (provided the coordinator does not fail before writing the COMMIT record).

The recovery procedure that the database server follows when a participant fails is identical to the procedure that is followed in two-phase commit. For more information about this procedure, see "How the two-phase commit protocol handles failures" on page 25-8.

Interpretation of heterogeneous commit error messages

When the database server fails to process a distributed transaction using heterogeneous commit, it returns one of the two error messages that are explained in the following topics.

Application attempts to update multiple gateway participants:
If your client application attempts to update data at more than one gateway
participant when HETERO_COMMIT is set to 1, the coordinator returns the
following error message:

```
-440 Cannot update more than one non-Informix DBMS within a transaction.
```

If you receive this error message, rewrite the offending application so that it
updates at most one gateway participant in a single distributed transaction.

Failed attempt to commit distributed transaction using heterogeneous commit:
The database server can fail to commit a distributed transaction while it is using
the heterogeneous protocol for one or more of the following reasons:

- Communication error
- Site failure
- Gateway failure
- Other unknown error

When such a failure occurs, the coordinator returns the following message:

```
-441    Possible inconsistent data at the target DBMS name due to
an aborted commit.
```

After the database server sends this message, it rolls back all update sites that are
participating in the transaction, with the possible exception of the work done at the
site of the gateway participant. The gateway participant might have committed its
updates if the failure occurred after the gateway participant processed the commit
message. If the gateway participant committed the updates, you must manually
roll back these updates.

Chapter 26. Manually recovering from failed two-phase commit

Distributed transactions follow the two-phase commit protocol. Certain actions occur independently of the two-phase commit protocol and cause the transaction to be inconsistently implemented. (See "Independent actions" on page 25-9.) In these situations, it might be necessary to recover manually from the transaction.

Determine if manual recovery is required

The following topics outline the steps in the procedure to determine database consistency and to correct the situation if required.

Each of these steps is described in the following topics.

Determine if a transaction was implemented inconsistently

Your first task is to determine whether the transaction was implemented inconsistently as a result of an independent action.

Global transaction ended prematurely

If you ran an **onmode -z** command to end the global transaction on the coordinator, the transaction might be inconsistently implemented. (For an explanation of how this situation can arise, see "Independent actions that result in an error condition" on page 25-10.) You can check for an inconsistent transaction by first examining the database server message log for the coordinator. Look for the following error message:

```
-716 Possible inconsistent transaction.
Unknown servers are server-name-list.
```

This message lists all the database servers that were participants. Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic end transaction

If you ran an **onmode -Z** *address* command to end a piece of work performed by a participant, and the coordinator decided to commit the transaction, the transaction is implemented inconsistently. (For a description of this scenario, see "The heuristic end-transaction scenario" on page 25-14.) Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic rollback

You can determine the specific database server participants affected by a heuristic decision to roll back a transaction in the following ways:

- Examine the return code from the COMMIT WORK statement in the application.

  The following message indicates that one of the participants performed a heuristic rollback:

  ```
  -698 Inconsistent transaction. Number and names of servers rolled back.
  ```

- Examine the messages in the database server message-log file.

If a database inconsistency is possible because of a heuristic decision at a participating database server, the following message is in the database server message-log file of the coordinator:

```
Mixed transaction result. (pid=nn user=user_id)
```

This message is written whenever error -698 is returned. Associated with this message is a list of the participant database servers where the transaction was rolled back. This is the complete list. The list that is created with the -698 error message might be truncated if many participants rolled back the transaction.

- Examine the logical log for each participant.

  If at least one participant rolls back its piece of work and one participant commits its piece of work, the transaction is implemented incorrectly.

Determine if the distributed database contains inconsistent data

If you determine that a transaction was inconsistently implemented, you must determine what this situation means for your distributed database system. Specifically, you must determine if data integrity has been affected.

A transaction that is inconsistently implemented causes problems whenever the piece of work rolled back by one participant is dependent on a piece of work that was updated by another participant. It is impossible to define these dependencies with SQL because distributed transactions do not support constraints that reference data at multiple database servers. The pieces of work are independent (no dependencies exist) only if the data could have been updated in two independent transactions. Otherwise, the pieces of work are considered to be dependent.

Before you proceed, consider the transaction that caused the error. Are the pieces of data that were updated and rolled back dependent on one another? Multiple updates might be included in a single transaction for reasons other than maintaining data integrity. For example, three possible reasons are as follows:

- Reduced transaction overhead
- Simplified coding
- Programmer preference

Verify also that every participant database server that is assumed to have committed the transaction actually modified data. A read-only database server might be listed as a participant that committed a transaction.

If an inconsistent transaction does not lead to a violation of data integrity, you can quit the procedure at this point.

Obtaining information from the logical log

To determine if data integrity has been affected by an inconsistently implemented global transaction, you must reconstruct the global transaction and determine which parts of the transaction were committed and which were rolled back. Use the **onlog** utility to obtain the necessary information. The procedure is as follows:

1. Reconstruct the transaction at the participant that contains the HEURTX record.

   a. A participant database server logical log is the starting point for your information search. Each record in the log has a local transaction identification number (**xid**). Obtain the **xid** of the HEURTX record.

   b. Use the local **xid** to locate all associated log records that rolled back as part of this piece of work.

2. Determine which database server acted as coordinator for the global transaction.

    a. Look for the PREPARE record on the participant that contains the same local **xid**. The PREPARE record marks the start of the two-phase commit protocol for the participant.

    b. Use the **onlog -l** option to obtain the long output of the PREPARE record. This record contains the global transaction identifier (GTRID) and the name of the coordinating database server. For information about GTRID, see "Obtain the global transaction identifier."

3. Obtain a list of the other participants from the coordinator log.

    a. Examine the log records on the coordinator database server. Find the BEGPREP record.

    b. Examine the long output for the BEGPREP record. If the first 32 bytes of the GTRID in this record match the GTRID of the participant, the BEGPREP record is part of the same global transaction. Note the participants displayed in the ASCII part of the BEGPREP long output.

4. Reconstruct the transaction at each participant.

    a. At each participant database server, read the logical log to find the PREPARE record that contains the GTRID associated with this transaction and obtain the local **xid** for the piece of work performed by this participant.

    b. At each participant database server, use the local **xid** to locate all logical-log records associated with this transaction (committed or rolled back).

After you follow this procedure, you know what all the participants for the transaction were, which pieces of work were assigned to each participant, and whether each piece of work was rolled back or committed. From this information, you can determine if the independent action affected data integrity.

Obtain the global transaction identifier

When a global transaction starts, it receives a unique identification number called a global transaction identifier (GTRID). The GTRID includes the name of the coordinator. The GTRID is written to the BEGPREP logical-log record of the coordinator and the PREPARE logical-log record of each participant.

