HCL OneDB 2.0.1

Administrator's Guide

Contents

Chapter 1. Administrator's Guide

The *HCL OneDB™ Administrator's Guide* provides the information required to administer HCL OneDB™.

A companion volume, the *HCL OneDB™ Administrator's Reference*, contains reference material for using HCL® OneDB® database servers. If you must tune the performance of your database server and SQL queries, see your *HCL OneDB™ Performance Guide*.

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

The database server

Overview of database server configuration and administration

After you install HCL OneDB™, you configure the database server system and start administering the database server.

When you install HCL OneDB™, 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 *HCL OneDB™ Installation Guide*.

When you install HCL OneDB™, 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 *HCL OneDB™ Installation Guide for UNIX™, Linux™, and Mac OS X* and the *HCL OneDB™ Installation Guide for Windows™*.

You must have the correct permissions to administer the database server. For most administration tasks, you need the following permissions:

- On UNIX™, you must be logged in as user **root**, user **informix**, or the owner of the non-root installation. If role separation is enabled, you must be granted the DBSA role.
- On Windows™, you must be a member of the **Informix-Admin** group.

You have various options to choose from when you configure the database server. Configuration includes customizing your environment and the database server. You can control how the database server runs and what function is available.

You must configure connectivity to connect to client administration tools and applications.

You must do some initial administration tasks to finish setting up your database server system. After you configure the database server, your administration responsibilities include a set of routine tasks.

Database server concepts

To administer the database server, you must understand key concepts around storage, configuration, logging, CPU use, shared memory use, and automation.

Root dbspace

The root dbspace is the initial dbspace, or storage space, that the database server creates. The root dbspace contains reserved pages and internal tables that describe and track all physical and logical units of storage. The root dbspace is the default location for logical logs, the physical log, databases, and temporary tables. The database server cannot run without the root dbspace.

Configuration (onconfig) file

The database server requires a configuration file. Typically, the name of the configuration file is `onconfig.`*`server_name`*. The `onconfig` file contains configuration parameters that control database server properties. The database server reads the `onconfig` file during startup, shutdown, and for some operations while the server is running. Many configuration parameters can also be set dynamically while the database server is running.

Virtual processors

A virtual processor runs multiple threads to perform queries and other tasks. The operating system schedules virtual processors as CPU processes. Multiple virtual processors run multiple threads in parallel. Virtual processors are divided into classes where each class is dedicated to processing a specific type of thread.

Logical logs

The database server contains several logical log files that record data manipulation operations for logged databases, data definition operations for all databases, and administrative information such as checkpoint records and additions and deletions of chunks. A logical log is similar to a transaction log in other relational database management systems.

Physical log

The physical log stores the before-images of pages. "Before images" are images of pages that are taken before the database server records the changed pages on disk. The unmodified pages are available in case the database server fails or a backup procedure requires the pages to provide an accurate snapshot of the database server data.

Buffer pool

The buffer pool contains buffers that cache pages from disk in shared memory. Operations on pages that are cached run faster than operations on pages that must be retrieved from disk.

Caches

The database server uses caches to store information in shared memory instead of performing a disk read or another operation to obtain the information. Caching information improves performance for multiple queries that access the same tables.

Scheduler

The Scheduler is a subsystem that runs a set of tasks at predefined times or as determined internally by the server. Tasks are SQL statements can either collect information or run a specific operation. Some tasks are internal to the database server and run automatically. You can enable other tasks, if appropriate. You can also create your own tasks and schedule when they are run.

System databases

The system databases contain information about the database server. The **sysmaster** database contains the system-monitoring interface (SMI) tables. The SMI tables provide information about the state of the database server. The **sysadmin** database contains the tables that contain and organize the Scheduler tasks and sensors, store data that is collected by sensors, and record the results of Scheduler jobs and SQL administration API functions.

Environment configuration

You configure your environment by setting environment variables and creating or modifying files that relate to the environment variables. You can control whether environment variables are set at the environment level, for a specific user, or for a database session. You must set environment variables for the database server environment and for the client environments.

If you choose to create a database server instance during installation, the installation program sets the mandatory environment variables. Otherwise, you must set environment variables before you start the database server. The following environment variables are mandatory:

- The **ONEDB_HOME** environment variable specifies the directory where you installed the database server.
- The **ONEDB_SERVER** environment variable specifies the name of the database server.
- The **ONCONFIG** environment variable specifies the name of the `onconfig` file in the `ONEDB_HOME/etc` directory.
- The **PATH** environment variable must include the `ONEDB_HOME/bin` directory.

To configure the database server environment, you can set other environment variables:

- If you plan to create an sqlhosts file with a non-default name or location, set the **ONEDB_ SQLHOSTS** environment variable to the name and path of your sqlhosts file.
- If you plan to use the DB-Access utility to run SQL statements, specify terminal properties with the **ONEDB_ TERM** or a similar environment variable.
- If you need Global Language Support (GLS), set GLS environment variables.
- If you want to enable other functionality, set the appropriate environment variables. Some environment variables control functionality that is also controlled by configuration parameters. Environment variables override configuration parameter settings.

To configure client environments, you can set the environment variables that are supported by your client API. For more information, see your client API manual.

You can choose from multiple methods for setting environment variables. For example, you can run the SET ENVIRONMENT statement to set environment variables for the current session. You can add environment variable settings to log in scripts, at the command prompt, or in a configuration file.

Database server configuration

You must customize the database server properties and features by setting configuration parameters, create storage spaces, and configure connectivity. You can automate startup.

You customize the database server properties by setting or modifying configuration parameters in the `onconfig` file. The current version of HCL OneDB™ does not use some configuration parameters that are used in earlier versions of the server.

If you choose to configure a database server during installation, many configuration parameter and environment variables are set and a set of storage spaces are created automatically. Alternatively, you can manually configure the database server.

When you start the database server for the first time, disk space is initialized and the initial chunk of the root dbspace is created. Any existing data in that disk space is overwritten. Shared memory that the database server requires is also initialized. When you subsequently start the database server, only shared memory is initialized. Although the root dbspace is the default location of log files and databases, you can store log files and databases in other storage spaces to prevent the root dbspace from running out of space.

Storage space creation and management

You can create multiple storage spaces to store different types of objects, such as, data, indexes, logs, temporary objects, instead of storing everything in the root dbspace. The way that you distribute the data on disks affects the performance of the database server. You can configure the database server to both automatically minimize the storage space that data requires and automatically expand storage space as needed. You can segregate storage and processing resources among multiple client organization by configuring multitenancy.

A storage space is composed of one or more chunks. The maximum chunk size is 4 TB. You can have up to 32766 chunks in an instance.

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

The following storage spaces are the most common:

dbspace

Stores databases, tables, logical-log files, and the physical log file.

Temporary dbspaces store temporary tables.

sbspace

Stores 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. Some features of HCL OneDB™ require sbspaces, such as Enterprise Replication, J/Foundation, spatial data types, and basic text search queries. In some cases, sbspaces are created automatically when needed.

Temporary sbspaces store temporary smart large objects without logging metadata or user data.

plogspace

Stores the physical log. If you do not create a plogspace, the physical log is stored in a dbspace.

Other types of storage spaces store specialized types of data.

If you create a server during installation, some storage spaces are created automatically.

Automatically minimizing storage space

You can minimize the amount of space that data needs by configuring automatic data compression and consolidation. You can compress data, consolidate and return free space, and merge extents. You can specify how frequently each of the operations occurs.

You can automatically rotate message logs to limit the amount of space for the logs.

Automatically extending storage space

After you create storage spaces, you can configure the server to automatically extend each storage space as needed. You create a storage pool of entries for available raw devices, cooked files, and directories, and you make sure that the SP_AUTOEXPAND configuration parameter set to the default value of 1. All types of storage spaces except external spaces (extspaces) are automatically expanded.

Automatically managing the location of data

You can automate the process of deciding where to locate databases, tables, and indexes. You can enable the database server to choose the most optimal location for databases, table, and indexes, and to automatically fragment tables. Instead of creating a new database in the root dbspace by default, the database server chooses the location by favoring non-critical spaces, spaces that have the most efficient page size, and other factors. The database server fragments new tables by round-robin and adds more fragments when necessary as the table grows.

You can override the automatic behavior by specifying a location for a database or table.

Multitenancy

You can create multiple tenant databases in a single HCL OneDB™ instance to segregate data, storage space, and processing resources among multiple client organizations.

Automatic performance tuning

You can set configuration parameters and Scheduler tasks to enable the database server to automatically adjust values that affect performance. By default, many automatic tuning configuration parameters and Scheduler tasks are set to solve common performance issues.

You can configure the database server to adjust resources to improve performance:

- Increase the number of CPU virtual processors (VPs), up to the number of CPU processors or the number that you specify. Set the VPCLASS configuration parameter for the cpu class to autotune=1.
- Increase the number of AIO VPs. Set the VPCLASS configuration parameter for the aio class to autotune=1.
- Increase the size of the buffer pool. Set the BUFFERPOOL configuration parameter to enable the automatic extension of the buffer pool.
- Increase or decrease the size of private memory caches for CPU VPs. Set the VP_MEMORY_CACHE_KB configuration parameter to the initial size of the private memory caches.
- Increase the number of logical log files to improve performance. Set the AUTO_LLOG configuration parameter to 1, plus the name of the dbspace in which to add log files, and an optional maximum number of KB for all logical log files.
- Increase the size of the physical log as needed to improve performance. Create the plogspace to store the physical log.

If you created a server during installation, the buffer pool, logical log, and physical log are configured for automatic extension.

The following automatic tuning options are enabled by default. You can control whether the options are enabled.

- Increase the number of CPU virtual processors to half the number of CPU processors to ensure optimum performance. Control with the **auto_tune_cpu_vps** task in the Scheduler.
- Increase the number of AIO virtual processors and page cleaner threads increase I/O capability. Control with the AUTO_TUNE configuration parameter. Control with the AUTO_AIOVPS configuration parameter.
- Process read-ahead requests to reduce the time to wait for disk I/O. Control with the AUTO_TUNE configuration parameter. Control with the AUTO_READAHEAD configuration parameter.
- Trigger checkpoints as frequently as necessary and add logical log files as needed to avoid the blocking of transactions. Control with the AUTO_TUNE and the DYNAMIC_LOGS configuration parameters. Control with the AUTO_CKPTS and the DYNAMIC_LOGS configuration parameters.
- Tune LRU flushing to improve transaction throughput. Control with the AUTO_TUNE configuration parameter. Control with the AUTO_LRU_TUNING configuration parameter.
- Reoptimize SPL routines and reprepare prepared objects after the schema of a table is changed to prevent manual processes and errors. Control with the AUTO_TUNE configuration parameter. Control with the AUTO_REPREPARE configuration parameter.

- Updates statistics that are stale or missing at scheduled intervals to improve query performance. Control with Auto Update Statistics tasks in the Scheduler and the AUTO_TUNE configuration parameter. Control with Auto Update Statistics tasks in the Scheduler and with the AUTO_STAT_MODE configuration parameter.
- Run light scans on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data. Control with the BATCHEDREAD_TABLE configuration parameter.
- Fetch a set of keys from an index buffer to reduce the number of times that a buffer is read. Control with the BATCHREAD_INDEX configuration parameter.
- Increase shared memory caches to improve query performance. Control with the DS_POOLSIZE, PC_POOLSIZE, PLCY_POOLSIZE, and USRC_POOLSIZE configuration parameters.

Feature configuration

You can configure the database server to support the types of optional functionality that you need.

The following features are often enabled:

Parallel database queries

You can control the resources that the database server uses to perform decision-support queries in parallel. You must balance the requirements of decision-support queries against the requirements of online transaction processing (OLTP) queries. The resources that you must consider include shared memory, threads, temporary table space, and scan bandwidth.

Data replication

Data replication is the process of representing database objects at more than one distinct site.

High-availability cluster configurations consist of a primary server and one or more secondary servers that contain the same data as the primary server. High-availability clusters can provide redundancy, failover, workload balancing, and scalability. You can direct connections from applications to cluster servers with Connection Manager.

Enterprise Replication replicates all or a specified subset of the data between geographically distributed database servers. You can define set of replication servers as a grid to administer and run queries across the servers. You can combine a high-availability cluster and Enterprise Replication on the same database server.

Auditing

If you enabled role separation when you installed the database server, you can audit selected activities. To use database server auditing, you must specify where audit records are stored, how to handle error conditions, and other configuration options. You also might want to change how users are audited if you suspect that they are abusing their access privileges.

Security

You can keep your data secure by preventing unauthorized viewing and altering of data or other database objects. Use network encryption to encrypt data that is transmitted between servers and clients, and between servers. You can use column-level encryption to store sensitive data in an encrypted format. You create secure connections to the database server with authentication and authorization processes. You can encrypt storage spaces. Discretionary access control verifies whether the user who is attempting to perform an operation is granted the required privileges to perform that operation. You can use label-based access control (LBAC) to control who has read access and who has write access to individual rows and columns of data.

Distributed queries

You can use the database server to query and update multiple databases across multiple database servers or within the same database server instance. HCL® OneDB® uses a two phase commit protocol to ensure that distributed queries are uniformly committed or rolled back across multiple database servers.

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.

Connectivity configuration

The connectivity information allows a client application to connect to the database server on the network. You must prepare the connectivity information even if the client application and the database server are on the same computer or node.

HCL OneDB™ client/server connectivity information, the sqlhosts information, includes the database server name, the type of connection that a client can use to connect to the database server, the host name of the computer or node on which the database server runs, and the service name by which it is known. You do not need to specify all possible network connections in the sqlhosts information before you start the database server. However, to make a new connection available you must shut down the database server and then restart it.

On UNIX™, the `sqlhosts` file contains connectivity information. On Windows™, the SQLHOSTS registry key contains connectivity information.The `sqlhosts` file contains connectivity information. You might also need to modify other connectivity and security files, depending on your needs.

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 client programs that are included in the HCL OneDB™ Client Software Development Kit (Client SDK), such as DB-Access, or API drivers.

Limit session resources

You can limit the resources available to individual sessions to more evenly distribute system usage, and prevent resource monopolization.

Set the following configuration parameters to impose limits on sessions:

- SESSION_LIMIT_LOCKS limits the number of locks a tenant session can acquire.
- SESSION_LIMIT_MEMORY limits the amount of memory that can be allocated for a session.
- SESSION_LIMIT_TEMPSPACE limits the amount of temporary table space that can be allocated for a session.

- SESSION_LIMIT_LOGSPACE limits the size of transactions within a session, based on the amount of log space that an individual transaction would fill.
- SESSION_LIMIT_TXN_TIME limits the amount of time that a transaction is allowed to run within a session.

You can set the IFX_SESSION_LIMIT_LOCKS environment option in the session, to specify a lower lock limit than the SESSION_LIMIT_LOCKS configuration parameter value.

Session limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Transactions and sessions that exceed a set limit are terminated by the **session_mgr** thread. The **session_mgr** thread starts when the database server starts, and remains inactive until a session limit is exceeded.

Automate startup and shutdown on UNIX™

You can modify startup and shutdown scripts on UNIX™ to automatically start and shut down the database server.

UNIX™ startup script

Modify the UNIX™ startup script to start the database server automatically when your computer enters multiuser mode.

1. Add UNIX™ and database server utility commands to the UNIX™ startup script so that the script performs the following actions:
    - Sets the **ONEDB_HOME** environment variable to the full path name of the directory in which the database server is installed.
    - Sets the **PATH** environment variable to include the `$ONEDB_HOME/bin` directory.
    - Sets the **ONEDB_SERVER** environment variable so that the **sysmaster** database can be updated (or created, if necessary).
    - Runs the oninit command, which starts the database server and leaves it in online mode.
2. If you plan to start multiple versions of the database server (multiple residency), you must add commands in the script to set the **ONCONFIG** and **ONEDB_SERVER** environment variables and run the oninit command for each instance of the database server.
3. If different versions of the database server are installed in different directories, you must add commands to the script to set the **ONEDB_HOME** environment variable and repeat the preceding steps for each version.

UNIX™ shutdown script

Modify your UNIX™ shutdown script to shut down the database server in a controlled manner whenever UNIX™ shuts down. The database server shutdown commands run after all client applications complete transactions and exit.

1. Add UNIX™ and database server utility commands to the UNIX™ shutdown script so that the script performs the following tasks:
    - Sets the **ONEDB_HOME** environment variable to the full path name of the directory in which the database server is installed.
    - Sets the **PATH** environment variable to include the `$ONEDB_HOME/bin` directory.

    - Sets the **ONCONFIG** environment variable to the appropriate configuration file.
    - Runs the onmode -ky command, which initiates an immediate shutdown and takes the database server offline.
2. If you are running multiple versions of the database server (multiple residency), you must add commands in the script to set the **ONCONFIG** environment variable and run the onmode -ky command for each instance.
3. If different versions of the database server are installed in different directories, you must add commands to the script to set the **ONEDB_HOME** environment variable and repeat the preceding steps for each version.

Automate startup on Windows™

You can automate startup of the database server on Windows™.

About this task

To start the database server automatically when Windows™ starts:

1. From the **Service** control application window, select the HCL OneDB™ 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.

What to do next

To stop automatic startup, clear the **Automatic** property.

Database server maintenance tasks

In addition to monitoring the database server for potential problems, regularly perform routine maintenance tasks to keep the server running smoothly and with optimum performance.

You can use the HCL OneDB™ command-line utilities to perform the following tasks. Not all of the following tasks are appropriate for every installation.

Backup 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 can create backups with the ON-Bar utility.

Check data for consistency

To ensure that data is consistent, perform occasional checks.

Manage logical logs

To ensure database server performance, perform logical-log administration tasks, such as, 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 blocking user transactions.

Manage the physical log

To ensure database server performance, make sure that you allocate enough space for the physical log. You can change the size and location of the physical log. When the database server starts, it checks whether the physical log is empty because that implies that the server shut down in a controlled fashion. If the physical log is not empty, the database server automatically performs a *fast recovery*. Fast recovery automatically restores the 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

To ensure that the database server has the appropriate amount of shared memory to maintain performance goals, perform 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
- Changing forced residency (on or off, temporarily or for a session)
- Tuning checkpoint intervals
- Adding segments to the virtual portion of shared memory
- Configuring the SQL statement cache to reduce memory usage and preparation time for queries

Manage virtual processors

To ensure database server performance, configure enough virtual processors (VPs). The configuration and management of 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.

Manage the database server message log

To ensure that message log space does not fill, monitor the size of the database server message log. The database server appends new entries to this file. You can enable the automatic rotating of the database server message log to limit the total size of the log files.

Client/server communication

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

Client/server architecture

HCL® OneDB® 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.

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 1. Locations HCL OneDB™ 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.

Database server connections

A client application establishes a connection to a database server with either the CONNECT or DATABASE SQL statement.

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.

Supporting multiplexed connections

A *multiplexed connection* uses a single network connection between the database server and a client to handle multiple database connections from the client.

About this task

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.
   Example
   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.
   Example
   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` entry that the client uses for the database server connection.
   Example
   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.

Example

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` information 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` entries 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 sqlbreak() function is not supported during a multiplexed connection.

Connections that the database server supports

The database server supports different types of connections with client application.

About this task

The following types of connections are supported by the database server.

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

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. Add an `sqlhosts` entry to define a dbserver group for your database server.

Local connections

A *local connection* is a connection between a client and the database server on the same computer.

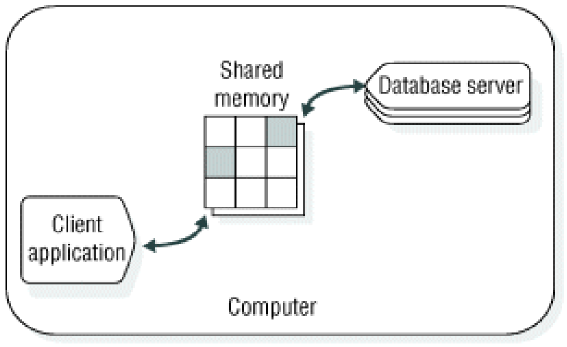
The following topics describe different types of local connections.

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 1. Client application and a database server communication through a shared-memory connection.

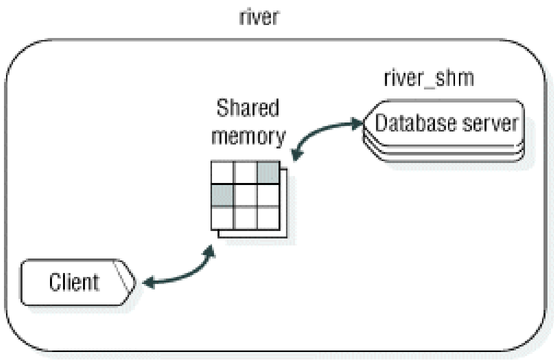


Shared memory provides fast access to a database server, but it poses some security risks. Errant or malicious applications might delete or view message buffers of their own or of other local users. Shared-memory communication is also vulnerable to programming errors if the client application performs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

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.