To see the GTRID, use the **onlog -l** option. The GTRID is offset 20 bytes into the data portion of the record and is 144 bytes long. The following example shows the **onlog -l** output for a BEGPREP record. The coordinator is **chrisw**.

```
4a064    188  BEGPREP  4         0  4a038         0    1
         000000bc 00000043 00000004 0004a038 .......C .......8
         00087ef0 00000002 63687269 73770000 ..~..... chrisw..
         00000000 00000000 00000000 00087eeb ........ ......~.
         00006b16 00000000 00000000 00000000 ..k..... ........
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000000 00000000 00000000
         00000000 00000001 6a756469 74685f73 ........ judith_s
         6f630000 736f6374 63700000          oc..soct cp..
```

The first 32 bytes of the GTRID are identical for the BEGPREP record on the coordinator and the PREPARE records on participants, which are part of the same global transaction. For example, compare the GTRID for the PREPARE record in the following example with that of the BEGPREP record in the previous example.

```
c7064    184  PREPARE  4       0  c7038    chrisw
         000000b8 00000044 00000004 000c7038  .......D ......p8
         00005cd6 00000002 63687269 73770000  ...... chrisw..
         00000000 00000000 00000069 00087eeb  ........ ...i..~.
         00006b16 00000000 00000010 00ba5a10  ..k..... ......Z.
         00000002 00ba3a0c 00000006 00000000  ......:. ........
         00ba5a10 00ba5a1c 00000000 00000000  ..Z...Z. ........
         00ba3a0e 00254554 00ba2090 00000001  ..:..%ET .. .....
         00000000 00ab8148 0005fd70 00ab8148  .......H ...p...H
         0005fe34 0000003c 00000000 00000000  ...4...< ........
         00000000 00ab80cc 00000000 00ab80c4  ........ ........
         00ba002f 63687269 73770000 00120018  .../chrisw......
         00120018 00ba0000                    ........
```

Decide if action is needed to correct the situation

If an inconsistent transaction creates an inconsistent database, the following three options are available to you:

- Leave the networked database in its inconsistent state.
- Remove the effects of the transaction wherever it was committed, thereby rolling back the entire transaction.
- Reapply the effects of the transaction wherever it was rolled back, thereby committing the transaction.

You can leave the database in its inconsistent state if the transaction does not significantly affect database data. You might encounter this situation if the application that is performing the transaction can continue as it is, and you decide that the price (in time and effort) of returning the database to a consistent state by either removing the effects or reapplying the transaction is too high.

You are not required to make this decision immediately. You can use the methods described in the following paragraphs to determine what data the transaction was updating and which records are affected.

As you make your decision, consider that no automatic process or utility can perform a rollback of a committed transaction or can commit part of a transaction that has been rolled back. The following paragraphs describe how to look through the database server message log and the logical log to locate affected records. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you must determine whether to roll back or to commit the transaction. You must also program the compensating transaction that performs the rollback or the commit.

Example of manual recovery

This example illustrates the kind of work that is involved in manual recovery. The following SQL statements were executed by user **nhowe**. Error -698 was returned.

```
dbaccess
CREATE DATABASE tmp WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
CREATE DATABASE tmp@apex WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
DATABASE tmp;
BEGIN WORK;
```

```
INSERT INTO t VALUES (2);
INSERT INTO tmp@apex:t VALUES (2);
COMMIT WORK;
### return code -698
```

The following excerpt is taken from the logical log at the current database server:

addr	len	type	xid	id	link		
.....							
17018	16	CKPOINT	0	0	13018	0	
18018	20	BEGIN	2	1	0	08/27/91 10:56:57	
3482	nhowe						
1802c	32	HINSERT	2	0	18018	1000018	102
4							
1804c	40	CKPOINT	0	0	17018	1	

begin	xid	id	addr	user
1	2	1	1802c	nhowe

addr	len	type	xid	id	link		
19018	72	BEGPREP	2	0	1802c	6d69	1
19060	16	COMMIT	2	0	19018	08/27/91 11:01:38	
1a018	16	ENDTRANS	2	0	19060	580543	

The following excerpt is taken from the logical log at the database server **apex**:

addr	len	type	xid	id	link		
.....							
16018	20	BEGIN	2		1	0	08/27/91
10:57:07	3483	pault					
1602c	32	HINSERT	2	0	16018	1000018	102
4							
1604c	68	PREPARE	2	0	1602c	eh	
17018	16	HEURTX	2	0	1604c	1	
17028	12	CLR	2	0	1602c		
17034	16	ROLLBACK	2	0	17018	08/27/91 11:01:22	
17044	40	CKPOINT	0	0	15018	1	

begin	xid	id	addr	user
1	2	1	17034	--------

addr	len	type	xid	id	link	
18018	16	ENDTRANS	2	0	17034	8806c3
....						

First, you would try to match the transactions in the current database server log
with the transactions in the **apex** database server log. The BEGPREP and PREPARE
log records each contain the GTRID. You can extract the GTRID by using **onlog -l**
and looking at the data portion of the BEGPREP and PREPARE log records. The
GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple,
though less precise, approach is to look at the time of the COMMIT or ROLLBACK
records. The times must be close, although there is a slight delay because of the
time taken to transmit the commit (or rollback) message from the coordinator to
the participant. (This second approach lacks precision because concurrent

transactions can commit at the same time although concurrent transactions from one coordinator would probably not commit at the same time.)

To correct this sample situation

1. Find all records that were updated.
2. Identify their type (insert, delete, update) using **onlog** and the table of record types.
3. Use the **onlog -l** output for each record to obtain the local **xid**, the tblspace number, and the rowid.
4. Map the tblspace number to a table name by comparing the tblspace number to the value in the **partnum** column of the **systables** system catalog table.
5. Using your knowledge of the application, determine what action is required to correct the situation.

In this example, the time stamps on the COMMIT and ROLLBACK records in the different logs are close. No other active transactions introduce the possibility of another concurrent commit or rollback. In this case, an insert (HINSERT) of assigned rowid 102 hex (258 decimal) was committed on the current database server. Therefore, the compensating transaction is as follows:

```
DELETE FROM t WHERE rowid = 258
```

Part 7. Overview of automatic monitoring and corrective actions

You can use the SQL administration API, the Scheduler, and drill-down queries to manage automatic maintenance, monitoring, and administrative tasks.

These components of IBM Informix simplify the collection of information and maintenance of the server in complex systems.

SQL administration API

> The SQL administration API performs remote administration through SQL functions. Because SQL administration API operations occur entirely in SQL, these functions can be used in client tools to administer the database server.

Scheduler

> The Scheduler is a set of tasks that execute SQL statements at predefined times or as determined internally by the server. The SQL statements can either collect information or monitor and adjust the server.

Query Drill-Down

> Query drill-down provides statistical information about recently executed SQL statements to track the performance of individual SQL statements and analyze statement history.

You can use the SQL administration API and the Scheduler on the primary server of an HDR pair of servers.

Each of these tools requires additional disk space to store information.

You can also use a PHP-based web browser administration tool, the OpenAdmin Tool (OAT) for Informix (OAT), to administer multiple database server instances from a single location. Some tasks that you can perform with OAT include:

- Defining and managing automated tasks through the SQL administration API and the Scheduler
- Creating and displaying performance histograms for analysis and tuning of SQL statements

Related concepts:

Chapter 27, "The Scheduler," on page 27-1

Chapter 29, "Query drill-down," on page 29-1

Related tasks:

"Viewing SQL administration API history" on page 28-2

Related reference:

Chapter 28, "Remote administration with the SQL administration API," on page 28-1

Chapter 27. The Scheduler

You can use the Scheduler to create jobs to run administrative tasks or collect information at predictable times. The Scheduler uses SQL statements instead of operating system job scheduling tools.

The Scheduler is controlled by a set of tables in the **sysadmin** database.

The Scheduler has four different job types that you can choose from:

Task   Runs an action at a specific time and frequency.

Sensor
> Runs an action at a specific time and frequency to collect data, create a results table, store the data in the results table, and purge old data after a specified time.

Startup task
> A task that runs only when the server moves from quiescent mode to online mode.

Startup sensor
> A sensor that runs only when the server moves from quiescent mode to online mode.

The action of a task or sensor can be one or more SQL statements, user-defined routines, or stored procedures.

In addition to defining an action for a task or sensor, you can also use the Scheduler to:

- Associate tasks and sensors into functional groups
- Track the execution time and return value each time a task or sensor is run
- Define alerts with varying severity
- Define thresholds to control when tasks or sensors are run

The Scheduler contains built-in tasks and sensors that run automatically. You can modify the built-in tasks and sensors and define your own tasks and sensors.

Disk space requirements

The Scheduler tables and sensor results tables can use significant amounts of disk space.

You can use the following formula to estimate the disk usage for one sensor:

Number of rows collected * size of the row collected * the frequency of data collection per day * the retention period

Repeat this estimate for all sensors and you can determine a close estimate of the space required.

You can reduce the amount of data stored by decreasing the frequency of data collection or shortening the retention period by updating the **ph_task** table.

You can move the **sysadmin** database to a different dbspace by using the SQL administration API, however, all existing data in the database will be lost.

For more information about the **sysadmin** database, see the *IBM Informix Administrator's Reference*.

Related concepts:

Part 7, "Overview of automatic monitoring and corrective actions"

Related tasks:

"Automatically optimizing data storage" on page 9-56

Related reference:

➪ The Scheduler Tables (Administrator's Reference)

Scheduler tables

The Scheduler tables are located in the **sysadmin** database and contain information about tasks and sensors.

The **sysadmin** database contains the Scheduler tables listed in the following table. The **ph_task** table has a direct relationship with each of the other tables.

Table 27-1. Scheduler tables

Table	Description
ph_alert	Contains a list of errors, warnings, or information messages associated with tasks that must be monitored. The **ph_alert** table contains built-in alerts that the database server uses automatically. You can add your own alerts.
ph_group	Contains a list of group names. Each task and sensor is a member of a group. The **ph_group** table contains built-in groups that the database server uses. You can add your own groups.
ph_run	Contains information about how and when each task and sensor was run.
ph_task	Lists tasks and sensors and contains information about how and when the database server runs them. The **ph_task** table contains built-in tasks and sensors that the database server uses automatically. You can add your own tasks and sensors.
ph_threshold	Contains a list of thresholds that are associated with tasks or sensors. If a threshold is met, the associated task can perform an action, such as inserting an alert in the **ph_alert** table. The **ph_threshold** table contains built-in thresholds that the database server uses. You can add your own thresholds.
results	Multiple tables that contain historical data collected by sensors. The structure of these tables is determined by the CREATE TABLE statement in the sensor definition in the **ph_task** table.

For details about these tables, see the *IBM Informix Administrator's Reference*.

Related reference:

⇨ The Scheduler Tables (Administrator's Reference)

Built-in tasks and sensors

The Scheduler contains built-in tasks and sensors that run automatically.

The following table shows the built-in Scheduler tasks and sensors. Sensors have results tables to store the information they collect, and retention periods to determine how long that information is stored. You can change task and sensor properties, for example, the frequency, by updating the **ph_task** table. Some tasks are triggered by thresholds. You can change thresholds by updating the **ph_threshold** table. You can disable a task or sensor by changing the value of the **tk_enable** column in the **ph_task** table to f.

You can determine how long tasks take by querying the **run_duration** column in the **ph_task** table.

Table 27-2. Built-in tasks and sensors

Task or sensor	Description	Results table	Frequency	Retention
Alert Cleanup	This task removes all alert entries from the **ph_alert** table that are older than the threshold of 15 days. The threshold is named ALERT HISTORY RETENTION in the **ph_threshold** table.		Once a day	
auto_crsd	This task compresses, shrinks, repacks, and defragments tables and fragments.

By default, this task is disabled. You must enable it by updating the **ph_task** table.

Each of the operations has 2 rows in the **ph_threshold** table: one to control whether it is enabled and one to control its threshold.

For more information, see "Automatically optimizing data storage" on page 9-56. | | Once a week | |
| autoreg exe | This task registers database extensions when they are first used. | | As necessary | |

Table 27-2. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
autoreg migrate-console	Internal. This task checks every database with a logging option of log or buffered log, and, if necessary, migrates all built-in database extensions to the correct version for the database server. This task creates sub-tasks for individual databases, as necessary.		At server startup	
autoreg vp	This task creates a specialized virtual processor for a database extension as necessary.		As necessary	
auto_tune_cpu_vps	This task automatically adds CPU virtual processors if the number of allocated VPs is less than half the number of CPU processors on the computer. For more information, see Automatic addition of CPU VPs.		At server startup	
Auto Update Statistics Evaluation	This task analyzes all the tables in all logged databases, identifies the tables whose distributions must be updated, and generates UPDATE STATISTICS statements for those tables, based on the current automatic update statistics (AUS) policies. The AUS policies are set by thresholds in the **ph_threshold** table: • AUS_AGE: statistics are updated after 30 days. • AUS_CHANGE: statistics are updated after 10 percent of the data is changed. • AUS_AUTO_RULES: guidelines are followed for updating statistics. • AUS_SMALL_TABLES: tables containing fewer than 100 rows always have their statistics updated. For more information, see Automatic Update Statistics.		Once a day	

Table 27-2. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
Auto Update Statistics Refresh	This task runs the UPDATE STATISTICS statements generated by the Auto Update Statistics Evaluation task. The PDQ priority for updating statistics is set to 10 by the threshold named AUS_PDQ in the **ph_threshold** table.		Saturday and Sunday between 1:00 AM and 5:00 AM	
bad_index_alert	This task checks for corrupted indexes. If any corrupted indexes are found, a warning alert is added to the **ph_alert** table. For more information, see Validate indexes.		Once a day	
bar_act_log_rotate	This task rotates the ON-Bar activity log file that is specified in the BAR_ACT_LOG configuration parameter. When the ON-Bar activity log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	

Table 27-2. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
bar_debug_log_rotate	This task rotates the ON-Bar debug log file that is specified in the BAR_DEBUG_LOG configuration parameter. When the ON-Bar debug log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	
check_backup	This task checks to ensure that backups have run since the time specified by thresholds in the **ph_threshold** table: • REQUIRED LEVEL BACKUP: maximum of 2 days between backups of any level • REQUIRED LEVEL 0 BACKUP: maximum of 2 days between level-0 backups If a backup has not occurred, a warning alert is added to the **ph_alert** table.		Once a day	
check_for_ipa	This task adds an entry in the **ph_alert** table for each table that has one or more outstanding in-place alter operations.		Once a week	
idle_user_timeout	This task terminates user sessions that have been idle for longer than 60 minutes. By default, this task is disabled. You must enable it by updating the **ph_task** table. For more information, see "Automatically terminating idle connections" on page 1-16.		Every 2 hours	