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

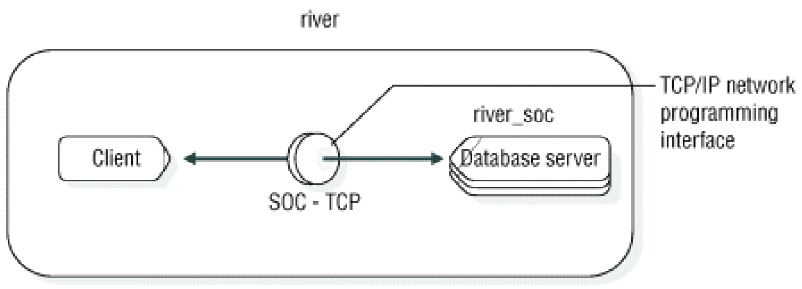
A network connection between a client application and a database server on the same computer is called a *local-loopback* connection.

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 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      riverol
```

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.

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

Windows: On the database server, the connectivity information is stored in the `sqlhosts` file; however, on clients the connectivity information is stored in the SQLHOSTS registry.

Windows: The connectivity information is stored in the SQLHOSTS registry, not in the `sqlhosts` file.

Network-configuration files

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

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 HCL® OneDB® client/server products 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, the HCL® OneDB® 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

  HCL® OneDB® 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 (`/`) character

  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 HCL® OneDB® 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 must support a TCP/IP or an IPCSTR connection with the coordinator, even if both database server instances 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.

The `hosts` and `services` files must be available to each computer that runs HCL® OneDB® client/server products.

UNIX:

- The `hosts` and `services` files are in the `/etc` directory.
- 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
  ```

Windows:

- The `hosts` and `services` files are in `%WINDIR%\system32\drivers\etc\`.
- You might want to 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.

TCP/IP connectivity files (UNIX™)

The `hosts` and `services` files are in the `/etc` directory.

Connectivity files must be available to each computer that runs an HCL® OneDB® client/server product.

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™)

The `hosts` and `services` network-configuration files prepare the SQLHOSTS registry key for the TCP/IP network protocol.

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, information is read on both the client side and server side.

The following information is read on the client side:

- The **ONEDB_SERVER** environment variable.
- The `hosts` file information (ONEDB_ SQLHOSTS environment variable, `$ONEDB_HOME/etc/sqlhosts` file, 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.

Multiple TCP/IP ports

You can modify the services file to take advantage of having multiple Ethernet cards.

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
  ```
- Add one `sqlhosts` 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 **ONEDB_SERVER** environment variable provides.

Network security files

HCL® OneDB® 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.

Trusted-host information

Users on trusted hosts are allowed to access the local system without supplying a password. You can include an optional user name to limit the authentication to a specific user on a specific host.

Use one of the following *trusted-hosts files* to specify remote hosts for rlogin, rsh, rcp, and rcmd remote-authentication:

- `hosts.equiv`
- The file that is specified by a database server's REMOTE_SERVER_CFG configuration parameter

Use trusted-hosts information only for client applications that do not supply a user account or password. 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.

To use trusted-host information for authentication, specify the `s=1` or `s=3` options in `sqlhosts` file entries. If you do not specify an `s` option, `s=3` is the default.

On Windows™, the trusted-host file is in the `\%WINDIR%\system32\drivers\etc` directory.

On Linux™ and UNIX™ systems, the trusted-host file is in the `$ONEDB_HOME/etc/` directory.

The `hosts.equiv` file has the following requirements:

- It must be owned by user **informix**
- It belong to group **informix**
- Permissions on the file must be restricted so that only user **informix** can modify the file. Using octal permissions, one of the following values is appropriate:
  - 644
  - 640
  - 444
  - 440

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

```
rlogin hostname
```

If you log-in successfully without receiving a password prompt, the client is trusted. This method of determining if a client is trusted does not work when the file specified by the REMOTE_SERVER_CFG configuration parameter is used

Trusted-host file entries

To avoid an extra DNS lookup, specify the host name both with and without the domain name. For example, if the trusted host is named **host1** and it is in the domain **example.com**, then add the following entries to the trusted-host file:

```
#trustedhost       username
host1              informix
host1.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) **host2.example.com** might refer to itself with the local host name **viking**. If this situation occurs, specify both host-name formats:

```
#trustedhost
host2.example.com
viking
```

Using the file specified by the REMOTE_SERVER_CFG configuration parameter instead of the hosts.equiv file

In the following situations, use the REMOTE_SERVER_CFG configuration parameter and the file that the parameter specifies:

- 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.

To add entries to the file specified by the REMOTE_SERVER_CFG configuration parameter, you can manually enter the information or you can run the admin() or task() function with the **cdr add trustedhost** argument. If you run **cdr add trustedhost** argument with the admin() or task() function on a server in a high-availability cluster, the trusted-host information is added to the trusted-host files of all database servers in the cluster. Do not run the admin() or task() function with the **cdr list trustedhost** argument if you have manually entered trusted-host information on any of the database servers in a high-availability cluster or Enterprise Replication domain.

Trusted-user information

A user can list hosts from which they can connect as a trusted user in their `.rhosts` file.

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` entry. If you do not specify an `s` option, `s=3` is the default.

There may be reasons why a users `.rhosts` file cannot be used. For example, a non-root installation might not have read access to a specific users `.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
```

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.

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.

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™: 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

User impersonation

The database server must impersonate the client to run a process or program on behalf of the client for certain client queries or operations.

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

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

The sqlhosts file (UNIX™)

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

If you store the information in another location, you must set the **ONEDB_ SQLHOSTS** 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 the **ONEDB_ SQLHOSTS** environment variable
- In separate `sqlhosts` files in each database server 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 HCL OneDB™ 64-bit product installed on a Windows™ 64-bit system or an HCL OneDB™ 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 HCL OneDB™ 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 HCL OneDB™ 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:

- regedt32
- Setnet32

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

The default location of the `sqlhosts` file is:

UNIX™:

```
$ONEDB_HOME/etc/sqlhosts
```

Windows™:

```
%ONEDB_HOME%\etc\sqlhosts.%ONEDB_SERVER%
```

If you store the information in another location, you must set the **ONEDB_ SQLHOSTS** 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 the **ONEDB_ SQLHOSTS** environment variable
- In separate `sqlhosts` files in each database server directory

Creating the sqlhosts file with a text editor (UNIX™)

Each computer that hosts a database server or a client must have an `sqlhosts` file.

Before you begin

The `sqlhosts` file is located, by default, in the `$ONEDB_HOME/etc` directory. As an alternative, you can set the **ONEDB_ SQLHOSTS** environment variable to the full path name and file name of a file that contains the `sqlhosts` information.

About this task

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.

Example

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 the HCL OneDB™ 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 HCL OneDB™ 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.

Before you begin

If you specify a shared SQLHOSTS registry key, you must set the ONEDB_ SQLHOSTS 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 ONEDB_ SQLHOSTS computer. If the database server does not find an SQLHOSTS registry key on the ONEDB_ SQLHOSTS computer, or if ONEDB_ SQLHOSTS 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 HCL OneDB™ Setnet32 window opens.
2. Click the **Server Information** tab to display the **Server Information** page, which has the following elements:
   Choose from:
   - **HCL OneDB™ Server**

     Select an existing HCL OneDB™ 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 *HCL OneDB™ Administrator's Guide*.

- **Make Default Server**

  Sets the ONEDB_SERVER 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.

The sqlhosts information

The `sqlhosts` information contains connectivity information for each database server and 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.

Each computer that hosts a database server or a client must include connectivity information. The information is stored in an `sqlhosts` file on UNIX™ operating systems, or in an SQLHOSTS registry key on Windows™ operating systems.The connectivity information is stored in an `sqlhosts` file, except on Windows™ clients, where it is stored in an SQLHOSTS registry key.

In the `sqlhosts` file, each row contains the connectivity information for one database server, or the definition for one group.

- The connectivity information for each database server includes four fields of required information and one field for options.
- The group definition contains information in only three of the fields.

In the registry, 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 that are used for the SQLHOSTS information.

Field name in the `sqlhosts` file	Field name in the SQLHOSTS registry key	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 keyword `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.
options	OPTIONS	Options that describe or limit the connection	Group options

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

The Internet Assigned Numbers Authority (IANA) assigns service names and port numbers for HCL® OneDB® database servers.

The services names and port numbers for database servers are:

Port/service	IANA code	Description
sqlexec	9088/tcp	HCL OneDB™ SQL Interface

These service names are created in the `sqlhosts.std` file of HCL OneDB™. You are not required to change installed HCL OneDB™ 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 HCL OneDB™.)

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 HCL OneDB™ using non-standard ports.

For more information, see the IANA organization website.

sqlhosts connectivity information

Fields in the `sqlhosts` file or SQLHOSTS registry key describe connectivity information.

Syntax

(explicit id unique_45_Connect_42_altfrag001) unique_45_Connect_42_altfrag001 { *dbservername* { *connection_type* | **`group`** } *hostnameservicename* [`<Options>`[]] }

Element	Purpose	Restrictions
dbservername	Names the database server for which the connectivity information is being specified. If specified with the group keyword instead of the connection type, names a group to treat multiple, related database server entries as one logical entry. You can use groups to establish or change client/server connections, or 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. The database server must exist. Its name must be specified by the DBSERVERNAME or DBSERVERALIASES configuration parameter in the `onconfig` file. A database server group cannot be nested inside another database server group. Database servers can be members of one group.
connection_type	Describes the type of connection that is 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. If the group keyword is specified, must be null (-).
servicename	Specifies the alias for the port number. The interpretation of the service name field depends on the type of connection in the connection-type field.	The field length is limited to 128 bytes. If the group keyword is specified, must be null (-).

dbservername field

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

If an `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` in the `sqlhosts` file and PROTOCOL in the SQLHOSTS registry key.

The following table summarizes the possible connection-type values for database server connections on different operating systems.

Table 2. Summary of connection-types

Values for UNIX™	Values for Windows™	Description	Connection type
drsoctcp	drsoctcp	Distributed Relational Database Architecture™ (DRDA®) - connection for .NET Core Provider 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 .NET Core Provider 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
onsoctcp	onsoctcp	Sockets with TCP/IP protocol	Network
onsqlmux	onsqlmux	Multiplexed connection	Network

Note: The connection-type 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` information.

Host name field

The host name is entered in the `hostname` field in the `sqlhosts` file, and in the **HOST** registry key.

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 that are 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.

UNIX™: Shared-memory and stream-pipe communication

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.

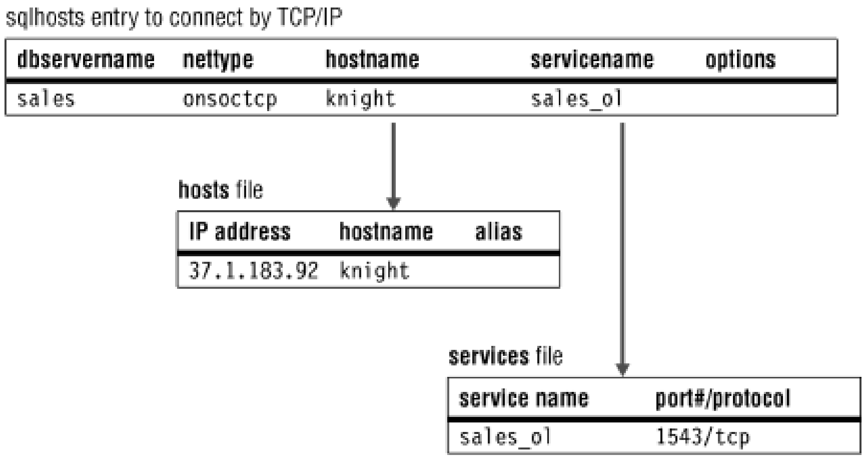
Service name field

Network communication with TCP/IP

The service name field is called `servicename` on the UNIX™ operating system and **SERVICE** on the Windows™ operating system. 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 following figure shows the relationship between the `sqlhosts` information and the `hosts` file, and the relationship of `sqlhosts` information to the `services` file.

Figure 4. Relationship of `sqlhosts` information to `hosts` and `services` files



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

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

Windows™: Named-pipe communication

For 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.

UNIX™: Shared-memory and stream-pipe communication

For 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

For multiplexed connections (`onsqlmux`), the `hostname` field must have an entry, but the entry is ignored. Dashes (-) can be used as entries.

sqlhosts file and SQLHOSTS registry key options

You can include server options and group options in the `sqlhosts` file or SQLHOSTS registry key.

The following syntax fragments show the server options. The syntax fragment for group options is described in a section after the server options.

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

Server options

```
“ [ , b = size ] ”

“ [ , cfd = filepath ] ”

“ [ , g = group ] ”

“ [ , k = { 0 | 1 } ] ”

“ [ , m = { 0 | 1 } ] ”

“ { [ <Lookup options> <PAM options> ] | [ , s=6 ] } ”
```

Lookup options

```
“ [ , r= { 0 | 1 } ] ”

“ [ , s= { 0 | 1 | 2 | 3 } ] ”
```

PAM options

```
“ , s=4, pam_serv =( name ), pamauth =( { challenge | password } ) ”
```

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

Element	Purpose	Restrictions
b	Specifies, in bytes, the size of the communications buffer space for TCP/IP connections.	The maximum buffer size supported is 32 KB.
cfd	Indicates the storage location for communication files that are used in shared-memory and stream-pipe connections.	The length of the cfd path is restricted to 70 bytes. Relative-path byte lengths include `$ONEDB_HOME`.
g	Specifies the name of the group to which the database server belongs.	The group must be defined.

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

Element	Purpose	Restrictions
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 more network connections.	• Multithreaded client connections, shared-memory connections, and connections to subordinate database servers are not supported. • The sqlbreak() function is not supported.
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.
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 that is 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

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

Element	Purpose	Restrictions
		• Enterprise Replication and high-availability clusters • A multiplexed connection

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

Group options

```
“ [ c= { 0  | 1 } ] ”

“ [ , e=server ] ”

“ [ , i=identifier ] ”
```

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

Element	Purpose	Restrictions
c	Controls connection redirection. Indicates the order in which client applications choose database servers, or the aliases within a database server group.	
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. The server automatically recognizes any changes that are made.

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.

Attention: 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=16000 on a system that has 1000 users, the system might require 16 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.

Group connection-redirection option (c)

The c option is valid only for servers that are 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 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` information. If the client cannot connect to the first instance, it attempts to connect to the second instance, and so on.

Table 5. Settings for the connection-redirection option. (continued)

Setting	Result
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 HCL OneDB™ 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 `$ONEDB_HOME` for storing communication files:

- cfd=/*location* defines an absolute path
- cfd=*location* defines a path relative to `$ONEDB_HOME`

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

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

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

End of group option (e)

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. 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 end-of-group option is specified for a group, the group members are assumed to be contiguous. The end of the 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 that is specified by the parameters of your operating system, the network service immediately detects the broken connection and frees resources.

Table 6. 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 more network connections. You must restart the server after you enable this service.

Table 7. Settings for the multiplex option

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

PAM options (pam_serv, pam_auth, s=4)

The database server provides an interface to use pluggable authentication modules (PAM) for session authentication. To configure this interface, supply the PAM service name and the authentication method. Authentication can be the connection password or a user challenge that requires the user to answer a question. Distributed Relational Database Architecture (DRDA®) connections for .NET Core Provider clients can use password authentication, but not challenge authentication.

HCL OneDB™ PAM authentication calls the pam_authenticate() and pam_acct_mgmt() functions.

Table 8. Settings for PAM services

Option	Description	Settings
pam_	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.

Table 8. Settings for PAM services (continued)

Option	Description	Settings
serv		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 that is used by the PAM service. 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
s=4	Enables PAM authentication.	

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 9. 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.

Table 9. Settings for trusted-host and trusted-user lookup. (continued)

Setting	Result
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 information ensures that the connections between cluster servers are trusted. Secure ports that are listed in the sqlhosts information can be used only for cluster communication. Client applications cannot connect to secure ports.

Table 10. 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 11. Settings for netrc lookup options.

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

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 or Connection Manager SLA entries as a single entity for client connections to simplify connection redirection to database servers or Connection Managers. You must create group entries for database servers that participate in Enterprise Replication.

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

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

Use a dash (-) character (ASCII 45) for hostname and server/port values when you specify a connection information for a group.

High-availability cluster groups

A high-availability cluster groups `sqlhosts` have the following format:

```
#dbservername   nettype   hostname     servicename        options
 group_name      group     -            -                  c=1,e=member_name_n
 member_name_1   protocol  host_name_1  service_or_port_1  g=group_name
 member_name_2   protocol  host_name_2  service_or_port_2  g=group_name
 member_name_n   protocol  host_name_n  service_or_port_n  g=group_name
```

`C=1` is optional, and specifies that a random starting point in the list of group members is used for connection attempts. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

Enterprise Replication server groups

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. Enterprise Replication node groups have the following `sqlhosts` format:

```
#dbservername   nettype   hostname     servicename        options
 group_name_1    group     -            -                  i=identifier_1,e=member_name_1
 member_name_1   protocol  host_name_1  service_or_port_1  g=group_name_1

 group_name_2    -         -                               i=identifier_2,e=member_name_2
 member_name_2   protocol  host_name_2  service_or_port_2  g=group_name_2

 group_name_n    -         -                               i=identifier_n,e=member_name_n
 member_name_n   protocol  host_name_n  service_or_port_n  g=group_name_n
```

The `i=identifier` is required for Enterprise Replication. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

Connection Manager service-level agreement groups

Connection Manager SLA groups have the following `sqlhosts` format:

```
#dbservername          nettype   hostname   servicename             options
SLA_1_group_name        group     -          -                       c=1,e=SLA_name_1_from_CM_n
SLA_name_1_from_CM_1    protocol  CM_1_host  CM_1_port_or_service_1  g=SLA_1_group_name
SLA_name_1_from_CM_2    protocol  CM_2_host  CM_2_port_or_service_1  g=SLA_1_group_name
SLA_name_1_from_CM_n    protocol  CM_n_host  CM_n_port_or_service_1  g=SLA_1_group_name

SLA_2_group_name        group     -          -                       c=1,e=SLA_name_2_from_CM_n
SLA_name_2_from_CM_1    protocol  CM_1_host  CM_1_port_or_service_2  g=SLA_2_group_name
```

```
SLA_name_2_from_CM_2   protocol   CM_2_host   CM_2_port_or_service_2   g=SLA_2_group_name
SLA_name_2_from_CM_n   protocol   CM_n_host   CM_n_port_or_service_2   g=SLA_2_group_name

SLA_n_group_name        group      -           -                        c=1,e=SLA_name_n_from_CM_n
SLA_name_n_from_CM_1   protocol   CM_1_host   CM_1_port_or_service_n   g=SLA_n_group_name
SLA_name_n_from_CM_2   protocol   CM_2_host   CM_2_port_or_service_n   g=SLA_n_group_name
SLA_name_n_from_CM_n   protocol   CM_n_host   CM_n_port_or_service_n   g=SLA_n_group_name
```

`c=1` is optional, and specifies that a random starting point in the list of group members is used for connection attempts. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

Creating a group in the sqlhosts file (UNIX™)

You can define a 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 members of the group. Include the g=*group* option.

Example

Example

The following example shows definition of a group named **g_asia**. The group contains four members.

```
#dbservername  nettype   hostname  servicename  options
g_asia         group     -         -            c=1,e=manilla
tokyo          onsoctcp  node_1    service_1    g=g_asia
beijing        onsoctcp  node_2    service_2    g=g_asia
seoul          onsoctcp  node_3    service_4    g=g_asia
manilla        onsoctcp  node_4    service_5    g=g_asia
```

Setting up the database server group registry key Windows™

The SQLHOSTS registry key fields must be set to during database server setup.

About this task

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 48 and 8 on page 48 for the remaining fields of the SQLHOSTS registry key.
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: 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: 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.

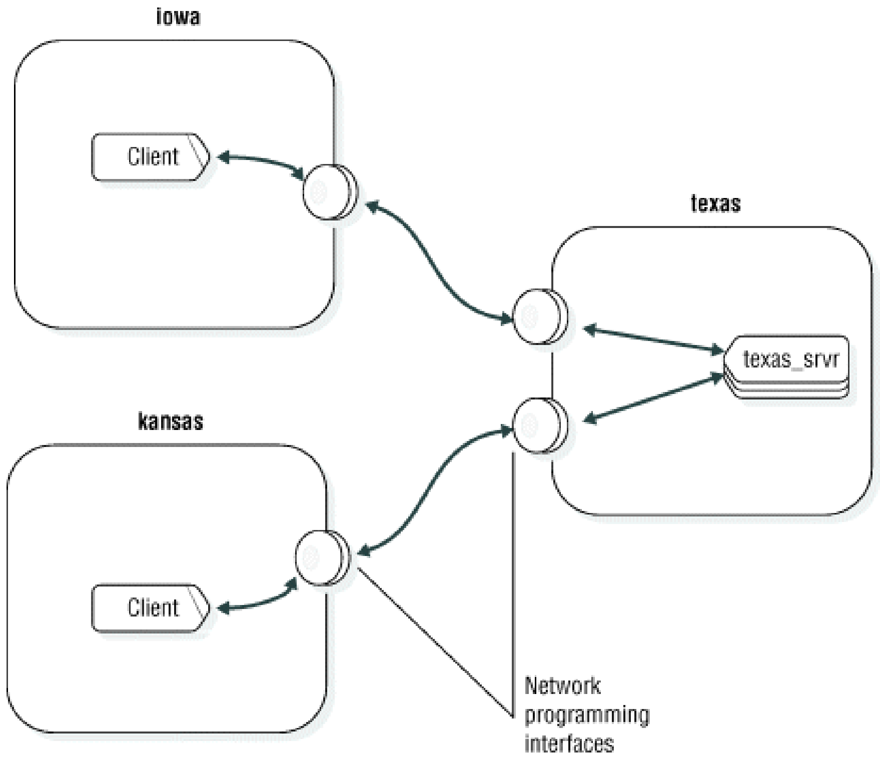
```
#address       hostname   alias
123.45.67.81   texas1
123.45.67.82   texas2
```

If the client application and database server share the `sqlhosts` information, 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 5. 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.

HCL OneDB™ support for IPv6 addresses

On all platforms, HCL OneDB™ 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 HCL OneDB™ 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. HCL OneDB™ and Client SDK retrieve the IP address from the name service.

You can treat HCL OneDB™ 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 HCL OneDB™ 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 HCL OneDB™ 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 HCL OneDB™ 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

HCL OneDB™ 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 `$ONEDB_HOME/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
  ```

Configuration 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

UNIX: When you configure connectivity, also consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTIONS configuration parameters. These parameters can 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 *HCL OneDB™ Security Guide*.