Table 27-2. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
ifx_ha_monitor_log_replay_task	This task monitors the high-availability cluster replay position.		Not set	
Job runner	This task runs server tasks for the OpenAdmin Tool (OAT) for Informix in the background with a private dbWorker thread. For internal use by the OpenAdmin Tool (OAT) for Informix only.		as necessary	
Job results clean up	This task removes OpenAdmin Tool (OAT) for Informix job results entries that are older than the threshold of 30 days. For internal use by the OpenAdmin Tool (OAT) for Informix only.		Once a day	
mon_checkpoint	This sensor saves information about checkpoints.	mon_checkpoint	Every hour	7 days
mon_command_history	This task deletes rows from the **command_history** table that are older than the threshold of 30 days. The threshold is named COMMAND HISTORY RETENTION in the **ph_threshold** table.		Once a day	
mon_config	This sensor saves the most recent value for each configuration parameter in the onconfig file.	mon_config	Once a day	
mon_config_startup	This sensor saves the value for each configuration parameter in the onconfig file when the server starts.	mon_config	At server startup	99 days
mon_low_storage	This task scans the list of dbspaces to find spaces that fall below the threshold specified by the SP_THRESHOLD configuration parameter. Then, the task expands the spaces by extending chunks or adding chunks using entries in the storage pool. For more information, see "Automatic space management" on page 9-26.	mon_low_storage	Every hour	7 days

Table 27-2. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
mon_memory_system	This sensor collects information about the amount of memory the server uses.	mon_memory_system	Every hour	7 days
mon_profile	This sensor saves server profile information.	mon_prof	Every 4 hours	30 days
mon_sysenv	This startup sensor saves information about the environment when the database server starts.	mon_sysenv	At server startup	60 days
mon_table_names	This sensor saves table names along with their creation time.	mon_table_names	Once a day	30 days
mon_table_profile	This sensor saves table profile information, including the total number of update, insert, and delete operations that occurred on this table.	mon_table_profile	Once a day	7 days
mon_users	This sensor saves profile information about each user.	mon_users	Every 4 hours	7 days
mon_vps	This sensor collects virtual processor information.	mon_vps	Every 4 hours	15 days
online_log_rotate	This task rotates the online message log file that is specified in the MSGPATH configuration parameter. When the online message log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	
post_alarm_message	This task posts alerts.		Every hour	
Save SQL Trace	This sensor saves the current SQL History Trace buffer contents to a table when enabled by the user in the OpenAdmin Tool (OAT) for Informix. For internal use by the OpenAdmin Tool (OAT) for Informix only.	sql_savesnap	Every 15 minutes	1 day
SET tk_enable	This task enables the tasks that rotate message log files.		3 A.M. every 30 days	

Related reference:

➡ The Scheduler Tables (Administrator's Reference)

Creating a task

You can create a Scheduler task to perform a specific action at specific times.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To create a task, use an INSERT statement to add a row into the **ph_task** table:

1. Include values for the following columns:
   a. **tk_name**: Give the task a unique name.
   b. **tk_type**: Change the job type to TASK or STARTUP TASK.
   c. **tk_description**: Add a description of the action the task performs.
   d. **tk_execute**: Add the action that the task performs. The action can be a user-defined function, a single SQL statement, or a multiple-statement prepared object that was created using the PREPARE SQL to enable the assembly of one or more SQL statements at runtime. The length of the command is limited to 2048 bytes.
2. Optional: Change the default values for the following columns:
   - **tk_start_time**: The default start time is 8:00:00. For a startup task, set the start time to NULL.
   - **tk_stop_time**: The default stop time is 19:00:00. For a startup task, set the stop time to NULL.
   - **tk_frequency**: The default frequency once a day. For a startup task, set the frequency to NULL.
   - **tk_group**: The default group is MISC.
   - **tk_monday** through **tk_sunday**: The default is to run every day.

The task runs at the specified start time and subsequently at the time calculated from the frequency.

Example

The following task uses the SQL administration API to take a checkpoint every two minutes between the hours of 8:00 A.M. and 7:00 P.M. on Mondays, Wednesdays, and Fridays.

```
INSERT INTO ph_task
( tk_name,
tk_description,
tk_type,
tk_group,
tk_execute,
tk_start_time,
tk_stop_time,
tk_frequency,
tk_Monday,
tk_Tuesday,
tk_Wednesday,
tk_Thursday,
tk_Friday,
```

```
                    tk_Saturday,
                    tk_Sunday)
                    VALUES
                    ( "Example Checkpoint",
                    "Example to do a checkpoint every 2 minutes.",
                    "TASK",
                    "EXAMPLES",
                    "EXECUTE FUNCTION admin('checkpoint')",
                    DATETIME(08:00:00) HOUR TO SECOND,
                    DATETIME(19:00:00) HOUR TO SECOND,
                    INTERVAL ( 2 ) MINUTE TO MINUTE,
                    't',
                    'f',
                    't',
                    'f',
                    't',
                    'f',
                    'f');
```

The following example shows the code for a task that runs once a day at 2:00 A.M. to ensure that the **command_history** table contains only recent data. In this example, the definition of recent data is stored in a **Command History Interval** column in the **ph_threshold** table.

```
INSERT INTO ph_task
(
tk_name,
tk_group,
tk_description,
tk_type,
tk_execute,
tk_start_time,
tk_frequency
)
VALUES
(
"mon_command_history",
"TABLES",
"Monitor how much data is kept in the command history table",
"TASK",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY INTERVAL' ) ",
"2:00:00",
"1 0:00:00"
);
```

Related concepts:

"Actions for task and sensors" on page 27-12

Related tasks:

"Modifying the scheduler" on page 27-19

Creating a sensor

You can create a Scheduler sensor to collect and store data about the database server.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To create a sensor, use an INSERT statement to add a row into the **ph_task** table:

1. Include values for the following columns:

- **tk_name**: Give the task a unique name.
- **tk_description**: Add a description of the action the task performs.
- **tk_result_table**: Add the name of the table that holds the data that the sensor gathers.
- **tk_create**: Add a CREATE statement to create the results table. The results table must have an INTEGER column named ID to hold the sensor ID. You can add other columns to the table.
- **tk_execute**: Add the action that the sensor performs. The action can be a user-defined function, a single SQL statement, or a multiple-statement prepared object that was created using the PREPARE SQL to enable the assembly of one or more SQL statements at runtime.