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 the `onconfig` file or Windows™ registry key:

```
DBSERVERNAME nyc_research
```

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

- In the **ONEDB_SERVER** 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
```

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
```

Because each dbserver name has a corresponding `sqlhosts` entry, 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:

```
DBSERVERALIASESname-number,name-number
```

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 HCL OneDB™. If you specify a maximum number, you can also specify whether you want HCL OneDB™ 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

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 97. For information about the NETTYPE configuration parameter, see the *HCL OneDB™ 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 OneDB 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 HCL OneDB™ 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

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 HCL OneDB™ virtual processors (VPs).

Specifying NUMFDSERVERS information is useful if HCL OneDB™ 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.

Connection information set in the HA_ALIAS configuration parameter

The HA_ALIAS configuration parameter defines a network alias that is used for server-to-server communication in a high-availability cluster. The specified network alias is also used by Connection Managers, the ifxclone utility, and onmode -d commands.

Set the HA_ALIAS configuration parameter for each server in a high-availability cluster. The HA_ALIAS configuration parameter is required for high-availability cluster servers that use shared-memory connections.

The HA_ALIAS configuration parameter value must match a DBSERVERNAME or DBSERVERALIASES configuration parameter value that is associated with a TCP `sqlhosts` file entry.

Environment variables for network connections

The **CONNECT_TIMEOUT** (connect time) and **CONNECT_RETRIES** (connect retry) environment variables 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 **ONEDB_ SHMBASE** (shared-memory base).

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

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.

Before you begin

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

About this task

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';
```

Distributed Relational Database Architecture™ (DRDA®) communications

DRDA is a set of protocols that enables multiple database systems and application programs to work together.

This section contains information about how to configure HCL OneDB™ to use the Distributed Relational Database Architecture™ (DRDA®).

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 is exchanged and the exchange method.

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

Connection Managers support DRDA®, so you can use connection management to redirect client connection requests to appropriate database servers. Connection Managers can also provide automatic failover for high-availability clusters using DRDA®.

Enterprise Replication, data replication, and HCL OneDB™ utilities, such as DB-Access, require standard HCL OneDB™ connections. Enterprise Replication utilities do not operate over DRDA® connections. However, Enterprise Replication connections can coexist with DRDA® connections.

When you use DRDA® with ANSI-compliant databases, unbuffered logging and implicit transactions are enforced. If you migrate an application that is based on a non-ANSI-compliant database to a DRDA® environment, the connection must handle application logic for statements that need transactions. For example, a BEGIN WORK statement is required before a concatenation operator in a stored procedure.

You can secure DRDA® connections between a common client API and HCL OneDB™ in the following ways:

- Encrypted password security or an encrypted user ID and encrypted password security
- Password authentication through a pluggable authentication module

You can secure DRDA® connections between a common client API and HCL OneDB™ with password authentication through a pluggable authentication module.

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.

About this task

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.

Example

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.

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`. HCL OneDB™ 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.

The DRDAEXEC thread and queries from clients

For every DRDA® client, HCL OneDB™ 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.

SQL and supported and unsupported data types

When using DRDA®, HCL OneDB™ syntax is supported over the common API. When using DRDA® connections, HCL OneDB™ rounds decimal and money values to 32-digit precision for all data retrieval operations on decimal or money data types.

The following data types are supported over the common API:

- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BSON
- BYTE
- CHAR(32k)
- CLOB
- DATE
- DATETIME
- DECIMAL

- FLOAT
- INT
- INT8
- INTERVAL
- JSON
- LVARCHAR(32k)
- MONEY
- NCHAR(32k)
- NVARCHAR(255)
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT
- VARCHAR(255)

HCL OneDB™ 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

Display DRDA® connection information

Use onstat and onmode commands to display information that includes the DRDA® thread name and an indicator that distinguishes SQLI and DRDA® sessions

The following commands display information about the thread name and session:

- 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.

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.

Examples of client/server configurations

The next several sections show the correct `sqlhosts` entries 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

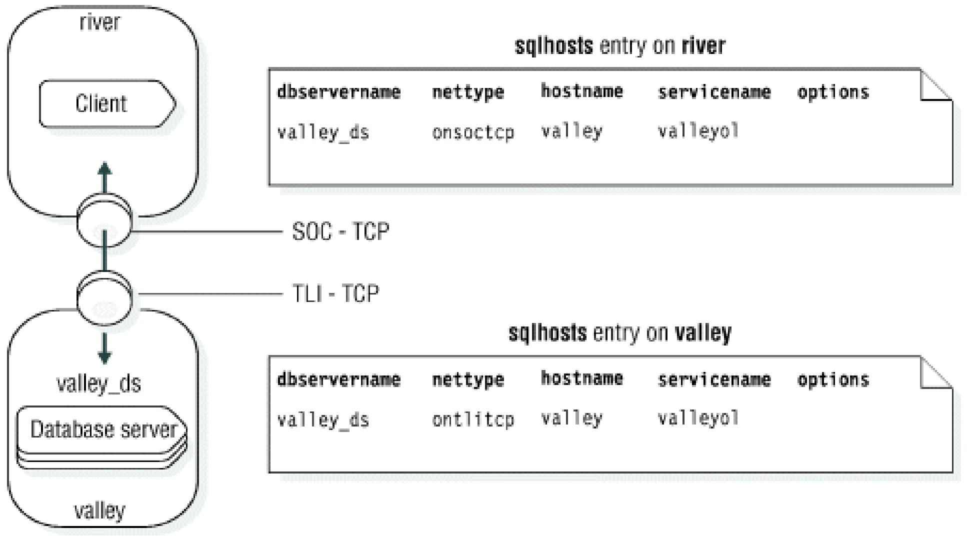
Examples of shared-memory and local-loopback connections can be found with the explanation of shared memory and local-loopback connections.

A network connection

The following figure shows a network-connection configuration for two database servers.

The client application is on host **river** and the database server is on host **valley**.

Figure 6. An example of a network client/server configuration


river
Client
SOC - TCP
TLI - TCP
valley_ds
Database server
valley
sqlhosts entry on river
dbservername nettype hostname servicename options
valley_ds onsoctcp valley valleyol
sqlhosts entry on valley
dbservername nettype hostname servicename options
valley_ds ontlitcp valley valleyol

An `sqlhosts` entry for the **valley_ds** database server is defined on both computers.

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`.

Multiple connection types

A single instance of the database server can provide more than one type of connection.

The following figure illustrates a configuration with more than one type of connection. 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.

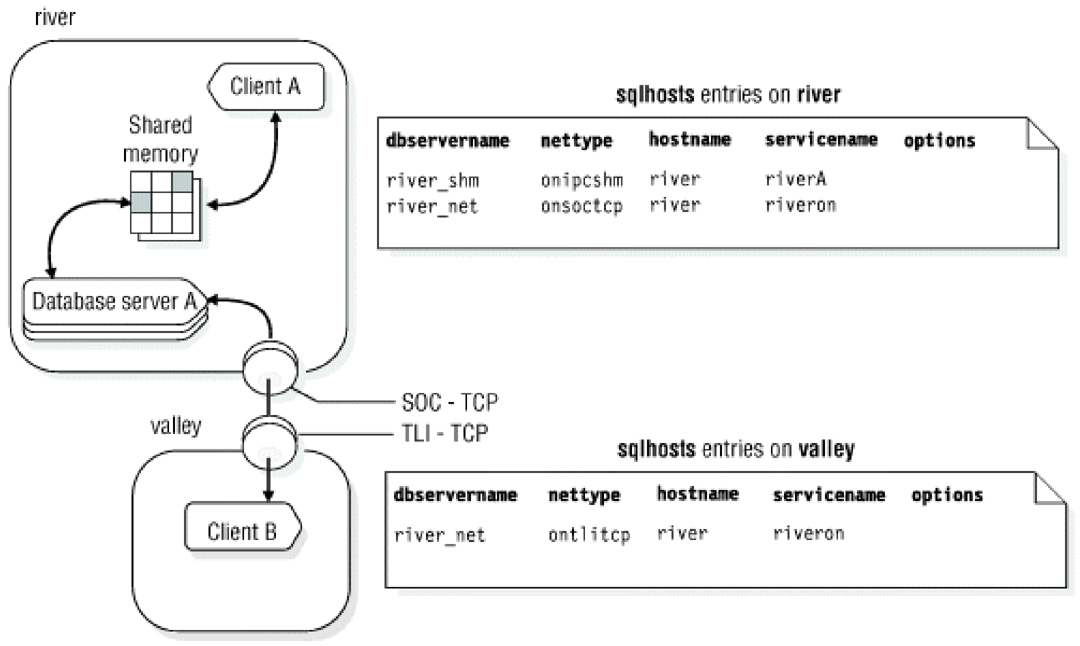
When you want the database server to accept more than one type of connection, you must take the following actions:

- Add DBSERVERNAME and DBSERVERALIASES entries in the `onconfig` file.
- Add an `sqlhosts` entry 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 7. An example of a UNIX™ client/server configuration that uses multiple connection types



sqlhosts entries on **river**

dbservername	nettype	hostname	servicename	options
river_shm	onipcshm	river	riverA	
river_net	onsoctcp	river	riveron	

sqlhosts entries on **valley**

dbservername	nettype	hostname	servicename	options
river_net	ontlitcp	river	riveron	

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, 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'
```

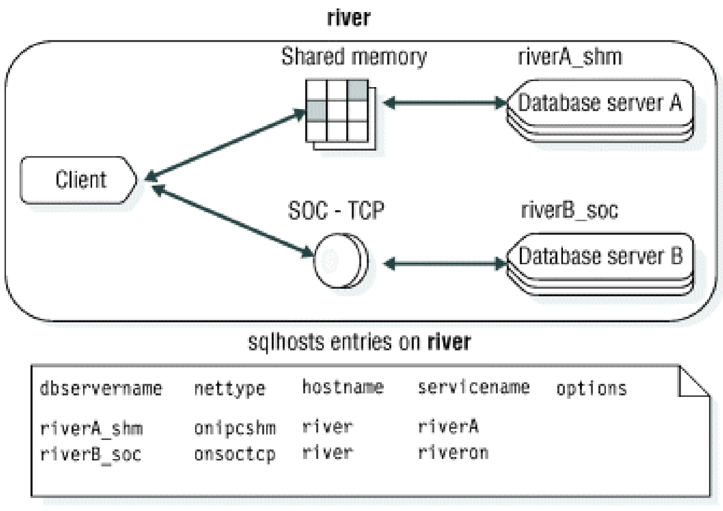
In the `sqlhosts` file, the **nettype** value associated with **river_net** specifies a network (TCP/IP) connection, so Client B uses a network connection.

Accessing multiple database servers

When more than one database server is active on a single computer, it is known as *multiple residency*.

The following figure shows a configuration with two database servers on host **river**.

Figure 8. Multiple database servers on UNIX™



```
dbservername   nettype    hostname   servicename   options

riverA_shm     onipcshm   river      riverA
riverB_soc     onsoctcp   river      riveron
```

For the configuration in previous example, you must prepare an `onconfig` file for database server **A** and another one for database server **B**. The `sqlhosts` file 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
```

Database server initialization

The database server requires both disk-space initialization and shared-memory initialization.

Shared-memory 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.

Disk-space initialization

Disk-space initialization uses the values that are stored in the configuration file to create the initial chunk of the root dbspace on disk. You use the oninit -i command from the command line to initialize disk space. 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.

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 is deleted.

You can prevent accidental disk initialization by setting the FULL_DISK_INIT configuration parameter to 0. When this configuration parameter is set to 0, the oninit -i command fails if the root dbspace exists.

The key difference between shared-memory initialization and disk-space initialization is that shared-memory initialization has no effect on disk-space allocation or layout. No data is deleted.

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 main tasks are completed during the two types of initialization:

- Process the configuration file on page 65
- Create shared-memory segments on page 66
- Initialize shared-memory structures on page 66
- Initialize disk space, if necessary on page 67
- Start all required virtual processors on page 67
- Make necessary conversions on page 67
- Start fast recovery and checkpoint, if necessary on page 67
- Document configuration changes on page 68
- Update the oncfg_servername.servernum file on page 68
- Change to quiescent mode on page 68
- Drop temporary tblspaces (optional) on page 68
- Set forced residency, if requested on page 68
- Change to online mode and return control to user on page 68
- Create or update SMI tables as necessary on page 69
- Monitor maximum number of user connections at each checkpoint on page 69

Process the configuration file

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, which specifies the `onconfig` file that contains your configuration parameters, must be set 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 uses the default values that are built in the server.If you omit any configuration parameters in your `onconfig` file, the database server reads the configuration values from the `$ONEDB_HOME/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 segments

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 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, which is used for data buffers and internal tables
- Virtual portion, used for most system and user-session memory requirements
- IPC communication portion, which is 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 122.

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 shared-memory structures

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 that is required for the logical-log buffer, initializes the structures, and links together the three individual buffers that form the logical-log buffer.

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, if necessary

Disk space is initialized only when you start the server for the first time or when you run the oninit -i command. Disk space is not initialized 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.

If the DISK_ENCRYPTION configuration parameter is set, the root dbspace is encrypted.

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.

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 that are started for making connections. For more information about virtual processors, see Virtual processors on page 76.

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.

Start fast recovery and checkpoint, if necessary

The database server checks if fast recovery is required and, if so, starts it. Fast recovery is not performed during disk-space initialization because there is not yet anything to recover.

For more information about fast recovery, see Fast recovery on page 334.

After fast recovery completes, the database server runs 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 332.

Document configuration changes

The database server compares the current values that are 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.

Update 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 `$ONEDB_HOME/etc` directory on UNIX™ or in your `%ONEDB_HOME%\etc` directory on Windows™. The database server uses the `oncfg_`*`servername.servernum`* file during a full-system restore for salvaging the logical log.

Change to quiescent mode

The database server now moves to quiescent mode or online mode, depending on how you started the initialization or database-server restart process.

Drop temporary tblspaces (optional)

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 182.

This task is performed when the database server is restarted; it is not performed during disk-space initialization.

If you use the -p option of oninit to initialize the database server, the database server skips this step.

Set forced residency, if requested

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.

Change to online mode and return control to user

Control returns to the user when the database server writes the `HCL OneDB™ Dynamic Server initialized - complete disk initialized` message in the message log only if initialization, not database-server restart, occurred and dynamically allocates a virtual shared-memory segment.

The server returns control to the user. Any error messages that are generated by the initialization procedure are displayed in the following locations:

- The command line
- The database server message log file, which is specified by the MSGPATH configuration parameter

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.

Create or update SMI tables as necessary

The database server creates the system-monitoring interface (SMI) tables and other system databases if they do not exist.

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 re-creates the **sysmaster** database during conversion. For more information about SMI tables, see the chapter on the **sysmaster** database in the *HCL OneDB™ 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.

The database server drops and re-creates the **sysutils** database during disk initialization, conversion. ON-Bar stores backup and restore information in the **sysutils** database. Wait until the message `sysutils database built successfully` displays in the message log.

The database server creates the **sysuser** database during initialization. The **sysuser** database is used for Pluggable Authentication Module (PAM) authentication in HCL OneDB™ server to server communication.

The database server creates the **sysadmin** database during initialization. The **sysadmin** database provides remote administration and scheduler API features in HCL OneDB™.

After the SMI tables and system databases are 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 105.

Monitor maximum number of user connections at each checkpoint

The database server prints the maximum number of user connections in the message log at each checkpoint in the following format: `maximum server connections` *`number`*. You can monitor the number of users who connected to the database server since the last restart or disk initialization.

The number that is 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 12. 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.

Table 12. Operating modes (continued)

Operating mode	Description	Users allowed access
		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.

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 `$ONEDB_HOME/bin/oninit` must be changed to allow public execute permission - root:informix:6755 in a standard HCL OneDB™ installation.

Windows™ only

Table 12: Operating modes on page 70 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 12: Operating modes on page 70 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 13. Changing operating modes in windows

Changing operating mode	Administrators group	HCL OneDB™-Admin group
command-line utilities such as **starts**		X
ISA		X
services control panel	X	

Changing database server operating modes

Use the oninit and onmode utilities to change from one database server operating mode to another. Use 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 HCL OneDB™ service for the `newyork` database server is:

```
HCL OneDB 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.

You can change database server modes in the following ways:

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.

UNIX: Run the oninit -s command.

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.

UNIX: Run the oninit command.

Windows: In the Services tool, select the database server service and click **Start**. Alternatively, 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.

Run the oninit -j command.

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.

Run the onmode -m command.

Windows: In 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.

Run the onmode -s or the onmode -sy command.

Windows: In 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.

Run the onmode -u or the onmode -uy command.

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. Also see Specifying administration mode users on page 75.

Run the onmode -j command.

Change from administration to online mode

When you move the database server from administration to online mode, all users can access the database server.

Run the onmode -m command.

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.

Run the onmode -s command.

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.

Run the onmode -k or the onmode -ky command.

Windows: In 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.

Specifying administration mode users

You can specify which users can connect to the database server in administration mode.

Temporary administration mode users

User **informix** or a DBSA can use the onmode -j -U or the oninit -U command to grant individual users access to the database server in administration mode for a session.

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 oninit -U command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Permanent administration mode users

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.

Disk, memory, and process management

Virtual processors and threads

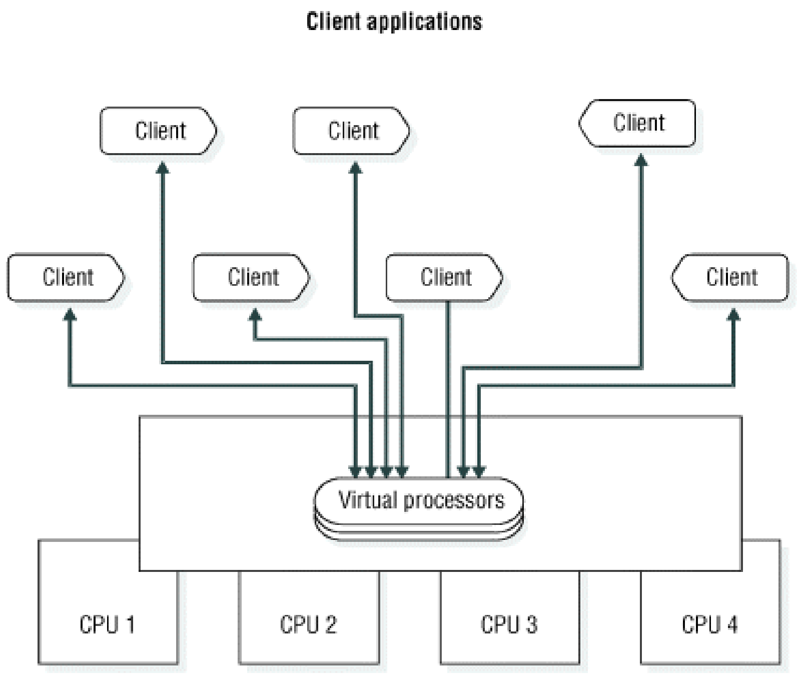
These topics describe virtual processors, explain how threads run within the virtual processors, and explain how the database server uses virtual processors and threads to improve performance.

Virtual processors

A virtual processor is a process that the operating system schedules for processing.

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.

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 9. 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 78.

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 *HCL OneDB™ Performance Guide*.

Advantages of virtual processors

A virtual processor provides a number of advantages.

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 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 80.

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.

In the following cases, virtual processors of the CPU class can run multiple session threads, working in parallel, for a single client:

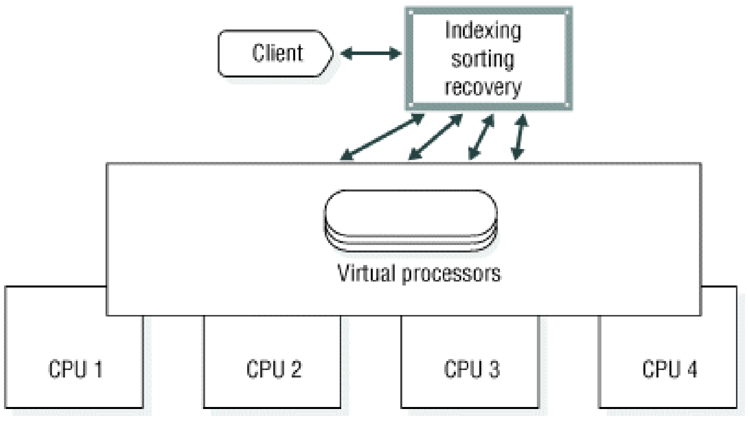
- Index building
- Sorting
- Recovery
- Scanning
- Joining

- Aggregation
- Grouping
- User-defined-routine (UDR) execution

For more information about parallel UDR execution, see *HCL OneDB™ User-Defined Routines and Data Types Developer's Guide*.

The following figure illustrates parallel processing.

Figure 10. 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.

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.

For more information, see Add virtual processors in online mode on page 106.

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 105.

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 89.

How virtual processors service threads

A virtual processor services multiple threads concurrently by switching between them.

At a given time, a virtual processor can run only one thread. 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:

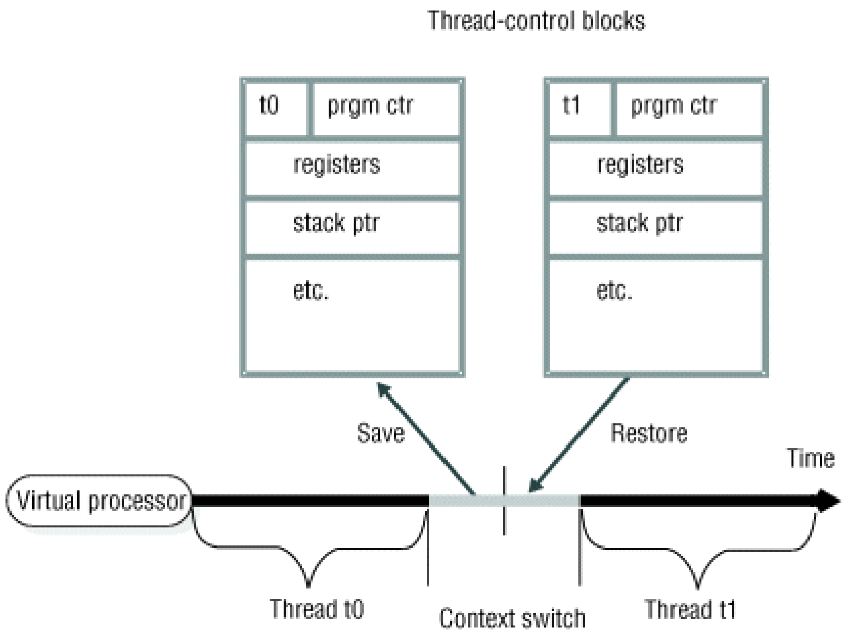
- A predetermined point in the code
- When the thread can no longer execute until some condition is met

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.

Figure 11. Context switch: how a virtual processor switches from one thread to another



Stacks

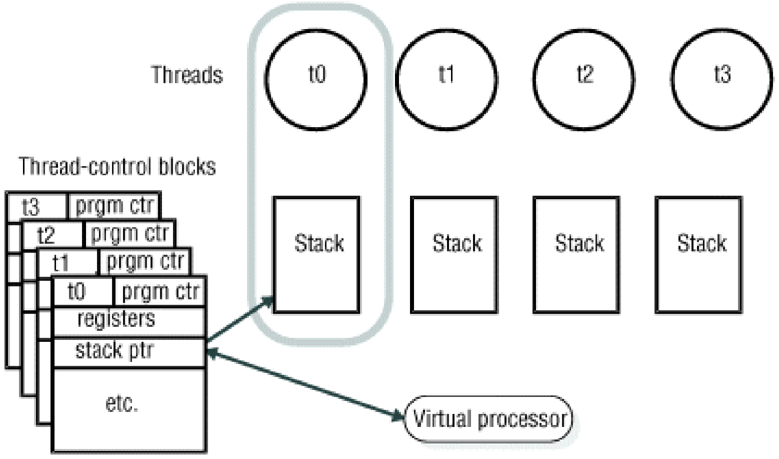
The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*.

For information about how to set the size of the stack, see Stacks on page 127.

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 12. 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 107. 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 106.

Sleep queues

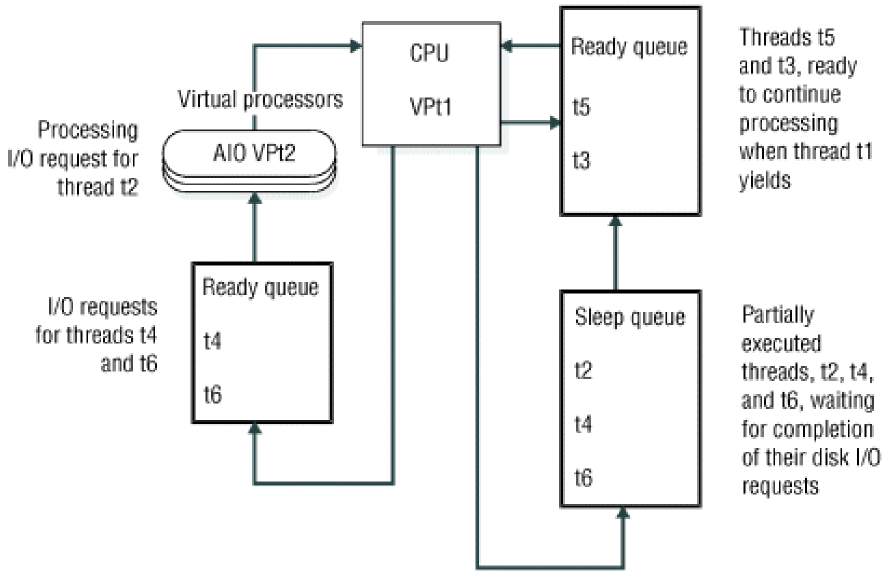
Sleep queues hold the contexts of threads that have no work to do at a particular time. A thread is put to sleep either for a specified period of time or forever.

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 13. How database server threads are queued to perform database I/O


CPU
VPt1
Ready queue
t5
t3
Threads t5 and t3, ready to continue processing when thread t1 yields
Virtual processors
Processing I/O request for thread t2
AIO VPt2
Ready queue
t4
t6
I/O requests for threads t4 and t6
Sleep queue
t2
t4
t6
Partially executed threads, t2, t4, and t6, waiting for completion of their disk I/O requests

Wait queues

Wait queues hold threads that must wait for a particular event before they can continue to run.

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 150.

Virtual processor classes

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 14. Virtual-processor classes

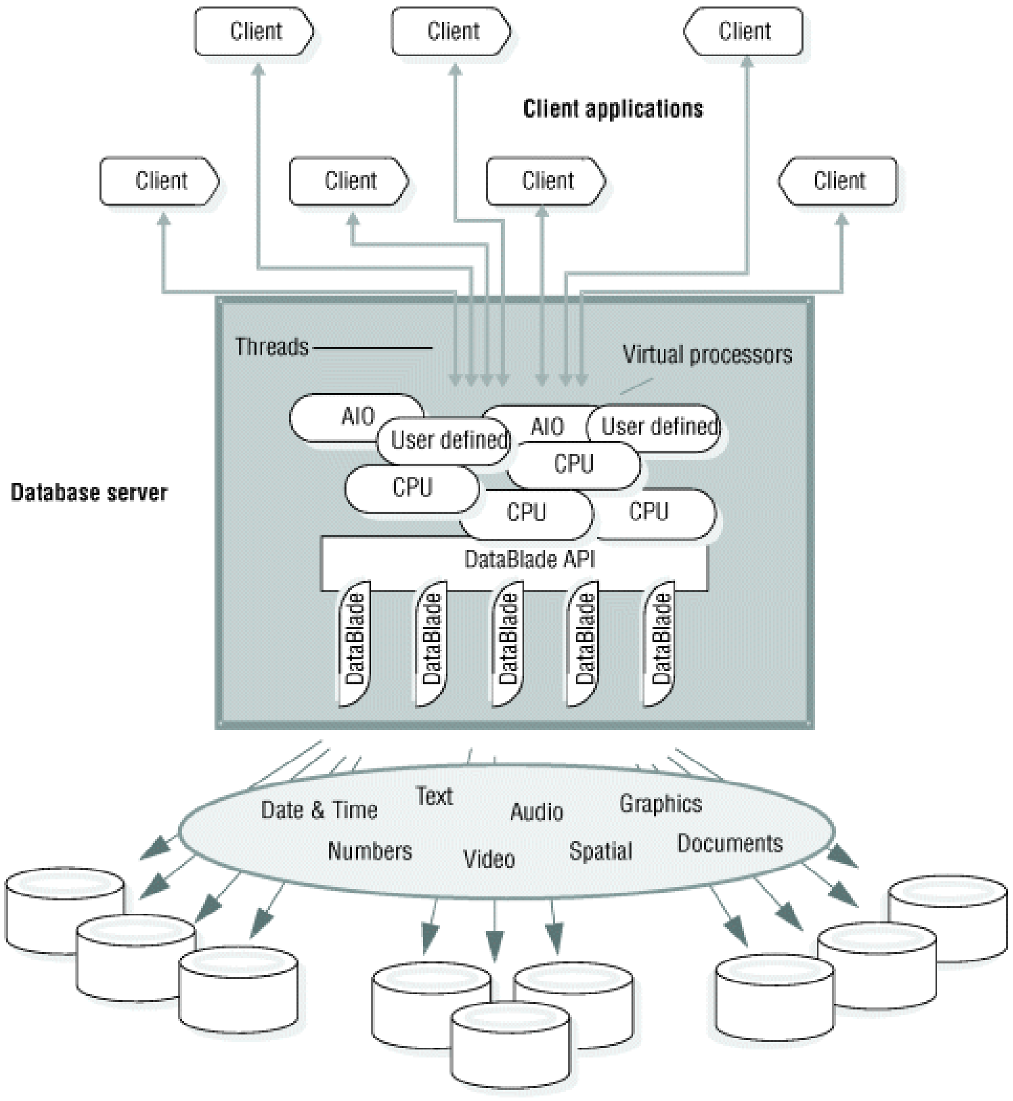
Virtual-processor class	Category	Purpose
ADM	Administrative	Performs administrative functions.
ADT	Auditing	Performs auditing functions.
AIO	Disk I/O	Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
BTS	Basic text searching	Runs basic text search 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.
Encrypt	Encryption	Used by the database server when encryption or decryption functions are called. On Windows™ systems, the number of encrypt virtual processors is always set to 1, regardless of the value that is set in the `onconfig` file.
IDSXMLVP	XML publishing	Runs XML publishing functions.
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.

Table 14. Virtual-processor classes (continued)

Virtual-processor class	Category	Purpose
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.
tenant	Multitenancy	Runs session threads for tenant databases. Tenant virtual processors are a special case of user-defined processors that are specific to tenant databases.
TLI	Network	Uses the Transport Layer Interface (TLI) to perform network communication.
classname	User defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected.

The following figure illustrates the major components and the extensibility of the database server.

Figure 14. Database server



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 *HCL OneDB™ Administrator's Reference* and the *HCL OneDB™ 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.

You can configure the database server to automatically add CPU VPs when needed, up to the number of CPU processors.

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:

```
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.

Use the VPCLASS configuration parameter to specify the following information about CPU virtual processors:

- 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
- Whether the database server automatically adds CPU virtual processors as needed

In addition to considering the number of CPUs in the computer and the number of users who connect to the database server, also consider 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.

Note: Use the VPCLASS configuration parameter instead of the following discontinued configuration parameters: AFF_SPROC, AFFNPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS.

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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ 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 106.

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 add or drop CPU class virtual processors, see Add virtual processors in online mode on page 106 and Drop CPU and user-defined virtual processors on page 106.

Prevent priority aging

Some operating systems lower the priority of long-running processes as they accumulate processing time, a feature of the operating system known as *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 *HCL OneDB™ Administrator's Reference*.

Processor affinity

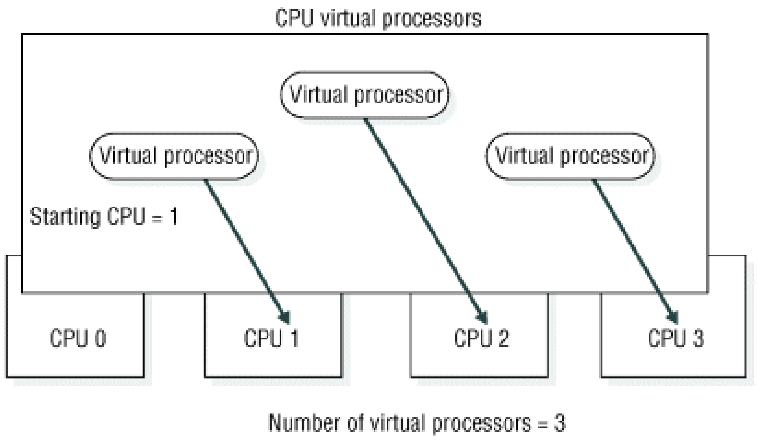
The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*.

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.

Figure 15. 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 *HCL OneDB™ 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

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 *HCL OneDB™ 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 Set virtual-processor configuration parameters on page 105 and the topics about configuration parameters in the *HCL OneDB™ Administrator's Reference*.

Assign a UDR to a user-defined virtual-processor class

The SQL CREATE FUNCTION statement registers a user-defined routine.

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 *HCL OneDB™ Administrator's Reference*.

For more information about the CREATE FUNCTION statement, see the *HCL OneDB™ Guide to SQL: Syntax*.

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 Add virtual processors in online mode on page 106 and Drop CPU and user-defined virtual processors on page 106.

Tenant virtual processor class

Tenant virtual processor classes are specific to tenant databases. If you configure multitenancy for your HCL OneDB™ instance, you can specify that session threads for tenant databases are run in tenant virtual processors instead of CPU virtual processors.

You can create a tenant virtual processor class by defining the class and the number of virtual processors when you create a tenant database. You can assign a tenant virtual processor class to multiple tenant databases. Set the VP_MEMORY_CACHE_KB configuration parameter to create a private memory cache for each CPU virtual processor and tenant virtual processor.

A tenant virtual processor class is automatically dropped when all tenant databases that include the virtual processor class in their definitions are dropped.

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 *HCL® 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 *HCL OneDB™ 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), and 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 94.

I/O priorities

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 15. 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.

I/O is performed to 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.

I/O is performed 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 332.) 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 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 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 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: HCL OneDB™ implements KAIO when HCL OneDB™ 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 HCL OneDB™.

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*, which is 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 pending work 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. You can start additional AIO virtual processors while the database server is in online mode. You cannot drop AIO virtual processors while the database server is in online mode.

You can enable the database server to add AIO virtual processors and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload. Include the autotune=1 keyword in the VPCLASS configuration parameter setting.

You can set the AUTO_AIOVPS configuration parameter to enable 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.

Manually controlling the number of AIO VPs

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 shows 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.

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 that is 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, HCL OneDB™ 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.

Network virtual processors

A client can connect to the database server in the through following ways: a network connection, a pipe, or 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 *HCL OneDB™ Administrator's Reference*.

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.

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 16. A listen thread for each listen port

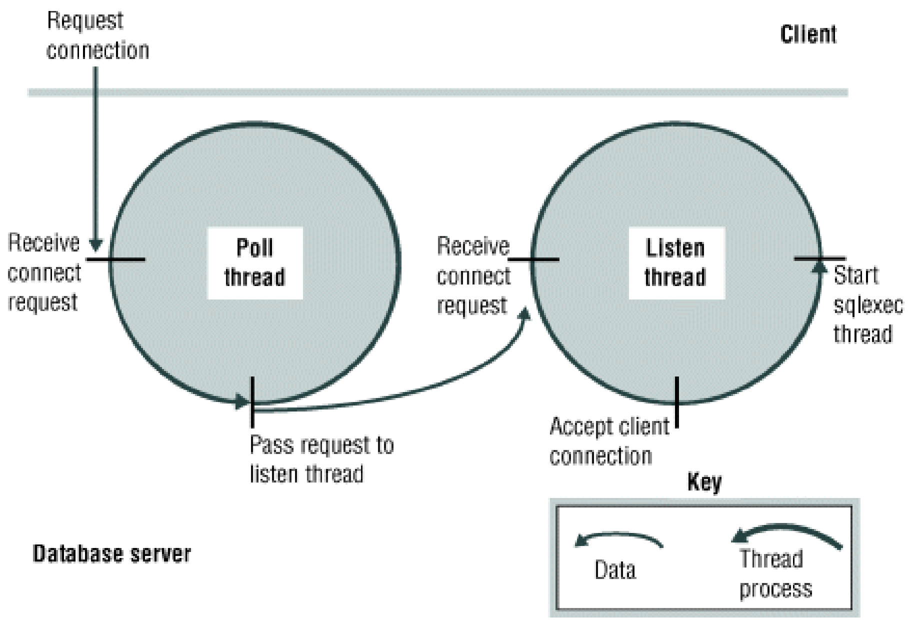
dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1

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 on page 98.

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 97 and to the NETTYPE configuration parameter in the *HCL OneDB™ 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 16. The roles of the poll and the listen threads in connecting to a client

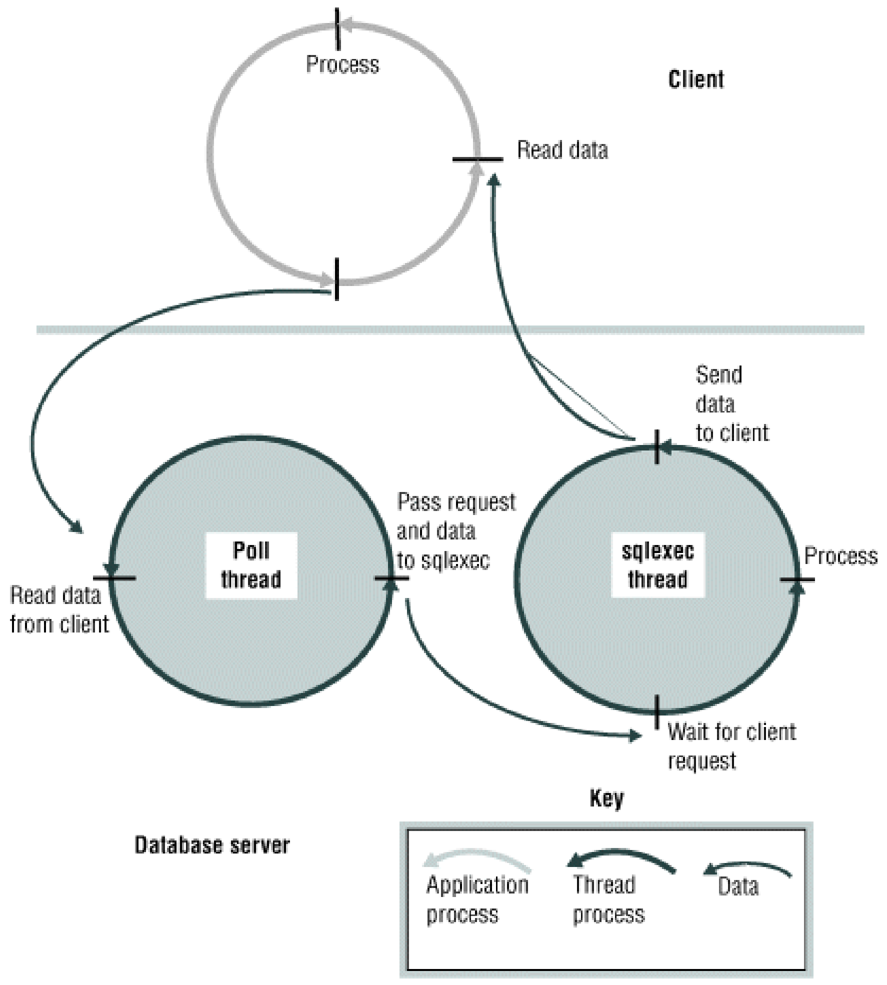


A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlexec** thread. For network connections, the poll thread places the message in a queue in the shared-memory global pool. The poll thread then wakes up the **sqlexec** thread of the client to process 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.

Figure 17. 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.

About this task

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, HCL OneDB™ 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 17. 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 21.

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 101, 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 101). 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 18. 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.

Before you begin

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.

About this task

To dynamically start, stop, or restart listen threads:

1. Run one of the following onmode -P commands:

   Choose from:

   - 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:

   Choose from:

   - 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");
     ```

Example

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");
```