2. Optionally change the default values for the following columns:
   - **tk_type**: The default value is SENSOR. For a startup sensor, change the value to STARTUP SENSOR.
   - **tk_delete**: The default interval after which to delete sensor data is one day.
   - **tk_start_time**: The default start time is 8:00:00. For a startup sensor, set the start time to NULL.
   - **tk_stop_time**: The default stop time is 19:00:00. For a startup sensor, set the stop time to NULL.
   - **tk_frequency**: The default frequency once a day. For a startup sensor, set the frequency to NULL.
   - **tk_group**: The default group is MISC.
   - **tk_monday** through **tk_sunday**: The default is to run every day.
   -

The sensor runs at the specified start time and subsequently at the time calculated from the frequency.

Examples

The following example shows the code for a sensor that tracks the startup environment of the database server. The **$DATA_SEQ_ID** variable is the current execution of the sensor.

```
INSERT INTO ph_task
(
tk_name,
tk_type,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete
)
VALUES
(
"mon_sysenv",
"STARTUP SENSOR",
"SERVER",
"Tracks the database servers startup environment.",
"mon_sysenv",
"create table mon_sysenv (ID integer, name varchar(250), value lvarchar(1024))",
"insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value
```

```
FROM sysmaster:sysenv",
NULL,
NULL,
NULL,
"60 0:00:00"
);
```

The following example shows the code for a sensor that collects information about the amount of memory that is being used and stores the information in the **mon_memory_system** table. If that table does not exist, the task creates it. This task, which runs every 30 minutes, deletes any data in the **mon_memory_system** table that has existed for more than 30 days.

```
INSERT INTO ph_task
(
tk_name,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete
)
VALUES
("mon_memory_system",
"MEMORY",
"Server memory consumption",
"mon_memory_system",
"create table mon_memory_system (ID integer, class smallint, size int8,
    used int8, free int8 )",
"insert into mon_memory_system select $DATA_SEQ_ID, seg_class, seg_size,
    seg_blkused, seg_blkfree FROM sysmaster:sysseglst",
NULL,
NULL,
INTERVAL ( 30 ) MINUTE TO MINUTE,
INTERVAL ( 30 ) DAY TO DAY
);
```

Related concepts:

"Actions for task and sensors"

Related tasks:

Actions for task and sensors

The action for a task or sensor is an SQL statement or routine that performs one or more operations.

SQL statements are useful if the action consists of a single operation. A stored procedure or a user-defined routine written in C or Java is useful if the action consists of multiple operations. The action is stored in the **tk_execute** column of the **ph_task** table.

You have a great deal of flexibility when you create an action. Possible types of operations include:

• Perform a DML operation. You can use a sensor to insert or update data in a table. You can use a task to delete older data from a table.

- Perform an administrative operation. You can use a task to run an SQL administration API function to administer the database server. For example, you can create a task to take checkpoints every two minutes.
- Perform an operation based on a threshold. You can use a threshold from the **ph_threshold** table to determine if a task action must be run. For example, you can create a task that adds a shared memory segment if the amount of available shared memory falls below a threshold value.
- Create an alert to report an operation or warn of a potential problem. For example, you can create a task to terminate idle users that inserts a row into the **ph_alert** table when a user session is terminated. You can also create a task to monitor backups and insert a warning into the **ph_alert** table when a backup has not occurred.

Use the following variables in your task or sensor action:
- **$DATA_TASK_ID**: Use to indicate the current task or sensor. This variable corresponds to the value of the **tk_id** field in the **ph_task** table.
- **$DATA_SEQ_ID**: Use to indicate the current execution of the task or sensor. This variable corresponds to the value of the **tk_sequence** field in the **ph_task** table and the **run_task_sequence** field in the **ph_run** table.

Examples

The following action is an SQL statement that the built-in **mon_command_history** task uses to remove older rows from the **command_history** table.

```
DELETE FROM command_history
WHERE cmd_exec_time < (
SELECT CURRENT - value::INTERVAL DAY to SECOND
FROM ph_threshold
WHERE name = 'COMMAND HISTORY RETENTION' )
```

The following example is an SQL statement that the built-in **mon_vps** sensor uses to add data to the **mon_vps** result table:

```
INSERT INTO mon_vps
SELECT $DATA_SEQ_ID, vpid, num_ready,
class, usecs_user, usecs_sys
FROM sysmaster:sysvplst
```

The following example is a stored procedure that terminates user sessions that are idle for longer than a value set by a threshold, and then adds an alert to the **ph_alert** table.

```
/*
 ***************************************************************
 *  Create a function that will find all users that have
 *   been idle for the specified time. Call the SQL admin API to
 *   terminate those users.  Create an alert to track which
 *   users have been terminated.
 ***************************************************************
 */
CREATE FUNCTION idle_timeout( task_id INT, task_seq INT)
RETURNING INTEGER

DEFINE time_allowed INTEGER;
DEFINE sys_hostname CHAR(16);
DEFINE sys_username CHAR(257);
DEFINE sys_sid      INTEGER;
DEFINE rc           INTEGER;

    {*** Get the maximum amount of time to be idle ***}
    SELECT value::integer
```

```
   INTO time_allowed
   FROM ph_threshold
         WHERE name = "IDLE TIMEOUT";

  {*** Find all users who are idle longer than the threshold ***}
  FOREACH SELECT admin("onmode","z",A.sid), A.username, A.sid, hostname
         INTO rc, sys_username, sys_sid, sys_hostname
         FROM sysmaster:sysrstcb A , sysmaster:systcblst B,
            sysmaster:sysscblst C
         WHERE A.tid = B.tid
         AND C.sid = A.sid
         AND lower(name) in  ("sqlexec")
         AND CURRENT - DBINFO("utc_to_datetime",last_run_time) > time_allowed UNITS MINUTE
         AND lower(A.username) NOT IN( "informix", "root")

         {*** If a user is successfully terminated, log ***}
         {*** the information into the alert table.    ***}
         IF rc > 0 THEN
            INSERT INTO ph_alert
               (
               ID, alert_task_id,alert_task_seq,
               alert_type, alert_color,
               alert_state,
               alert_object_type, alert_object_name,
               alert_message,
               alert_action
               ) VALUES (
               0,task_id, task_seq,
               "INFO", "GREEN",
               "ADDRESSED",
               "USER","TIMEOUT",
   "User "||TRIM(sys_username)||"@"||TRIM(sys_hostname)||
      " sid("||sys_sid||")"||
                           " terminated due to idle timeout.",
               NULL
               );
         END IF

   END FOREACH;

   RETURN 0;

END FUNCTION;
```

Related tasks:

"Creating a task" on page 27-9

"Creating a sensor" on page 27-10

"Creating a threshold" on page 27-15

"Creating an alert" on page 27-16

Creating a group

You can create a group to organize Scheduler tasks and sensors.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

When you create a task or sensor, you can specify a group name from the **ph_group** table in the **tk_group** column of the **ph_task** table.

To create a group:

Use an INSERT statement to add a row into the **ph_group** table in the **sysadmin** database. You must include a name for the group for the **group_name** column and a description of the group for the **group_description** column. The database server generates an ID for the group in the **group_id** column.

Example

The following example adds a group named TABLES:

```
INSERT INTO ph_group
(
group_name,
group_description
)
VALUES
(
"TABLES",
"Tasks that trim history and results tables."
);
```

Creating a threshold

You can create a threshold to determine under what conditions a Scheduler task or sensor is run.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

A threshold specifies a value that can be compared to a current value to determine whether a task or sensor must be run.