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 *HCL OneDB™ 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
```

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 *HCL OneDB™ Administrator's Reference*. For more information about MQ messaging, see the *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*. For more information about XML publishing, see the *HCL OneDB™ Database Extensions User's Guide*.

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 Virtual processors and threads on page 76.

Set virtual-processor configuration parameters

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.

The table lists the configuration parameters that are used to configure virtual processors.

Table 19. Configuration parameters for configuring virtual processors

Parameter	Description
MULTIPROCESSOR	Set to `1` to support multiple CPU virtual processors, or to `0` for only a single CPU VP
NETTYPE	Specifies parameters for network protocol threads and virtual processors
SINGLE_CPU_VP	Set to `0` to enable user-defined CPU VPs, or to any other setting for only a single CPU VP
VPCLASS	Each defines a VP class and its properties, such as how many VPs of this class start when the server starts
VP_MEMORY_CACHE_KB	Speeds access to memory blocks by creating a private memory cache for each CPU virtual processor

Start and stop virtual processors

When you start the database server, the oninit utility starts the number and types of virtual processors that you specify directly and indirectly.

You configure virtual processors primarily through configuration parameters and, for network virtual processors, through parameters in the sqlhosts information.

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 more virtual processors to improve performance, if necessary.

While the database server is in online mode, you can drop virtual processors of the CPU and user-defined classes.

To shut down the database server and stop all virtual processors, use the onmode -k command.

Add virtual processors in online mode

While the database server is running, you can start additional virtual processors for some virtual processor classes with the -p option of the onmode utility.

You can start these additional virtual processors with the -p option of the onmode utility. You can add network virtual processors with the NETTYPE configuration parameter.

You can start these additional virtual processors with the -p option of the onmode utility or ISA. You can add network virtual processors with the NETTYPE configuration parameter.

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 91.

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.

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 97.

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 90.

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 *HCL OneDB™ Performance Guide*.

For examples of output for the onstat -g commands, see information about the onstat utility in the *HCL OneDB™ Administrator's Reference*.

Monitor virtual processors with command-line utilities

Use onstat -g options to monitor virtual processors.

The following options can be used 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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ 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
user cpu	Minutes of user CPU used
syscpu	Minutes of system CPU used

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 Manage shared memory on page 142.

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.

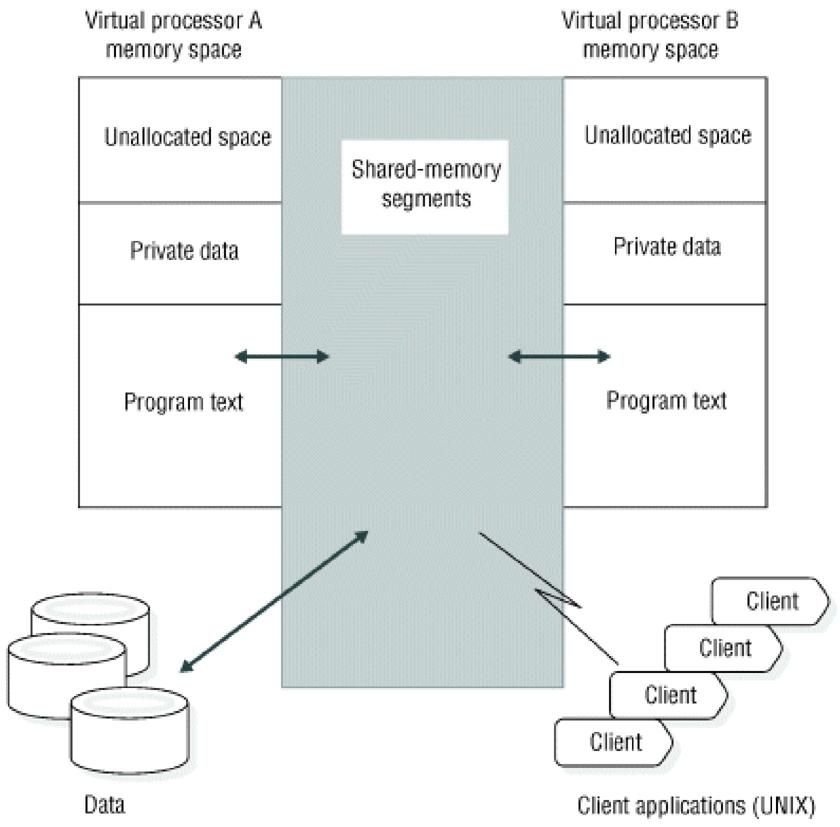
Shared memory provides the fastest method of interprocess communication, because it processes read and write messages at the speed of memory transfers.

Shared-memory use

The database server uses shared memory to enable virtual processors and utilities to share data and to provide a fast communications channel for local client applications that use IPC communication.

The following figure illustrates the shared-memory scheme.

Figure 18. How the database server uses shared memory



Shared-memory allocation

The database server creates portions in shared memory to handle different processes.

The database server creates the following portions of shared memory:

- The *resident* portion
- The *buffer pool* portion
- The *virtual* portion
- The IPC *communications* or message portion

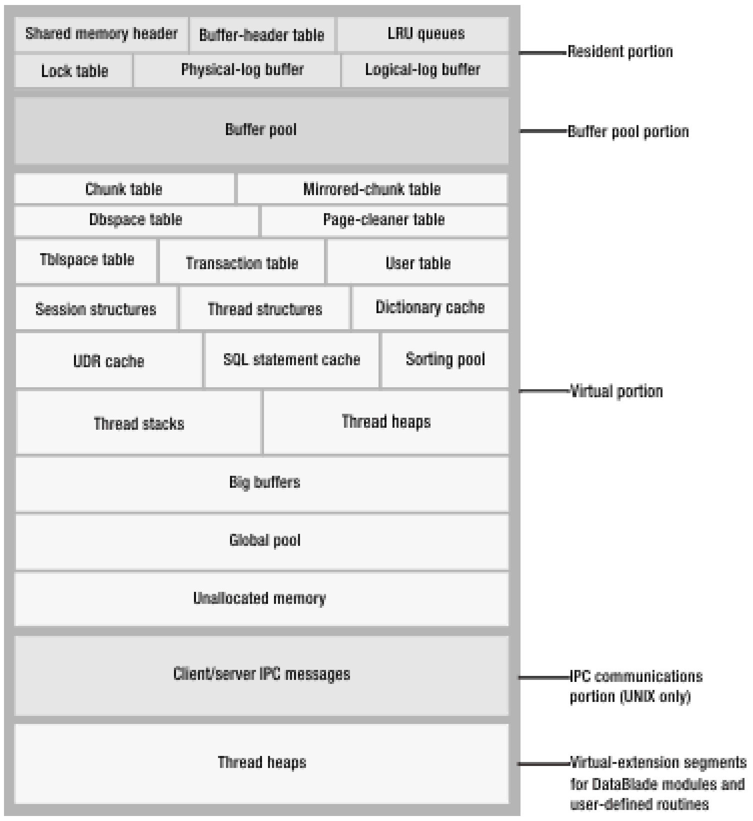
  If the `sqlhosts` file specifies shared-memory communications, the database server allocates memory for the communications portion.
- The *virtual-extension* portion

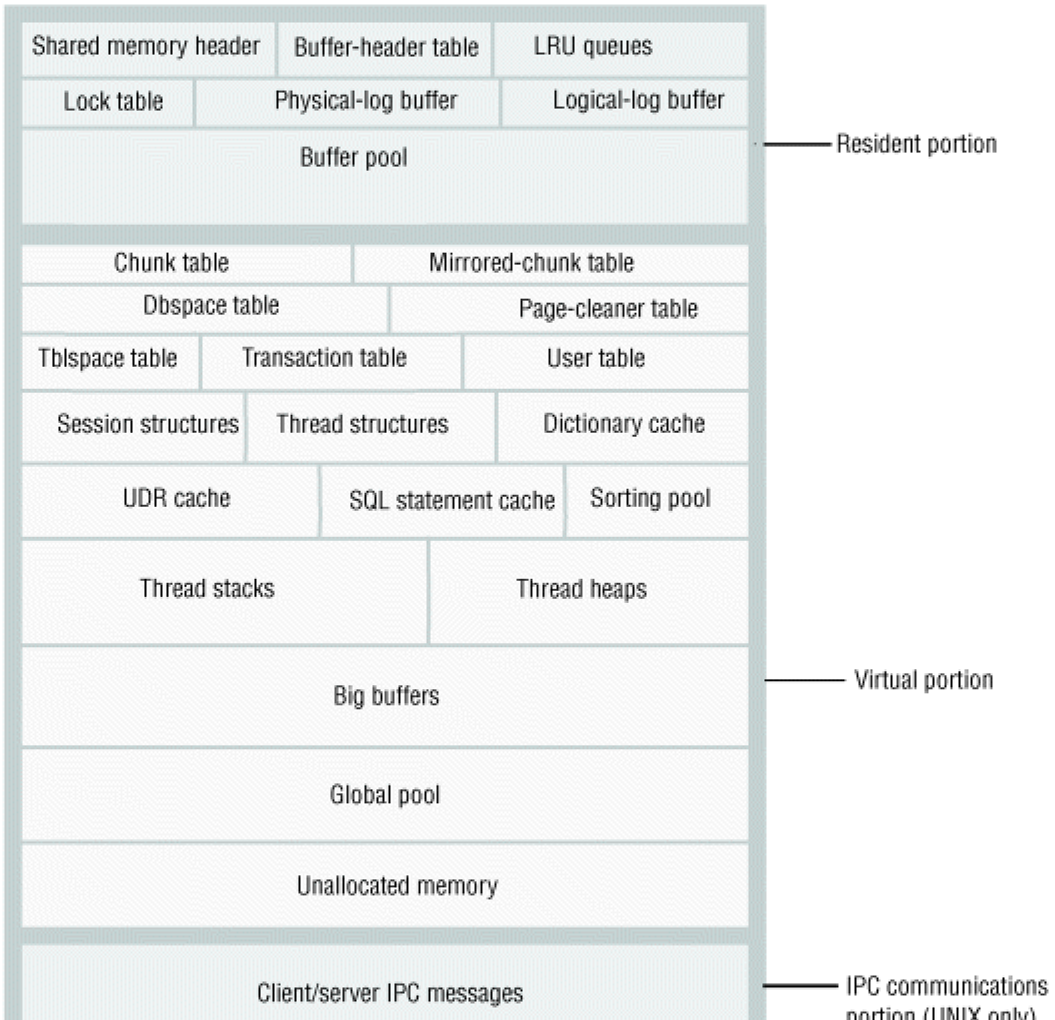
The database server adds operating-system segments, as required, to the virtual and virtual-extension portions of shared memory.

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 19. 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 *HCL OneDB™ 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

A number of processes attach to the database server 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 Client/server communication on page 13, and Network virtual processors on page 97.

If the **ONEDB_ SHMBASE** 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 **ONEDB_ SHMBASE** 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 **ONEDB_ SHMBASE** environment variable, see the *HCL OneDB™ Guide to SQL: Reference*.

How utilities attach to shared memory

Database server utilities such as onstat, onmode attach to shared memory.

The onstat, onmode, and ontape utilities attach to shared memory through one of the following files.

Operating system	File
UNIX™	`$ONEDB_HOME/etc/.infos.`*`servername`*
Windows™	`%ONEDB_HOME%\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 **ONEDB_SERVER** 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 *HCL OneDB™ 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 HCL OneDB™ machine notes.

Attach additional shared-memory segments

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.

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

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.

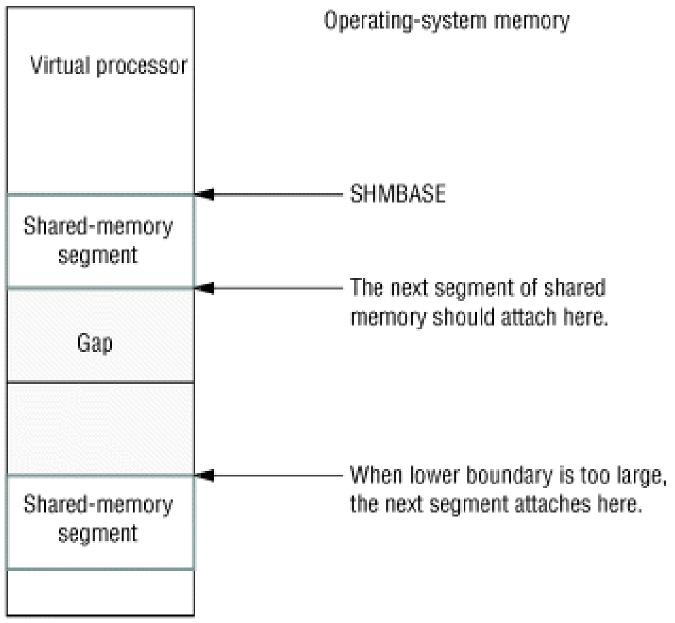
Define the shared-memory lower-boundary address

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.

Figure 20. Shared-memory lower-boundary address overview


Operating-system memory
Virtual processor
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

Resident portion of shared memory

The operating system, as it switches between the processes that run 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 HCL® OneDB® database server can adversely affect the performance of other applications, if any, on the same computer.

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, and 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.

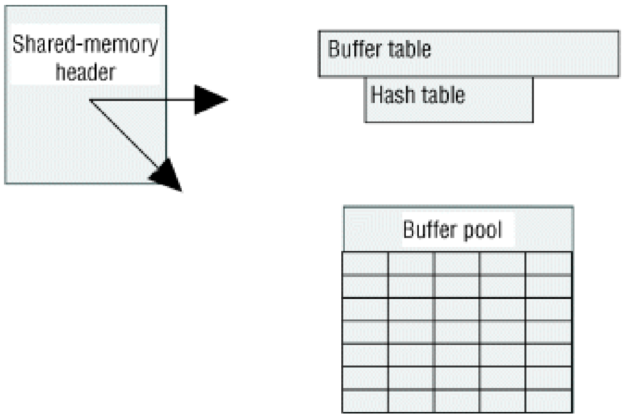
The size of the shared-memory header is about 200 KB, but the size varies depending on the computer platform. You cannot tune the size of the header.

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 21. 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. Each buffer is the size of one database server page. 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.

If you are creating a dbspace with a non-default page size, the dbspace must have a corresponding buffer pool. If a buffer pool for a non-default page size does not exist, the database server automatically creates a large-page buffer. For example, if you create a dbspace with a page size of 6 KB, the database server creates a buffer pool with a size of 6 KB.

If the database server is in online, quiescent, or administration mode, you can also use the onparams -b command to add a new buffer pool of a different size. When you use the onparams -b command, the information you specify is automatically transferred to the `onconfig` file as a new entry of the BUFFERPOOL configuration parameter.

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.

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.

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.

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 by mutexes and lock-access information.

The onstat -b command displays information about the buffers. The LG_ADDBPOOL log record and the **sysbufpool** system catalog table contain information about each buffer pool.

Logical-log buffer

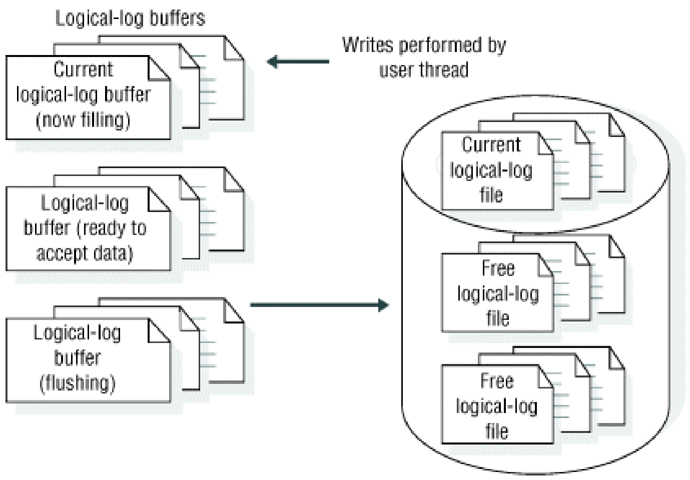
The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server.

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 22. The logical-log buffer and its relation to the logical-log files on disk


Logical-log buffers
Writes performed by user thread
Current logical-log buffer (now filling)
Logical-log buffer (ready to accept data)
Logical-log buffer (flushing)
Current logical-log file
Free logical-log file
Free logical-log file

For a description of how the database server flushes the logical-log buffer, see Flush the logical-log buffer on page 138.

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 *HCL OneDB™ Administrator's Reference*.

For information about the affect of TEXT and BYTE data on shared memory buffers, see Buffer large-object data on page 139.

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 137. For information about monitoring the physical-log file, see Monitor physical and logical-logging activity on page 337.

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 *HCL OneDB™ 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 118. For more information about how the data replication buffer is used, see How data replication works on page 390.

Lock table

A lock is created when a user thread writes an entry in the lock table. A single transaction can own multiple locks. The lock table is the pool of available locks.

For an explanation of locking and the SQL statements associated with locking, see the *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference* and the topics about configuration effects on memory utilization in your *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*.

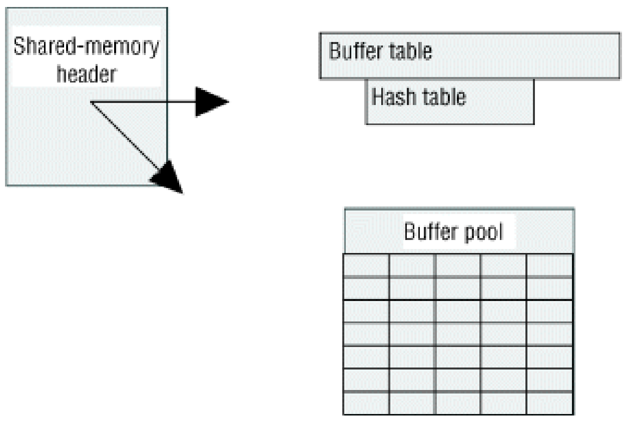
For more information about using and monitoring locks, see the topics about locking in your *HCL OneDB™ Performance Guide* and the *HCL OneDB™ Guide to SQL: Tutorial*.

Buffer pool portion of shared memory

The buffer pool portion of shared memory contains the buffers that store dbspace pages that are read from disk.

The following figure illustrates the shared-memory header and the buffer pool.

Figure 23. 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 or the overall size of the buffer pool. Each buffer is the size of one database server page. Too few buffers can severely affect performance. You can set the BUFFERPOOL configuration parameter to allow the database server to automatically increase the number of buffers as needed to improve performance. Otherwise, you must monitor the database server and tune the number of buffers to determine an acceptable value.

A buffer pool manages one size of pages. You need a different buffer pool for each page size that is used by storage spaces in the database server. The database server automatically creates the required buffer pools. For example, if you create the first dbspace that has a page size of 6 KB, the database server creates a buffer pool to cache the default number of 6 KB pages in memory. You can control the properties of buffer pools with the BUFFERPOOL configuration parameter.

If the database server is in online, quiescent, or administration mode, you can also use the onparams -b command to add a buffer pool of a different size. When you use the onparams -b command, the information that you specify is transferred automatically to the `onconfig` file as a new entry of the BUFFERPOOL configuration parameter.

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.

Automatic LRU (least recently used) tuning affects all buffer pools and adjusts the lru_min_dirty and lru_max_dirty values that can be explicitly set by the BUFFERPOOL configuration parameter.

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 by mutexes and lock-access information.

The onstat -b command shows information about the buffers.

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.

Components of the virtual portion of shared memory

The virtual portion of shared memory stores a variety of data.

The following data is stored in the virtual portion of shared memory:

- 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 159.

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 116. For information about how to monitor the buffers, see Monitor buffers on page 151.

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 240.

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 or removable. 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.

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 240.

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 *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*.

Tblspace table

he tblspace table tracks all active tblspaces in a database server instance.

TAn 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 245.

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 *HCL OneDB™ 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 *HCL OneDB™ Guide to SQL: Tutorial*, the *HCL OneDB™ Guide to SQL: Reference*, and the *HCL OneDB™ 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 *HCL OneDB™ 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. The created data structure is 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.

You can impose restrictions on the memory allocated for sessions by setting the SESSION_LIMIT_MEMORY configuration parameter to specify the maximum amount of memory that a session can allocate. The limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

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 80.

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 81. For information about how to monitor the size of the stack for a session, see monitoring sessions and threads section in your *HCL OneDB™ Performance Guide*.

The size of the stack space for user threads is specified by the STACKSIZE parameter in the `onconfig` file. You can change the size of the stack for all user threads, if necessary, by changing the value of STACKSIZE.

You can use the **ONEDB_ STACKSIZE** environment variable to override the **STACKSIZE** value in the server configuration file. Set **ONEDB_ STACKSIZE** in the environment and recycle the instance.

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 *HCL OneDB™ 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 *HCL OneDB™ 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 optimized SQL statements that a user runs. When users run a statement that is stored in the SQL statement cache, the database server does not optimize the statement again, so performance improves.

For more information, see Set SQL statement cache parameters on page 147. For details on how these parameters affect the performance of the SQL statement cache, see the *HCL OneDB™ Performance Guide*.

Sort memory

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.

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

For information about parallel sorts, see your *HCL OneDB™ 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 *HCL OneDB™ Performance Guide*.

Global pool

The global pool stores structures that are global to the database server.

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 *HCL OneDB™ 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 113.

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 150.

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.

The two types of locks are:

- 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. HCL OneDB™ 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. If the AUTO_LRU_TUNING configuration parameter is enabled, the values of the **lru_max_dirty** and **lru_min_dirty** fields are reset automatically as needed to improve performance.

Components of LRU queue

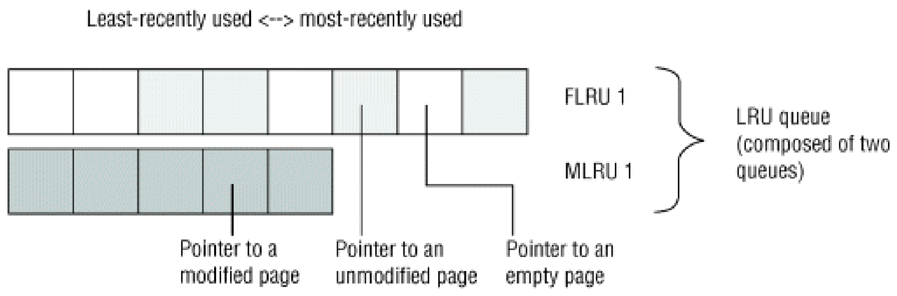
Each LRU queue is composed of a pair of linked lists.

The linked list pairs are 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 24. LRU queue


Least-recently used <--> most-recently used
FLRU 1
MLRU 1
LRU queue (composed of two queues)
Pointer to a modified page
Pointer to an unmodified page
Pointer to an empty page

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 buffers on page 151.

Number of LRU queues to configure

Multiple LRU queues reduce user-thread contention and allow multiple cleaners to flush pages from the queues so that the percentage of dirty pages is maintained 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))
```

You specify the number of LRU queues by setting the lrus value in the BUFFERPOOL configuration parameter. The default number of LRU queues depends on the number of CPUs on your computer:

- If you have a uniprocessor computer, the default value of the lrus field is 8.
- If you have a multiprocessor computer and the MULTIPROCESSOR configuration parameter is enabled, the default value of the lrus field is the greater of 8 or the number of CPU virtual processors.