To create a threshold:
1. Use an INSERT statement to add values for the following columns in the **ph_threshold** table:
   - **name**: the name of the threshold
   - **task_name**: the name of the task from the **ph_task** table
   - **value**: the value of the threshold
   - **value_type**: the data type of the threshold (STRING or NUMERIC)
   - **description**: a description of what the threshold does
2. Write the task or sensor action to use the threshold.

Example

The following example adds a threshold named IDLE TIMEOUT for a task named Idle_timeout:

```
INSERT INTO ph_threshold
(
name,
task_name,
value,
value_type,
description
)
VALUES
(
"IDLE TIMEOUT",
"Idle_timeout",
```

```
"60",
"NUMERIC",
"Maximum amount of time in minutes for non-informix users to be idle."
);
```

The task action subtracts the time of the last user action from the current time and compare that value with the value column in the **ph_threshold** table.

Related concepts:

"Actions for task and sensors" on page 27-12

Creating an alert

You can create an alert as part of the action of a Scheduler task or sensor.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To create an alert:

Use an INSERT statement to add a row to the **ph_alert** table. Include values for the following columns:

- **ID**: System generated; use 0 for the value.
- **alert_task_id**: Must reference the job ID from the **ph_task** table.
- **alert_task_seq**: Must reference the job sequence number from the **ph_task** table.
- **alert_type**: Choose INFO, WARNING, or ERROR.
- **alert_color**: Choose GREEN, YELLOW, or RED.
- **alert_state**: Choose NEW, IGNORED, ACKNOWLEDGED, ADDRESSED.
- **alert_object_type**: The type of object the alert describes, for example, SERVER.
- **alert_object_name**: The name of the object.
- **alert_message**: The message describing the alert.
- **alert_action**: An SQL statement or function that performs a corrective action, or NULL.

Example

The following example adds an alert to warn that a backup has not been taken. This code snippet is part of a stored procedure that takes **task_id** and **task_seq** as its arguments.

```
INSERT INTO ph_alert
(
ID,
alert_task_id,
alert_task_seq,
alert_type,
alert_color,
alert_state,
alert_object_type,
alert_object_name,
alert_message,
alert_action
)
VALUES
(
0,
task_id,
task_seq,
```

```
"WARNING",
"RED",
"NEW",
"SERVER",
"dbspace_name",
"Dbspace ["||trim(dbspace_name)|| "] has never had a level-0 backup.
    Recommend taking a level-0 backup immediately.",
NULL
);
```

Related concepts:

"Actions for task and sensors" on page 27-12

Monitor the scheduler

You can monitor Scheduler threads that are in progress with the **onstat -g dbc** command. You can view information about tasks and sensors that have completed in the **ph_run** table.

The Scheduler uses two types of threads while it is running:

- **dbWorker:** These threads are running scheduled tasks and sensors.
- **dbScheduler:** This thread prepares the next task or sensor that is scheduled to be run.

To view information about currently running tasks and sensor, and the next task or sensor that is run, use the **onstat -g dbc** command.

To view information about tasks and sensor that have completed, query the **ph_run** table in the **sysadmin** database. You must be connected to the **sysadmin** database as user **informix** or another authorized user.

Examples

The following output from the **onstat -g dbc** command shows two **dbWorker** threads and the **dbScheduler** thread:

```
Worker Thread(0)    46fa6f10
=====================================
Task:              47430c18
Task Name:         mon_config_startup
Task ID:           3
Task Type:         STARTUP SENSOR
Last Error
    Number             -310
    Message            Table (informix.mon_onconfig)
                         already exists in database.
    Time               09/11/2007 11:41
    Task Name          mon_config_startup

Task Execution:    onconfig_save_diffs

WORKER PROFILE
    Total Jobs Executed        10
    Sensors Executed           8
    Tasks Executed             2
    Purge Requests             8
    Rows Purged                0

Worker Thread(1)    46fa6f80
=====================================
Task:              4729fc18
Task Name:         mon_sysenv
```

```
Task ID:            4
Task Type:          STARTUP SENSOR
Task Execution:     insert into mon_sysenv select 1, env_name,
                      env_value FROM sysmaster:sysenv


WORKER PROFILE
    Total Jobs Executed      3
    Sensors Executed         2
    Tasks Executed           1
    Purge Requests           2
    Rows Purged              0


Scheduler Thread    46fa6f80
=====================================
Run Queue
    Empty
Run Queue Size        0
Next Task             7
Next Task Waittime    57
```

The following output shows the history of four Scheduler jobs from the **ph_run** table:

```
SELECT * FROM ph_run;

RUN_ID          1
RUN_TASK_ID     2
RUN_TASK_SEQ    1
RUN_RETCODE     0
RUN_TIME        2009-07-20 13:04:59
RUN_DURATION    0.131850300007433
RUN_ZTIME       1248109468
RUN_BTIME       1248109468
RUN_MTIME       1248109499

RUN_ID          2
RUN_TASK_ID     3
RUN_TASK_SEQ    1
RUN_RETCODE     0
RUN_TIME        2009-07-20 13:04:59
RUN_DURATION    0.120845244247991
RUN_ZTIME       1248109468
RUN_BTIME       1248109468
RUN_MTIME       1248109499

RUN_ID          3
RUN_TASK_ID     4
RUN_TASK_SEQ    1
RUN_RETCODE     0
RUN_TIME        2009-07-20 13:04:59
RUN_DURATION    0.00254887164461759
RUN_ZTIME       1248109468
RUN_BTIME       1248109468
RUN_MTIME       1248109499

RUN_ID          2087
RUN_TASK_ID     7
RUN_TASK_SEQ    742
RUN_RETCODE     0
RUN_TIME        2009-09-09 11:09:51
RUN_DURATION    0.00489335523104662
RUN_ZTIME       1248109468
RUN_BTIME       1248109468
RUN_MTIME       1252508991
```

Modifying the scheduler

You can modify the properties of Scheduler tasks, sensors, alerts, thresholds, or groups. You can modify both built-in properties and properties that you added.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To modify Scheduler properties:

Use an UPDATE statement for the appropriate Scheduler table in the **sysadmin** database.

Examples

The following example stops a task named `task1` from running:

```
UPDATE ph_task
  SET tk_enable = "F"
    WHERE tk_name = "task1";
```

The following example changes the amount of time that data collected by the built-in sensor `mon_profile` to 99 days:

```
UPDATE ph_task
SET tk_delete = "INTERVAL ( 99 ) DAY TO DAY"
WHERE tk_name = "mon_profile";
```

The following example changes the threshold named COMMAND HISTORY RETENTION to 20 so that the **command_history** table retains information about SQL administration API commands for 20 days:

```
UPDATE ph_threshold SET value = "20 0:00:00"
WHERE name = "COMMAND HISTORY RETENTION";
```

Related tasks:

"Creating a task" on page 27-9

"Creating a sensor" on page 27-10

Related reference:

"Built-in tasks and sensors" on page 27-3

Chapter 28. Remote administration with the SQL administration API

You can use the SQL administration API to perform remote administration tasks by using SQL statements.