After you provide an initial value for the lrus field in the BUFFERPOOL configuration parameter, monitor your LRU queues with the onstat -R command. If you find that the percentage of dirty LRU queues consistently exceeds the value of the lru_max_dirty field in the BUFFERPOOL configuration parameter, increase the value of the lrus field to add more LRU queues.

For example, if the value of the lru_max_dirty field is `70` and your LRU queues are consistently 75 percent dirty, you can increase the value of the lrus field. If you increase the number of LRU queues, you shorten the length of the queues, which reduces the work of the page cleaners. However, you must allocate enough page cleaners with the CLEANERS configuration parameter.

Number of cleaners to allocate

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.

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 136.

Number of pages added to the MLRU queues

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**, which 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 136.

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 136.

Read-ahead operations

The database 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.

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 130.)

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 on page 137.
- Logical-log buffer

  See Flush the logical-log buffer on page 138.

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 *HCL OneDB™ 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 triggered by specific conditions.

Flushing of the buffers is initiated when:

- 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 332.)

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 always flushes the contents of the physical-log buffer to disk before any data buffers.

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 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 Physical logging, checkpoints, and fast recovery on page 328.

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.

Types of writes during flushing

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.

The types of writes are as follows:

- Foreground write
- LRU write
- Chunk write

To see the write counts that the database server maintains, run the onstat -F command.

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.

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 131.

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

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.

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

A number of events can cause the logical-log buffer to flush.

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 Logical log on page 300.

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

A number of log records cause flushing of the logical-log buffers in a database that uses unbuffered logging.

The log records that cause flushing are:

- COMMIT
- PREPARE
- XPREPARE
- ENDTRANS

For a comparison of buffered versus unbuffered logging, see the SET LOG statement in the *HCL OneDB™ 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

Checkpoints occur after specific events.

For a detailed description of the events that occur during a checkpoint, see Checkpoints on page 332.

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 136.

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 *HCL OneDB™ Administrator's Reference*.

Blobpages and shared memory

Blobspace blobpages store large amounts of data.

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 305.

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 *HCL OneDB™ 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.

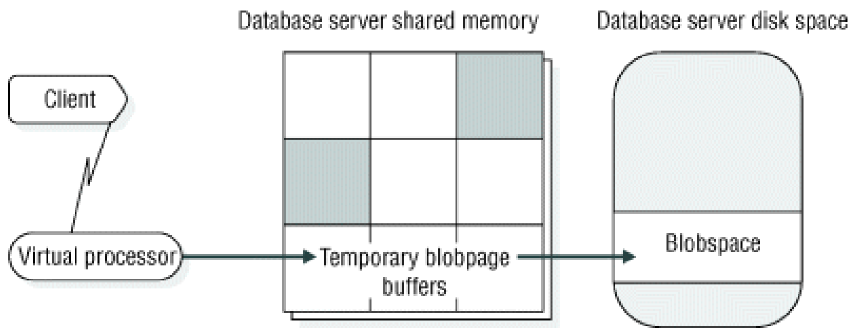
Simple large object data is transferred from the client-application process to the database server in 1 KB segments. The database server begins filling the blobspace buffers with the 1 KB pieces and attempts to buffer two blobpages at a time.

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 25. Writing simple large object to a blobspace


Database server shared memory
Database server disk space
Client
Virtual processor
Temporary blobpage buffers
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 166.

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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*. For information about creating an sbspace, see information about the onspaces utility in the *HCL OneDB™ 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
- 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 *HCL OneDB™ Administrator's Reference* and the chapter on shared memory in the *HCL OneDB™ Performance Guide*.

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 *HCL OneDB™ 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™.

For Windows™ version 2003 and earlier, you must enable this feature with an entry in the Windows™ boot file. Enabling larger shared-memory segments is referred to by Microsoft™ as 4-gigabyte tuning (4GT).

To add the entry, edit the `boot.ini` file (in the top level, or root directory). You can either add a 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™)

The operating system identifies each shared-memory segment with a shared-memory identifier. Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory.

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 114.

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, buffer pool, or virtual portion of shared memory.You can modify the configuration parameters that affect the resident or virtual portion of shared memory.

Set parameter for buffer pool shared memory

The BUFFERPOOL configuration parameter in the `onconfig` file specifies information about a buffer pool. Each page size that is used by the database server requires a buffer pool, which is represented in the `onconfig` file by a BUFFERPOOL configuration parameter entry.

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.

BUFFERPOOL

Specifies information about the buffer pool. The BUFFERPOOL configuration parameter must be defined for each page size that is used by dbspaces.

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

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.

Set parameters for shared-memory performance

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 buffer pool.

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.

Set SQL statement cache parameters

There are different ways that you can configure the SQL statement cache.

The following table shows ways to configure the SQL statement cache.

Table 20. 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
- onstat -g ssc all
- onstat -g ssc pool

For more information about these configuration parameters, onstat -g options, and onmode commands, see the *HCL OneDB™ 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 *HCL OneDB™ Performance Guide*. For details on qualifying and identical statements, see the *HCL OneDB™ 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 Changing database server operating modes on page 72.

Turn residency on or off for resident shared memory

You can turn residency on or off for the resident portion of shared memory.

Residency can be turned on or off 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 116.

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 **HCL OneDB™ 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.

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.

Configure the server response when memory is critically low

You can configure the actions that primary or standard database server takes 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 to process. Low-memory responses are useful for embedded applications that have memory limitations.

To set up automatic low-memory management on a primary or standard server:

- 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.
- Verify that the SHMTOTAL configuration parameter is set to a positive integer value.

To disable automatic low-memory management, run an SQL administration API command with the **scheduler lmm disable** argument.

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.

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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*.

Command-line utilities to monitor shared memory and latches

Use 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 *HCL OneDB™ 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 that are cached is an important measure of performance. 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 whether another thread is waiting for the buffer.

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. 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 were modified.

You can obtain information about buffer pool activity from the onstat utility, the **sysprofile** SMI table.

onstat commands to monitor buffers

You can use the following onstat commands to monitor buffers:

onstat -g buf

Run the onstat -g buf command to obtain statistics about how active and efficient each buffer is. The following types of statistics are shown:

- Page reads and writes
- Caching percentages
- Waits for buffers
- Flushes
- Extensions of the buffer pool
- Buffer pool segments
- Fast cache

onstat -B

Run the onstat -B command 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

onstat -b

Run the onstat -b command to obtain the following information about each buffer:

- Address of each buffer that is currently held by a thread
- Page numbers for the page that is held in the buffer
- Type of page that is held in the buffer (for example, data page, tblspace page, and so on)
- Type of lock that is placed on the buffer (exclusive or shared)
- Address of the thread that is 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 shown in the onstat -u output to obtain the session ID number.

onstat -X

Run the onstat -X command 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.

onstat -R

Run the onstat -R command to show information about buffer pools, the number of buffers in each LRU queue, and the number and percentage of the buffers that are modified or free.

onstat -F

Run the onstat-F command to obtain a count by write type of the writes that are performed and 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

The sysprofile SMI table

Query the **sysprofile** table to obtain statistics on cached reads and writes, write types, and total buffer waits. The following rows are relevant.

bufreads

Number of reads from buffers

bufwrites

Number of writes to buffers

buffwts

Number of times that any thread was required to wait for a buffer

chunkwrites

Number of chunk writes

dskreads

Number of reads from disk

dskwrites

Number of writes to disk

fgwrites

Number of foreground writes

lruwrites

Number of LRU writes

Monitor buffers with ON-Monitor (UNIX™)

You can use ON-Monitor to obtain statistics about cached reads and writes.

To access the onstat -p fields (**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
...
```

Deleting shared memory segments after a server failure

You must close shared memory segments after a database server failure.

About this task

Important: This procedure must be performed by a DBA with experience using HCL OneDB™. Consult technical support for assistance. This procedure is for UNIX™ systems only.

In the event of a failure of a 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.
      Example
      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 HCL OneDB™ instance

   b. Add the server number to 21078.
      Example
      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.
      Example
      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 4.d on page 155.

   f. Remove each shared memory ID associated with the number from 4.d on page 155.

What to do next

For more information about the onclean utility, see the *HCL OneDB™ Administrator's Reference*.

Consult your operating system documentation for the correct **ipcm** syntax for your system.

Data storage

The database server uses physical units of storage to allocate disk space. It stores data in logical units. 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 following topics define terms and explain concepts that you must understand to manage disk space. 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

The database server uses the following physical units to manage disk space:

Chunks on page 156

Pages on page 159

Blobpages on page 160

Sbpages on page 161

Extents on page 162

The database server stores data in the following logical units:

Dbspaces on page 163

Temporary dbspaces on page 165

Blobspaces on page 166

Sbspaces on page 166

Temporary sbspaces on page 174

Plogspace on page 176

Extspaces on page 176

Databases on page 177

Tables on page 177

Tblspaces on page 185

Partitions and offsets on page 159

The database server maintains the following storage structures to ensure physical and logical consistency of data:

Logical log

Physical log

Reserved pages

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.

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.

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.

A 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 201.

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 201.

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 199.

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 199.

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 HCL OneDB™ 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 *HCL OneDB™ 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 OneDB 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 OneDB 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.

Partitions and 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.

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 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.

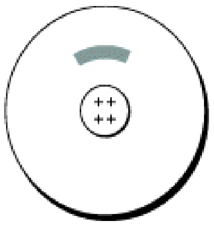
You might also be required to specify an offset to prevent the database server from overwriting partition information.

Pages

A *page* is the physical unit of disk storage that the database server uses to read from and write to HCL® OneDB® databases.

The following figure illustrates the concept of a page, represented by a darkened sector of a disk platter.

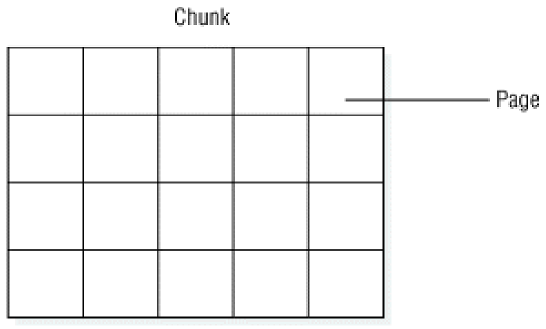
Figure 26. A page on disk



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.

Figure 27. A chunk, logically separated into a series of pages



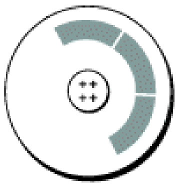
For information about how the database server structures data within a page, see the chapter on disk structures and storage in the *HCL OneDB™ Administrator's Reference*

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.

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.

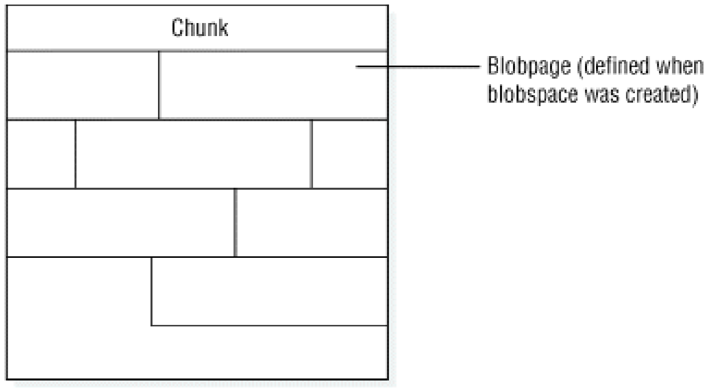
Figure 28. A blobpage on disk



For information about how HCL OneDB™ structures data stored in a blobpage, see structure of a blobspace blobpage in the disk structures and storage topics of the *HCL OneDB™ 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 29. 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.

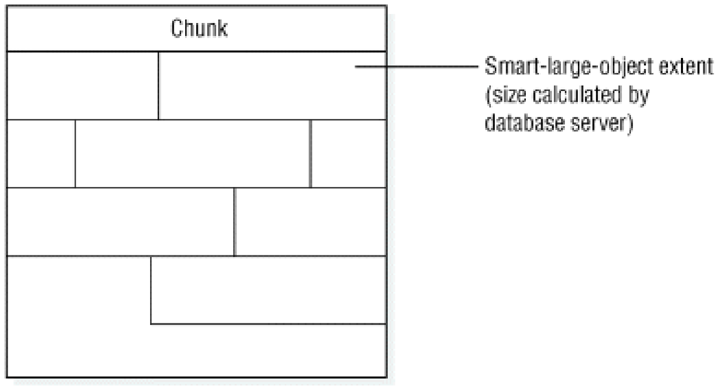
For information about how to determine the size of a blobpage, see Determine blobpage size on page 216. For a description of blobspaces, see Blobspaces on page 166.

Sbpages

An *sbpage* is the type of page that the database server uses to store smart large objects within an sbspace. 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 30. A chunk in an sbspace, logically separated into a series of extents


Chunk
Smart-large-object extent (size calculated by database server)

Smart large objects cannot be stored in a dbspace or blobspace. For more information, see Sbspaces on page 166, and sbspace structure in the disk structures and storage chapter of the *HCL OneDB™ 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 169.

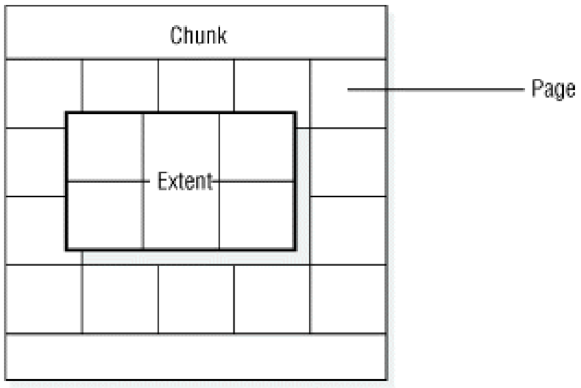
Extents

An extent consists of a collection of contiguous pages that store data for a given table.

When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. (See Tables on page 177.) 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 31. An extent that consists of six contiguous pages on a raw disk device



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.

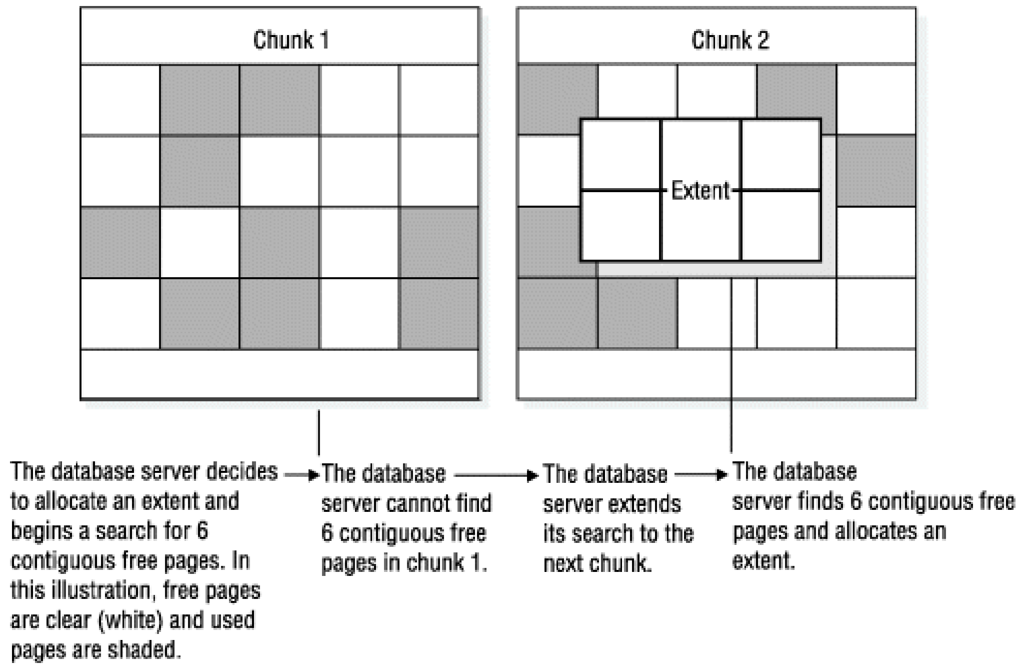
When you create a table, you can specify the size of the initial extent, and the size of the extents to be added as the table grows. You can also modify the size of an extent in a table in a dbspace, and you can modify the size of new subsequent extents. To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the *HCL OneDB™ Guide to SQL: Syntax* and disk structures in the *HCL OneDB™ 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 169.

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.

Figure 32. Process of extent allocation


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.

Dbspaces

A *dbspace* is a logical unit that can contain between 1 and 32,766 chunks. The database server uses the dbspace to store databases and tables. Place databases, tables, logical-log files, and the physical log in dbspaces.

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.

When you create a standard dbspace, you can specify the first and next extent sizes for the **tblspace** in the dbspace. Specifying the extent sizes reduces the number of **tblspace** extents and reduces the frequency of situations when you must place the **tblspace** extents in non-primary chunks.

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.

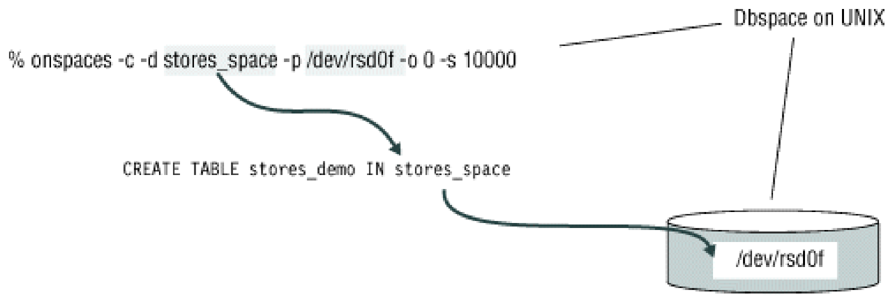
Control of where simple large object data is stored

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 that holds 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.

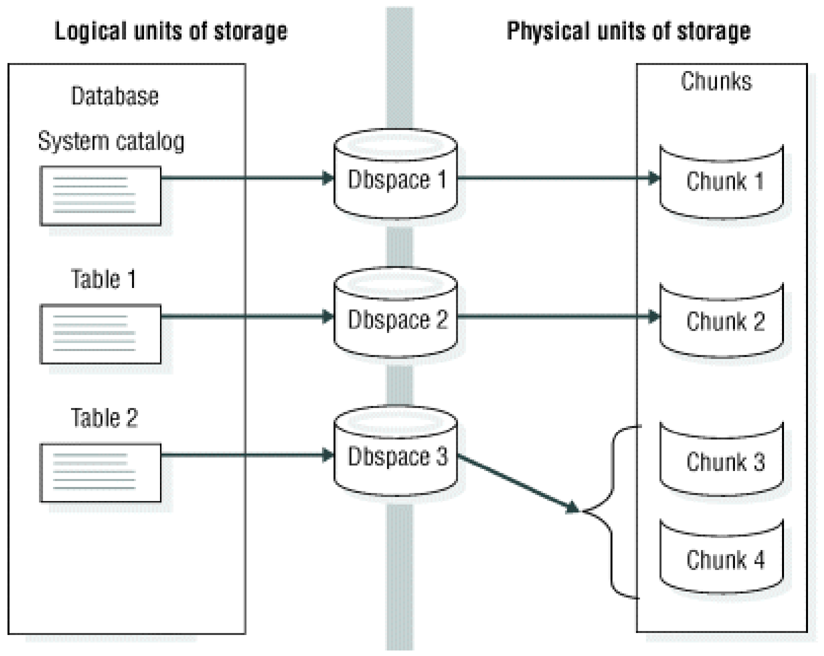
Figure 33. Control table placement with the CREATE TABLE... IN statement


Dbspace on UNIX
% onspaces -c -d stores_space -p /dev/rsd0f -o 0 -s 10000
CREATE TABLE stores_demo IN stores_space
/dev/rsd0f

Before you create a database or table in a dbspace, you must first create the dbspace.

A dbspace includes one or more chunks, as the following figure shows. You can add more chunks at any time. A database server administrator must to monitor dbspace chunks for fullness and to anticipate the necessity to allocate more chunks to a dbspace. When a dbspace contains more than one chunk, you cannot specify the chunk in which the data is located.

Figure 34. Dbspaces that link logical and physical units of storage



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 177 and the disk structures and storage chapter in the *HCL OneDB™ 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 188 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. It behaves differently from a standard dbspace in many ways.

A temporary dbspace is temporary only in the sense that the database server does not preserve any of its contents when the database server restarts. The database server never drops a temporary dbspace unless it is explicitly directed to do so.

Temporary dbspaces cannot be mirrored by the database server.

Whenever you start the database server, all chunks in temporary dbspaces are recreated from scratch. These chunks can therefore be located on RAM drives if desired.

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.

For a temporary table in a standard dbspace, at minimum the server logs table creation, the allocation of extents, and the dropping of the table. In contrast, the database server does not log any operations on 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.

Temporary dbspaces are never archived by the database server, reducing the size of your storage-space backup.

In addition to temporary tables, the database server uses temporary dbspaces to store the before images of data that is 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 root dbspace is used. If the root dbspace fills, the backup may fail.

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 210.

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 177) 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 246. For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see Manage disk space on page 197.

For information about the structure of a blobspace, see the topics about disk structures and storage in the *HCL OneDB™ Administrator's Reference*.

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 *HCL OneDB™ Guide to SQL: Reference*.

Advantages of using sbspaces

Sbspaces have advantages over blobspaces.

The following are advantages of using sbspaces:

- 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. HCL OneDB™ provides this smart-large-object interface in the DataBlade® API and the 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 *HCL OneDB™ 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 programming interface. For more information about the DataBlade® API functions, see the *HCL OneDB™ DataBlade® API Function Reference*. For more information about the functions, see the *HCL OneDB™ 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 *HCL OneDB™ Enterprise Replication Guide*.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the appropriate configuration parameters when defining a replication server. If the CDR_QDATA_SBSPACE or the CDR_DBSPACE configuration parameter is not set or is set to blank, the cdr define server command automatically creates the necessary disk space and sets the configuration parameters to appropriate values.

Metadata, user data, and reserved area

As with blobspaces and dbspaces, when you create an sbspace, you assign to it one or more chunks. The first chunk of an sbspace always has three areas.

The following are the three areas of the first chunk of an sbspace:

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 219 and the *HCL OneDB™ 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 Manage disk space on page 197.

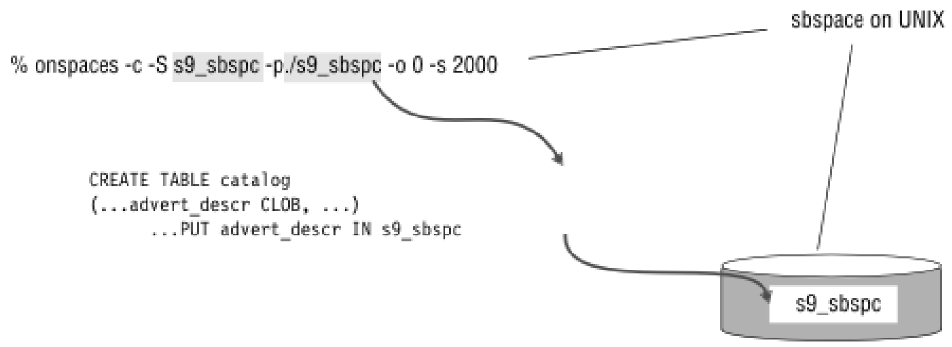
Important: Sbspace metadata is always logged, regardless of the logging setting of the database.

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.

Figure 35. Control smart-large-object placement