The SQL administration API functions take one or more arguments that define the task. Many of the tasks are ones that you can also complete with command-line utilities. The advantage of using the SQL administration API functions is that you can run them remotely from other database servers. You must be directly connected to the database server when you run command-line utility commands.

You can perform the following types of administrative tasks with the SQL administration API:
* Control data compression
* Update configuration parameters
* Check data, partitions, and extents consistency, control the B-tree scanner, and force a checkpoint
* Set up and administer Enterprise Replication
* Set up and administer high-availability clusters
* Control logging and logical logs
* Control shared-memory and add buffer pools
* Control mirroring
* Control decision-support queries
* Change the server mode
* Add, drop, and configure storage spaces
* Control the SQL statement cache
* Control and configure SQL tracing
* Start and stop the listen control threads dynamically
* Perform other tasks, such as moving the **sysadmin** database, terminating a session, or adding a virtual processor

For more information about the SQL administration API, see the *IBM Informix Administrator's Reference*.

Related concepts:

Part 7, "Overview of automatic monitoring and corrective actions"

Related reference:

➡ SQL Administration API Overview (Administrator's Reference)

SQL administration API admin() and task() functions

The SQL administration API consists of two functions: **admin()** and **task()** that are defined in the **sysadmin** database.

These functions perform the same tasks, but return results in different formats. The **task()** function returns a string that describes the results of the command. The **admin()** function returns an integer.

By default, only user **informix**, can connect to the **sysadmin** database. If user **root** or a member of the DBSA group is granted privileges to connect to the **sysadmin** database, user root or a member of the DBSA group can also run the SQL administration API **task()** and **admin()** functions.

You can use EXECUTE FUNCTION statement to execute the **admin()** and **task()** functions. For example, the following SQL statement, which is equivalent to the **oncheck -ce** command, instructs the database server to check the extents:

```
EXECUTE FUNCTION admin("check extents");
```

You use SQL administration API functions in your Scheduler task actions. For example, you can define a task that creates a dbspace by using the following statement in the task action:

```
EXECUTE FUNCTION admin("create dbspace","dbspace2","/work/dbspace2","20 MB");
```

For information about using **admin()** and **task()** functions and examples, see the *IBM Informix Administrator's Reference*.

Related reference:

➥ SQL Administration API Overview (Administrator's Reference)

Viewing SQL administration API history

You can view the history of all the SQL administration API functions that the were run in the previous 30 days in the **command_history** table in the **sysadmin** database.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

The **command_history** table shows if an administrative task was executed through an **admin()** or **task()** function and displays information about the user who ran the command, the time the command was run, the command, and the message returned when the database server completed running the command.

To display command history:

Use a SELECT statement to return the data from the **command_history** table.

The following example displays all command history for the past 30 days:

```
SELECT * FROM command_history;
```

The following table shows sample commands and the associated results in a sample **command_history** table. For a description of all information in the **command_history** table, see the *IBM Informix Administrator's Reference*.

Table 28-1. Example of some information in a command_history table

Command	Sample returned messages
set sql tracing on	`SQL tracing on with 1000 buffers of 2024 bytes.`
create dbspace	`Space 'space12' added.`
checkpoint	`Checkpoint completed.`
add log	`Added 3 logical logs to dbspace logdbs.`

Related concepts:

Part 7, "Overview of automatic monitoring and corrective actions"

Related reference:

➥ The command_history table (Administrator's Reference)

Controlling the size of the command_history table

You can reduce the retention period or remove rows from the **command_history** table to prevent it from becoming too large.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

By default, rows in the **command_history** table are automatically removed after a 30 days. The retention period is controlled by the COMMAND HISTORY RETENTION row in the **ph_threshold** table.

To reduce the retention period:

Use an UPDATE statement to modify the value of the COMMAND HISTORY RETENTION row in the **ph_threshold** table.

The following example sets the retention period to 25 days:

```
UPDATE ph_threshold
SET value = "25"
WHERE name = "COMMAND HISTORY RETENTION";
```

You can use SQL commands like DELETE or TRUNCATE TABLE to manually remove data from this table. You can also create a task in the **ph_task** table to purge data from the **command_history** table.

The following example shows a task that monitors the amount of data in the **command_history** table and purges data when it becomes too old.

```
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_command_history",
"TASK",
"TABLES",
"Monitor how much data is kept in the command history table",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY RETENTION' ) ",
DATETIME(02:00:00) HOUR TO SECOND,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

Related reference:

➥ The command_history table (Administrator's Reference)

Chapter 29. Query drill-down

You can use query drill-down, or SQL tracing, to gather statistical information about each SQL statement that was run and to analyze statement history.

SQL tracing helps you answer questions such as:
- How long do SQL statements take?
- How many resources are individual statements using?
- How long did statement execution take?
- How much time was involved waiting for each resource?

The statistical information is stored in a circular buffer, which is an in-memory pseudo table, called **syssqltrace**, that is stored in the **sysmaster** database. You can dynamically resize the circular buffer.

By default SQL tracing turned off, but you can turn it on for all users or for a specific set of users. When SQL tracing is enabled with its default configuration, the database server tracks the last 1000 SQL statements that ran, along with the profile statistics for those statements. You can also disable SQL tracing globally or for a particular user.

The memory required by SQL tracing is large if you plan to keep much historical information. The default amount of space required for SQL tracing is two megabytes. You can expand or reduce the amount of storage according to your requirements.

Information displayed includes:
- The user ID of the user who ran the command
- The database session ID
- The name of the database
- The type of SQL statement
- The duration of the SQL statement execution
- The time this statement completed
- The text of the SQL statement or a function call list (also called *stack trace*) with the statement type, for example:
  ```
  procedure1() calls procedure2() calls procedure3()
  ```
- Statistics including the:
  - Number of buffer reads and writes
  - Number of page reads and writes
  - Number of sorts and disk sorts
  - Number of lock requests and waits
  - Number of logical log records
  - Number of index buffer reads
  - Estimated number of rows
  - Optimizer estimated cost
  - Number of rows returned
- Database isolation level.

You can also specify escalating levels of information to include in the tracing, as follows:

- low-level tracing, which is enabled by default, captures the information shown in the example below. This information includes statement statistics, statement text, and statement iterators.
- Medium level tracing captures all of the information included in low-level tracing, plus the list of table names, database name and stored procedure stacks.
- high-level tracing captures all of the information included in medium-level tracing, plus host variables.

The amount of information traced affects the amount of memory required for this historical data.

You can enable and disable the tracing at any point in time, and you can change the number and size of the trace buffers while the database server is running. If you resize the trace buffer, the database server attempts to maintain the content of the buffer. If the parameters are increased, data is truncated. However, if the number or the size of the buffers are reduced, the data in the trace buffers might be truncated or lost.

The number of buffers determines how many SQL statements are traced. Each buffer contains the information for a single SQL statement. By default, an individual trace buffer is a fixed size. If the text information stored in the buffer exceeds the size of the trace buffer, then the data is truncated.

The following example shows SQL tracing information:

```
select * from syssqltrace where sql_id = 5678;

sql_id          5678
sql_address     4489052648
sql_sid         55
sql_uid         2053
sql_stmttype    6
sql_stmtname    INSERT
sql_finishtime  1140477805
sql_begintxtime 1140477774
sql_runtime     30.86596333400
sql_pgreads     1285
sql_bfreads     19444
sql_rdcache     93.39127751491
sql_bfidxreads  5359
sql_pgwrites    810
sql_bfwrites    17046
sql_wrcache     95.24815205913
sql_lockreq     10603
sql_lockwaits   0
sql_lockwttime  0.00
sql_logspace    60400
sql_sorttotal   0
sql_sortdisk    0
sql_sortmem     0
sql_executions  1
sql_totaltime   30.86596333400
sql_avgtime     30.86596333400
sql_maxtime     30.86596333400
sql_numiowaits  2080
sql_avgiowaits  0.014054286131
sql_totaliowaits 29.23291515300
sql_rowspersec  169.8958799132
sql_estcost     102
sql_estrows     1376
```

```
sql_actualrows    5244
sql_sqlerror      0
sql_isamerror     0
sql_isollevel     2
sql_sqlmemory     32608
sql_numiterators  4
sql_database      db3
sql_numtables     3
sql_tablelist     t1
sql_statement     insert into t1 select {+ AVOID_FULL(sysindices) } 0, tabname
```

For an explanation of all table rows, see information about the **syssqltrace** table in the **sysmaster** database section of the *IBM Informix Administrator's Reference*.

Related concepts:

Part 7, "Overview of automatic monitoring and corrective actions"

Specifying startup SQL tracing information by using the SQLTRACE configuration parameter

Use the SQLTRACE configuration parameter to control the default tracing behavior when the database server starts. By default, this parameter is not set. The information you set includes the number of SQL statements to trace and the tracing mode.

Any user who can modify the `onconfig` file can modify the value of the SQLTRACE configuration parameter and effect the startup configuration. However, only user **informix**, **root**, or a DBSA who has been granted connect privileges to the **sysadmin** database can use SQL administration API commands to modify the runtime status of the SQL tracing.

To specify SQL tracing information when the database server starts:
1. Set the SQLTRACE configuration parameter in the `onconfig` file.
2. Restart the database server.

Example

The following setting in the `onconfig` file specifies that the database server gathers low-level information about up to 2000 SQL statements executed by all users on the system and allocates approximately four megabytes of memory (2000 * two KB).

```
SQLTRACE level=LOW,ntraces=2000,size=2,mode=global
```

If you use only a percentage of the allocated buffer space (for example, 42 percent of the buffer space), the amount of memory that is allocated is still two KB.

If you do not want to set the SQLTRACE configuration parameter and restart the server, you can run the following SQL administration API command, which provides the same function as setting SQLTRACE for the current session:

```
EXECUTE FUNCTION task("set sql tracing on", 100,"1k","med","user");
```

After enabling the SQL tracing system in user mode, you can then enable tracing for each user. See "Enable SQL tracing" on page 29-4.

For more information about using **task()** and **admin()** functions, see the *IBM Informix Administrator's Reference*.

For more information about the SQLTRACE configuration parameter, including minimum and maximum values for some fields, see the *IBM Informix Administrator's Reference*.

Disable SQL tracing globally or for a session

Even if the mode specified in the SQLTRACE configuration parameter is `global` or `user`, you can disable SQL tracing if you want to completely turn off all user and global tracing and deallocate resources currently in use by SQL tracing. By default, SQL tracing is off for all users.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To disable global SQL tracing, run the SQL administration API **task()** or **admin()** function with the `set sql tracing off` argument.

To disable SQL tracing for a particular session, run the SQL administration API **task()** or **admin()** function with `set sql tracing off` as the first argument and the session identification number as the second argument.

Examples

The following example disables SQL tracing globally:
```
EXECUTE FUNCTION task('set sql tracing off');
(expression) SQL tracing off.

1 row(s) retrieved.
```

The following example disables SQL tracing for the session with an ID of 47:
```
EXECUTE FUNCTION task("set sql user tracing off",47);
```

For more information about using **task()** or **admin()** functions, see the *IBM Informix Guide to SQL: Syntax*.

Enable SQL tracing

After you specify `user` as the mode in the SQLTRACE configuration parameter, you must run the SQL administration API **task()** or **admin()** function to turn SQL history tracing on for a particular user.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

Global SQL tracing is not required to be enabled to allow SQL tracing for a particular user.

To enable SQL tracing for a particular user, run the SQL administration API **task()** or **admin()** function with `set sql tracing on` as the first argument and the user session ID as the second argument.

To enable user SQL tracing for all users except **root** or **informix**, you can run a **task()** or **admin()** function with the `set sql tracing on` argument and information that defines the users.

Example

The following example enables SQL tracing for the user with the session ID of 74:

```
EXECUTE FUNCTION task("set sql user tracing on", 74);
```

The following example enables the tracing of SQL statements of users who are currently connected to the system as long as they are not logged in as user **root** or **informix**.

```
dbaccess sysadmin -<<END
 execute function task("set sql tracing on", 1000, 1,"low","user");
 select task("set sql user tracing on", session_id)
  FROM sysmaster:syssessions
  WHERE username not in ("root","informix");
END
```

For more information about the **task()** and **admin()** functions, see the *IBM Informix Administrator's Reference*.

Enable global SQL tracing for a session

You can enable global SQL tracing for the current session by running the SQL administration API **task()** or **admin()** function.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

By default, global SQL tracing is not enabled. You can set the SQLTRACE configuration parameter to permanently enable global tracing.

To enable global user SQL history tracing for the current database server session, run the SQL administration API **task()** or **admin()** function with the **set sql tracing on** argument.

Example

The following example enables global low-level SQL tracing for all users:

```
EXECUTE FUNCTION task("set sql tracing on", 1000, 1,"low","global");
```

If a new user logs on to the system after your statement runs, you can enable tracing for the new user. See "Enable SQL tracing" on page 29-4.

For more information about the **task()** and **admin()** functions, see the *IBM Informix Guide to SQL: Syntax*.

Part 8. Appendixes

Appendix. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix products

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM Informix products. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

Related accessibility information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the IBM commitment to accessibility.

Dotted decimal syntax diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, that element is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 refers to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

?   Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

!   Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*   Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be

repeated. For example, if you hear the line `5.1* data-area`, you know that you can include more than one data area or you can include none. If you hear the lines `3*`, `3 HOST`, and `3 STATE`, you know that you can include `HOST`, `STATE,` both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.

3. The `*` symbol is equivalent to a loop-back line in a railroad syntax diagram.

+   Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line `6.1+ data-area`, you must include at least one data area. If you hear the lines `2+`, `2 HOST`, and `2 STATE`, you know that you must include `HOST`, `STATE`, or both. As for the * symbol, you can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy,

modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

Numerics

A

IBM®

Printed in USA

Spine information:

Informix Product Family Informix   Version 11.70   IBM Informix Administrator's Guide

IBM