```
% onspaces -c -S s9_sbspc -p./s9_sbspc -o 0 -s 2000
```

```
CREATE TABLE catalog
(...advert_descr CLOB, ...)
      ...PUT advert_descr IN s9_sbspc
```

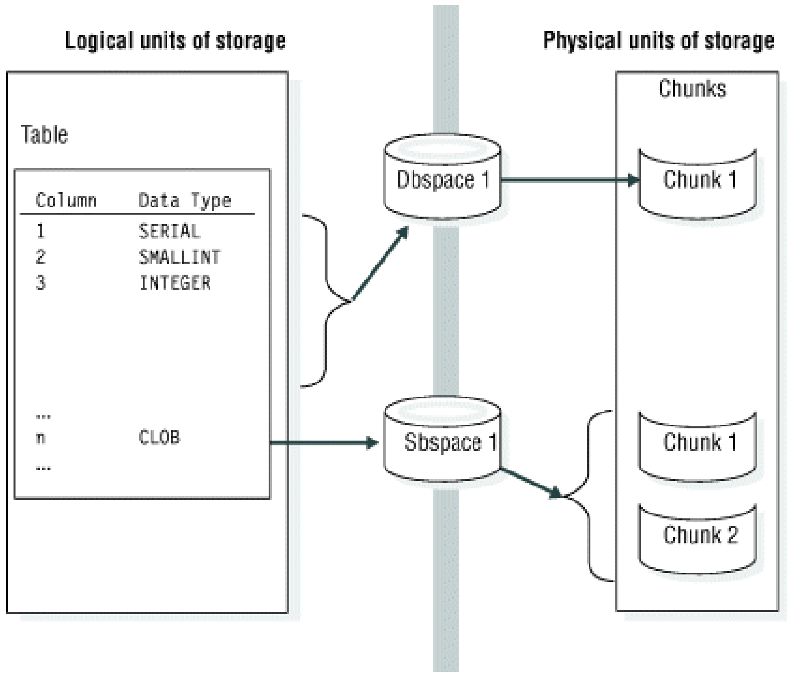
Before you specify an sbspace in a PUT clause, you must first create the sbspace. For more information about how to create an sbspace with the onspaces -c -S command, see Adding a chunk to a dbspace or blobspace on page 211. For more information about how to specify smart large object characteristics in the PUT clause, see the CREATE TABLE statement in the *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ Performance Guide*.

Figure 36. 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 342.

For information about using onspaces to perform the following tasks, see Manage disk space on page 197.

- 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. 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

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 *HCL OneDB™ DataBlade® API Function Reference*.
- The ifx_lo_specset_estbytes function

  For more information about the functions for smart large objects, see the *HCL OneDB™ 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 *HCL OneDB™ 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 the 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 219.

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 *HCL OneDB™ 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.

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 *HCL OneDB™ 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 306.

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 *HCL OneDB™ Guide to SQL: Syntax*.

The programmer can override this logging status with functions that the DataBlade® API and provide. For more information about the DataBlade® API functions for smart large objects, see the *HCL OneDB™ DataBlade® API Function Reference*. For more information about the functions for smart large objects, see the *HCL OneDB™ ESQL/C Programmer's Manual*.

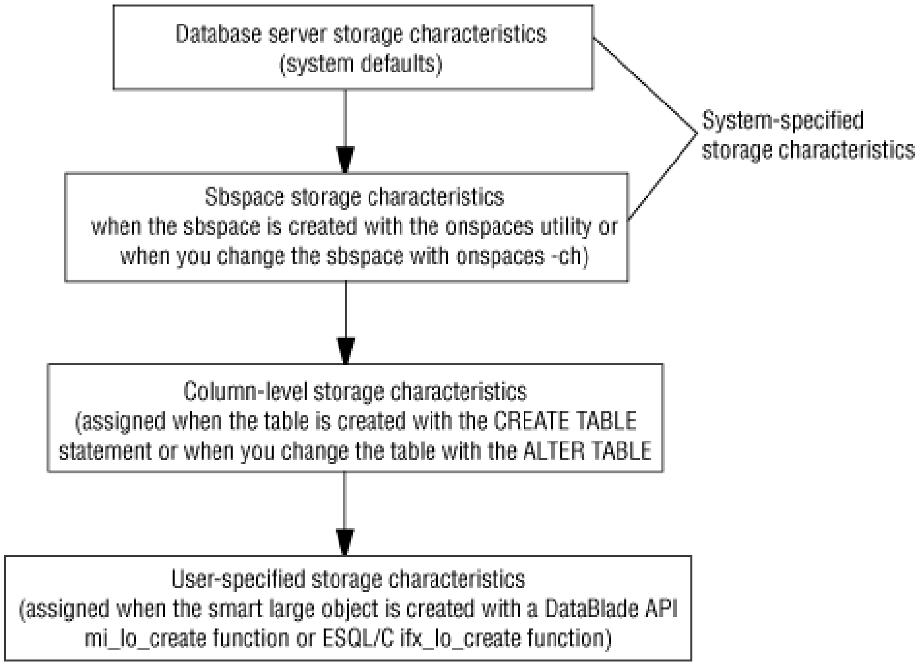
When you turn on logging for an sbspace, the smart large objects pass through the resident portion of shared memory. Although applications can retrieve pieces of a smart large object, you still must consider the larger size of data that might pass through the buffer pool and logical-log buffers. For more information, see Access smart large objects on page 141.

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 37. Storage-characteristics hierarchy


Database server storage characteristics (system defaults)
System-specified storage characteristics
Sbspace storage characteristics when the sbspace is created with the onspaces utility or when you change the sbspace with onspaces -ch)
Column-level storage characteristics (assigned when the table is created with the CREATE TABLE statement or when you change the table with the ALTER TABLE
User-specified storage characteristics (assigned when the smart large object is created with a DataBlade API mi_lo_create function or ESQL/C ifx_lo_create function)

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 *HCL OneDB™ 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 *HCL OneDB™ 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 programming interface provide.

More information about sbspaces

There are various tasks related to using and managing sbspaces.

The following table lists sources of information tasks related to using and managing sbspaces.

Table 21. Finding information for sbspace tasks

Task	Reference
Setting memory configuration parameters for smart large objects	Manage shared memory on page 142
Understanding sbpages	Sbpages on page 161
Specifying I/O characteristics for an sbspace	onspaces option in Storage characteristics of sbspaces on page 169
Allocating space for an sbspace	Creating an sbspace on page 218
Adding a chunk to an sbspace	Adding a chunk to an sbspace on page 219
Defining or altering storage characteristics for a smart large object	Alter storage characteristics of smart large objects on page 220 PUT clause of CREATE TABLE or ALTER TABLE statement in *HCL OneDB™ Guide to SQL: Syntax*
Monitoring sbspaces	Monitor sbspaces on page 249 Topics about table performance considerations in *HCL OneDB™ Performance Guide*
Setting up logging for an sbspace	Log sbspaces and smart large objects on page 306
Backing up an sbspace	Back up sbspaces on page 313
Checking consistency of an sbspace	Validate metadata on page 357
Understanding an sbspace structure	Topics about disk structures in the *HCL OneDB™ Administrator's Reference*
Using onspaces for sbspaces	Topics about utilities in the *HCL OneDB™ Administrator's Reference*
Creating a table with CLOB or BLOB data types	*HCL OneDB™ Guide to SQL: Syntax*
Accessing smart large objects in an application	*HCL OneDB™ DataBlade® API Programmer's GuideHCL OneDB™ ESQL/C Programmer's Manual*
Calculating the metadata area size Improving metadata I/O Changing storage characteristics	Topics about table performance in *HCL OneDB™ Performance Guide*
Understanding smart-large-object locking	Topics about locking in *HCL OneDB™ Performance Guide*

Table 21. Finding information for sbspace tasks (continued)

Task	Reference
Configuring sbspaces for temporary smart large objects	Topics about configuration effects on I/O activity in *HCL OneDB™ 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 221.

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 22. 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

Table 22. Temporary and standard sbspaces (continued)

Characteristics	Standard sbspace	Temporary sbspace
		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 *HCL OneDB™ 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 23. 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 23. Temporary and standard smart large objects (continued)

Characteristics	Smart large object	Temporary smart large object
Table type stored in	Permanent or temporary table	Temporary tables

Plogspace

A plogspace is a logical storage unit that is composed of one chunk that stores the physical log. When the physical log is in the plogspace, the database server increases the size of the physical log as needed to improve performance.

If you did not create a server during installation, the physical log is created in the root dbspace. However, you can create the plogspace to move the physical log to a different dbspace to prevent the physical log from filling the root dbspace. For optimal performance, create the plogspace on a different disk from the root dbspace or the location of the logical logs. If you created a server during installation, the plogspace is created automatically with a default size that depends on the value of the AUTO_TUNE_SERVER_SIZE configuration parameter.

By default, the chunk that you assign to the plogspace is extendable, therefore, the initial size of the chunk can be small. The database server automatically expands the chunk when the physical log requires more space.

The plogspace has the following restrictions:

- A database server instance can have only one plogspace.
- The plogspace can contain only the physical log.
- The plogspace can have only one chunk.
- The chunk must have the same page size as the root dbspace.

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 *HCL OneDB™ Virtual-Table Interface Programmer's Guide*. For more information about creating functions and primary access methods, see the *HCL OneDB™ Guide to SQL: Syntax*.

Databases

A database is a logical storage unit that contains tables and indexes. 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 stored in the dbspace that is specified by the IN clause of the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database is stored in the root dbspace, unless automatic location is enabled. You can enable automatic location by setting the AUTOLOCATE configuration parameter or session environment variable to a positive integer. The database server chooses the dbspaces in which to create new databases and new tables that are created without specified storage locations. Tables are automatically fragmented by round robin in the dbspaces that are chosen by the server.

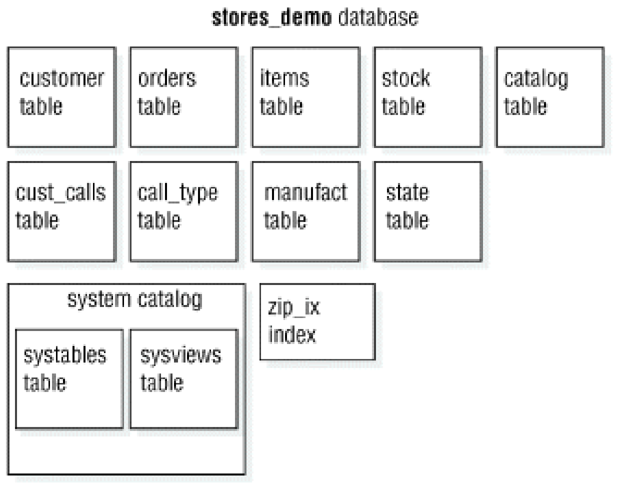
A database is stored in the dbspace that is specified by the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database is stored in the root dbspace.

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 that are contained in the **stores_demo** database.

Figure 38. 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 that is assigned to a specific physical device, the database size is limited to the size of that chunk.

Tables

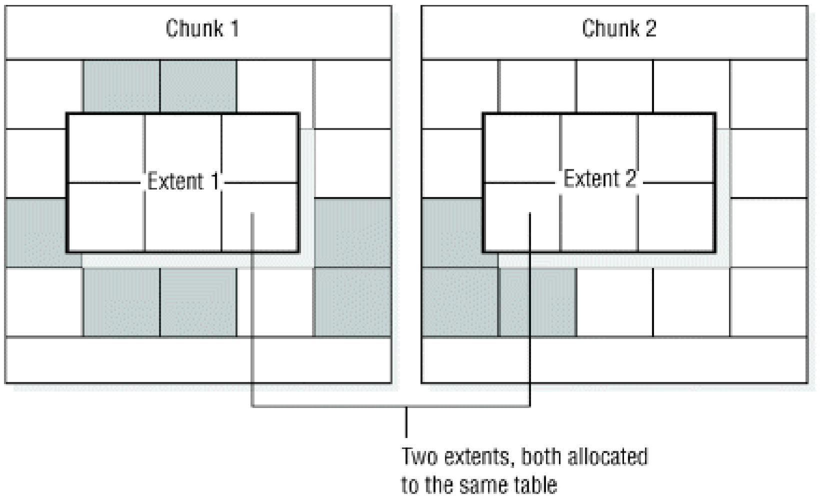
In relational database systems, a *table* is a row of column headings together with zero or more rows of data values. The row of column headings identifies one or more columns and a data type for each column.

When you create a table, the database server allocates disk space for the table in a block of pages that is called an extent. 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 (with the IN *dbspace* clause of the CREATE TABLE statement). When you do not specify the dbspace, the database server places the table in the dbspace where the database is located. You can fragment a table over more than one dbspace or within a dbspace by specifying a fragmentation distribution scheme. However, if you set the AUTOLOCATE configuration parameter to a positive integer, the database server automatically fragments new tables by round robin, in the dbspaces that are optimal for the table.

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.

A table, which is composed of extents, can span multiple chunks, as the following figure shows.
Figure 39. Table that spans more than one chunk



Simple large objects are in blobpages in either the dbspace with the data pages of the table or in a separate blobspace.

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.

Damaged tables

There are a number of ways you can damage a table.

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 HCL OneDB™

You can create logging or nonlogging tables in a logging database on HCL OneDB™. 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 HCL OneDB™. The flag values are the hexadecimal values for each table type in the **flags** column of **systables**.

Table 24. Table types for HCL OneDB™

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.

The following statement creates a RAW table called **r_tab**:

```
CREATE RAW TABLE IF NOT EXISTS
   r_tab1 (col1 INT, col2 CHAR(40) NOT NULL);
```

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.

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 *HCL OneDB™ 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 and the LOAD statement of DB-Access. 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. Altering the table mode, however, from unlogged to logged, or from logged to unlogged, is not allowed. Altering the logging mode of a table 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 *HCL OneDB™ Performance Guide*.

For more information, see Temporary tables on page 182.

Properties of table types

These topics explain loading tables, fast recovery, and backup and restore of table types.

Loading of data into a table

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.

HCL OneDB™ 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.

For more information about how to improve the performance of loading very large tables, see your *HCL OneDB™ Performance Guide*. For more information about using ALTER TABLE to change a table from logging to nonlogging, see the *HCL OneDB™ Guide to SQL: Syntax*.

Fast recovery of table types

There are fast recovery scenarios for the table types available with HCL OneDB™.

The following table shows fast recovery scenarios for table types.

Table 25. 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.

Table 25. Fast recovery of table types (continued)

Table type	Fast recovery behavior
	Incomplete transactions in a RAW table are not rolled back.

Backup and restore of RAW tables

There are backup scenarios for the table types available on HCL OneDB™.

The following table explains backup scenarios for table types.

Table 26. Backing up tables on HCL OneDB™

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 27. Restoring tables on HCL OneDB™

Table type	Restore allowed?
Standard	Yes. Warm restore, cold restore, and point-in-time restore work.
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 specific types of temporary table.

Disk space is required for the following temporary tables:

- 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 some SQL statements.

Temporary tables can be created with 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, HCL OneDB™ uses the dbspace of the database to store the temporary table. See the *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ 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.

Important: In addition to temporary tables, the database server uses temporary disk space to store the before images of data records that are overwritten while backups are occurring, and for overflow from query processing that occurs in memory. Make sure that you have correctly set the **DBSPACETEMP** environment variable or the DBSPACETEMP configuration 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.

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 210.

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 177.)

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 and index tblspace contain certain types of pages.

The table tblspace contains the following types of pages:

- Pages allocated to data
- Pages allocated to indexes

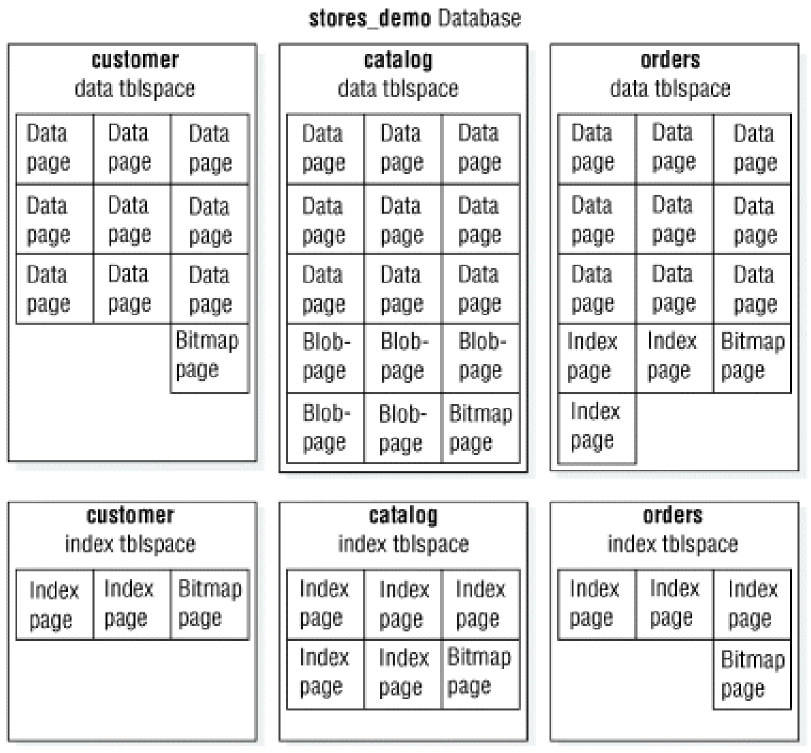
- Pages used to store TEXT or BYTE data in the dbspace (but not pages used to store TEXT or BYTE data in a blobspace)
- Bitmap pages that track page use within the table extents

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. Blobpages represent TEXT or BYTE data stored in a dbspace.

Figure 40. 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 *HCL OneDB™ Performance Guide*.

Figure 41. Three extents that belong to two different tblspaces in a single dbspace

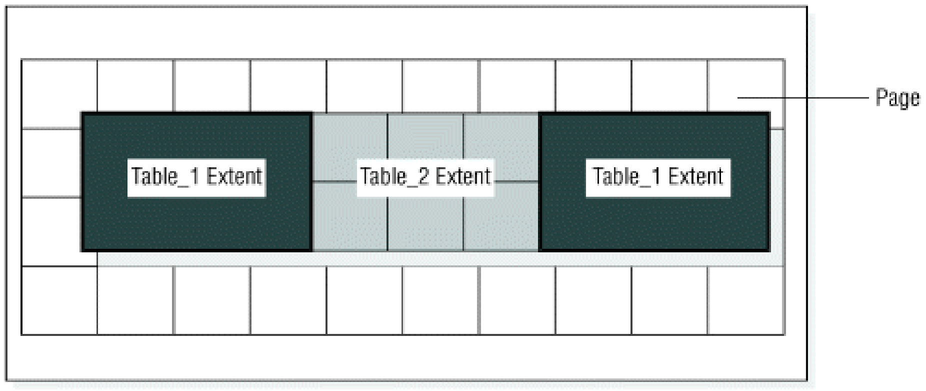


Table fragmentation and data storage

The fragmentation feature gives you more 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 on different storage spaces.

You can fragment a table in the following ways:

- Fragment a table over more than one dbspace. However, you cannot put fragments into dbspaces that have different page sizes. All fragments must have the same page size.
- Create multiple partitions of a fragmented table within a single dbspace if the fragmented table uses an expression-based or round-robin distribution scheme.

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.

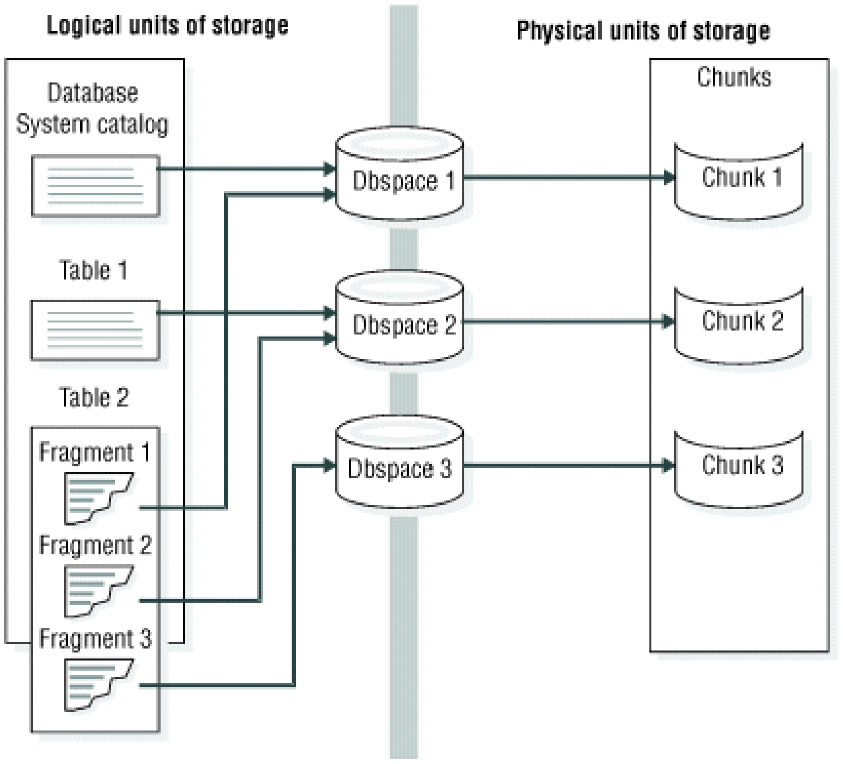
If you set the AUTOLOCATE configuration parameter or session environment variable to a positive integer, and you do not specify a location for the table, new tables are fragmented in round-robin order in dbspaces that are chosen by the database server.

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 42. Dbspaces that link logical units (including table fragments) and physical units of storage



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.

The following storage structures can be stored in the root dbspace:

Physical and logical logs (200 KB minimum for each type)

Although the root dbspace is the default location for the physical log and logical log files, move the log files to other dbspaces. You can set the AUTO_LLOG configuration parameter to specify the dbspace for logical log files. You can store the physical log in a plogspace.

Recommendation: Set up the system with a small physical log and a few small logical logs. For example, create three 1000 KB logical log files, or 3000 KB for the total log space. After the initial setup is complete, create a new dbspace for logical logs in an area that does not compete for I/O with other dbspaces, and set the AUTO_LLOG configuration parameter to that dbspace. Create a set of larger logical-log files in the dbspace for logical logs, and drop the original logs from the root dbspace. Then create a plogspace for the physical log. Make the plogspace large enough to hold your final physical log, and isolate it from other dbspaces as much as possible. This configuration optimizes logging performance and the root dbspace for the following reasons:

- The unused space that is left in the root dbspace after you move the logs is minimized.
- The physical and logical logs do not contend for space and I/O on the same disk as each other or the root dbspace.
- The server automatically increases the total logical log space and the size of the physical log if increasing logs measurably improves performance.

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 effect 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.

Temporary tables

Analyze 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 that is 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 user applications to estimate the amount of disk space that the database server might require for explicit temporary tables.

Data

Although the root dbspace is the default location for databases, do not store databases and tables in the root dbspace.

System databases (the size varies between versions)

The **sysmaster**, **sysutils**, **syscdr**, and **sysuuid** databases, and the system catalogs must be stored in the root dbspace. The **sysadmin** database is stored in the root dbspace by default, however, you can move the **sysadmin** database to a different dbspace.

Reserved pages (~24 KB)

The reserved pages contain control and tracking information that is used by the database server. Reserved pages must be stored in the root dbspace.

Tblspace tblspace (100 - 200 KB minimum)

The **tblspace** tblspace contains information about tblspaces. The **tblspace** tblspace must be stored in the root dbspace.

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.

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.

If you need to make the root dbspace larger after the server is initialized, you can add a chunk to the root dbspace. You can enable automatic space management to expand the root dbspace as needed.

Important: Mirror the root dbspace and other dbspaces that contain critical data such as the physical log and logical logs.

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 *HCL OneDB™ Performance Guide*.

The storage pool

Every instance of the database server 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, the database server 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 the database server 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 a database server instance.

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 *HCL OneDB™ 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 199.

  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 *HCL OneDB™ 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 Location of logical-log files on page 301. Also see Move logical-log files on page 325 and Change the physical-log location and size on page 337.
- 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 *HCL OneDB™ 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 *HCL OneDB™ Performance Guide*. For information about onstat options, see the *HCL OneDB™ 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

To optimize performance, use multiple storage spaces and multiple disks. The following figure shows a disk layout that is optimized for performance. This disk layout uses the following strategies to improve performance:

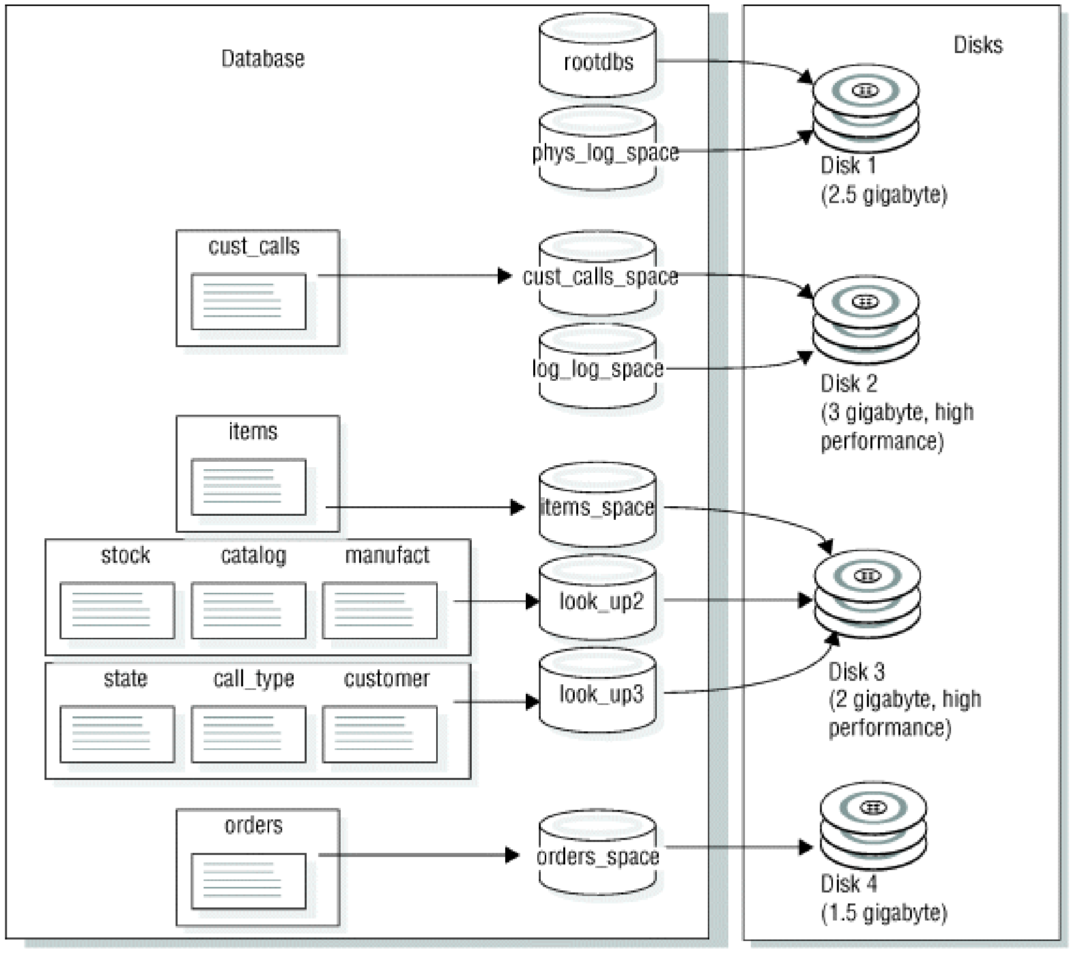
- Migration of the logical log and physical log files from the root dbspace

  This strategy separates the logical log and the physical log and reduces contention for the root dbspace. For best performance, take advantage of automatic performance tuning for the logical and physical logs:
    - Create a plogspace to enable the automatic expansion of the physical log.
    - Set the AUTO_LLOG configuration parameter to enable the automatic expansion of the logical log in a specified dbspace.

  If you create a server during installation, the plogspace is created and the AUTO_LLOG configuration parameter is set to a non-critical 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 large **cust_calls** table (the other two disks are too small for this task).

Figure 43. Disk layout optimized for performance

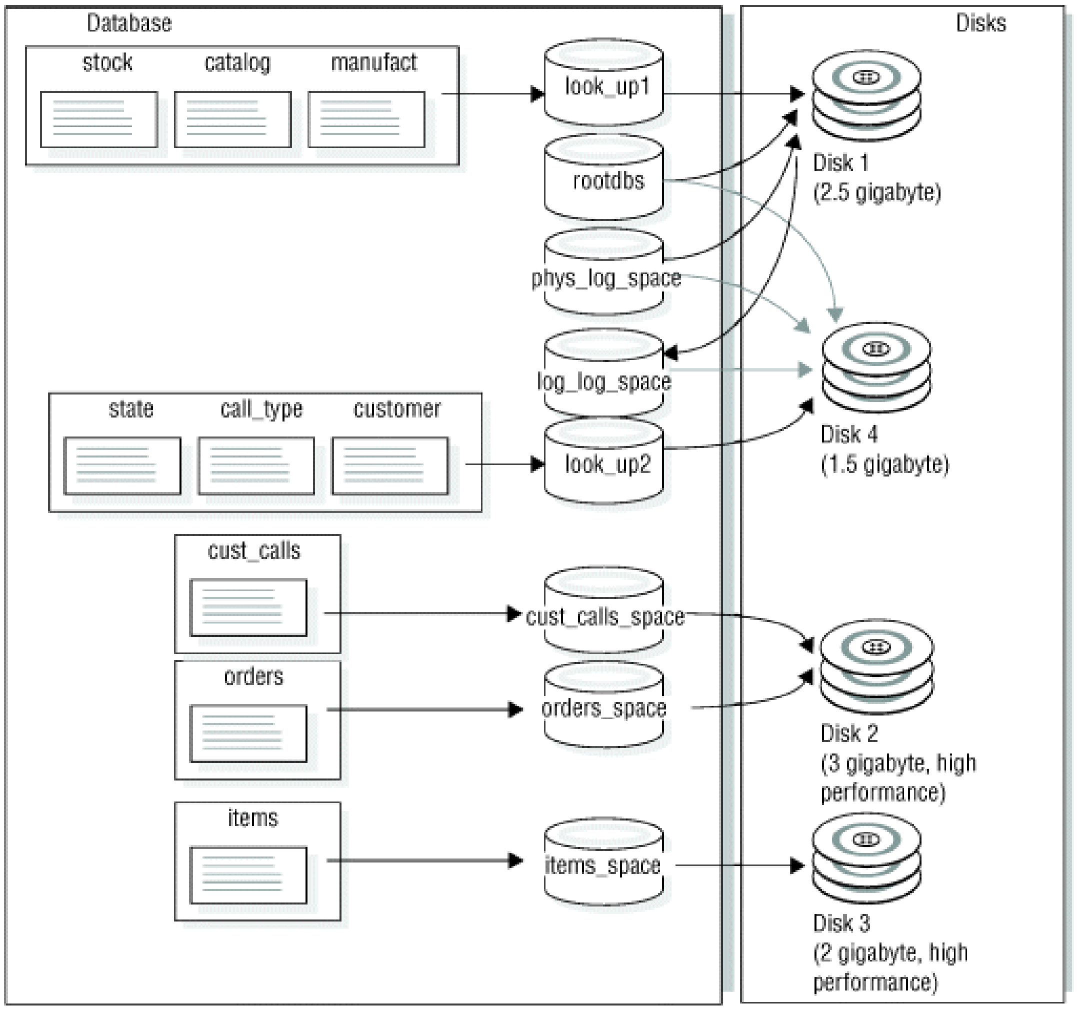


Sample layout when availability is highest priority

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 44. 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 163.)

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 343.

Manage disk space

You can use several utilities and tools to manage disk spaces and the data that the database server controls.

You can use the following utilities to manage storage spaces:

- The onspaces utility commands
- SQL administration API commands

Your *HCL OneDB™ 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.

You can generate SQL administration API or onspaces commands for reproducing the storage spaces, chunks, and logs that exist in a file with the dbschema utility.

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 158
- Amount of disk space needed to store data on page 188
- Disk-layout guidelines on page 191

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 200.

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 158.

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.

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 *HCL OneDB™ 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 204.

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.

About this task

,

To allocate disk space for a cooked file called `usr/data/my_chunk`, on UNIX™:

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.

Results

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 204, Creating a blobspace on page 215, and Creating an sbspace on page 218.

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.

About this task

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 158.
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 on page 199 and 5 on page 199 in Allocating cooked file spaces on UNIX on page 199.
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) on page 200.

Results

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. If all your partitions are FAT files, you can convert one to NTFS.

About this task

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:

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:

   Example

   ```
   c:> cd \usr\data
   ```
4. If necessary, convert the partition to NTFS by running the following command: convert /fs:ntfs
5. Create a null file with the following command: c:> copy nul my_chunk
6. If you want to verify that the file was created, use the dir command to do so.

Results

After you allocate 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 204 and Creating a blobspace on page 215.

Allocating raw disk space on Windows™

You can configure raw disk space on Windows™ as a logical drive or physical drive.

About this task

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.

   Example

   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
   ```

Results

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 200.)
- 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`.

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.

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.

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 in certain circumstances.

Perform a level-0 backup to ensure that you can restore 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 *HCL OneDB™ 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 HCL OneDB™ 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 The Scheduler on page 577 and Creating a task on page 587.

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 203.

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.

About this task

For information about creating a dbspace with a non-default page size, see Creating a dbspace with a non-default page size on page 207.

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 197.
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 207 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 on page 205.

   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 on page 206.
6. After you create the dbspace, you must perform a level-0 backup of the root dbspace and the new dbspace.

Example

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 163 and information about the onspaces utility in the *HCL OneDB™ Administrator's Reference*.

What to do next

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 207.

   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 *HCL OneDB™ 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.)

About this task

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**, HCL OneDB™ 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:
   Choose from:
   - 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.

Results

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.
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 *HCL OneDB™ 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.

About this task

The root dbspace uses the default page size. If you want to create a dbspace with a different 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.

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.

To create a dbspace with a non-default page size:

1. If you upgraded from a version before version 10.00, run the onmode -BC 2 command to enable the large chunk mode.
   By default, when HCL OneDB™ is first initialized or restarted, HCL OneDB™ starts with the large chunk mode enabled.
2. **Optional:** Create a buffer pool that corresponds to the page size of the dbspace. You can use the onparams utility or the BUFFERPOOL configuration parameter.

   If you create a dbspace with a page size that does not have a corresponding buffer pool, HCL OneDB™ automatically creates a buffer pool using the default values for the BUFFERPOOL configuration parameter as defined in the `onconfig` 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.

Example

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 - 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.

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.

Before you begin

Direct I/O must be available and the file system must support direct I/0 for the page size used for the dbspace chunk.

About this task

You can use HCL OneDB™ 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`.

Results

If you have an AIX® operating system, you can also enable concurrent I/O for HCL OneDB™ 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 *HCL OneDB™ 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 table 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** system catalog table 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` keywords in place of the `FRAGMENT BY EXPRESSION` keywords in the 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 *HCL OneDB™ 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 *HCL OneDB™ Guide to SQL: Syntax* for more syntax details, including information about named fragments in the GRANT FRAGMENT and 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.

About this task

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 204.

   Or, use the create tempdbspace command with the SQL administration API admin() or task() function.

   For more information, see create tempdbspace argument: Create a temporary dbspace (SQL administration API).
2. Use the **DBSPACETEMP** environment variable 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 mix of page sizes.

   For further information about DBSPACETEMP, see Database configuration parameters .
3. If you create more than one temporary dbspace, the dbspaces should be located on separate disks to optimize the I/O.

What to do next

After you have created 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 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
```

The equivalent SQL administration API statement:

```
EXECUTE FUNCTION task(“create tempdbspace”, “temp_space”, “/dev/raw_dev1”, “5 MB”, “5000 KB”);
```

Related information

create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API) on page

create tempdbspace argument: Create a temporary dbspace (SQL administration API) on page

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.

About this task

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 197.
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 on page 211.
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.

Example

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 OneDB 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 will run out of space and when it will run out of space.

Adding a chunk with ON-Monitor (UNIX™)

You can use ON-Monitor to add a chunk to a dbspace.

About this task

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 *HCL OneDB™ 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.

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.

Managing automatic location and fragmentation

You can control whether the database server automatically chooses the location for databases, indexes, and tables and automatically fragments tables. You can control the list of dbspaces in which the database server stores databases, indexes, and table fragments.

If you enable automatic location and fragmentation, the database server performs the following tasks:

- Stores new databases for which you do not specify a location in the optimal dbspace instead of in the root dbspace. By default, all dbspaces except dbspaces that are dedicated to tenant databases are available.
- Stores new tables and indexes for which you do not specify a location in the optimal dbspace instead of in the same dbspace as the database.
- Allocates an initial number of round-robin fragments for new tables. A table fragment does not have an extent until a row is inserted into the fragment, unless you include the FIRST EXTENT clause in the CREATE TABLE statement.
- Adds more table fragments as the table grows.

To enable automatic location and fragmentation, set the AUTOLOCATE configuration parameter or the AUTOLOCATE session environment variable to a positive integer.

Automatic location is not applicable to tenant databases or the tables, fragments, and indexes within tenant databases.

To view the list of available dbspaces, query the **sysautolocate** system catalog table.

To add a dbspace to the list of available dbspaces, run the task() or admin() SQL administration API function with the autolocate database, the autolocate database add, or the autolocate database anywhere argument.

To remove a dbspace from the list of available dbspaces, run the task() or admin() SQL administration API function with the autolocate database remove argument.

To disable automatic location and fragmentation for tables in a particular database, run the task() or admin() SQL administration API function with the autolocate database off argument.

To disable automatic location and fragmentation of tables in all databases, set the AUTOLOCATE configuration parameter or the AUTOLOCATE session environment variable to 0.

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 211.

For information about monitoring blobspaces, see Monitor storage spaces on page 203

Creating a blobspace

You can use onspaces or ON-Monitor to create a blobspace.

Before you begin

Before you create a blobspace:

1. Allocate disk space for the blobspace, as described in Allocate disk space on page 197.
2. Determine what blobpage size is optimal for your environment.

   For instructions, see Determine blobpage size on page 216.

About this task

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 216. 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 211.
4. After you create the blobspace, you must perform a level-0 backup of the root dbspace and the new blobspace.

Example

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 *HCL OneDB™ Administrator's Reference*.

What to do next

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 217. 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 306.)

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 *HCL OneDB™ 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 177.

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, you can use database server utility commands

The following database server utility commands to determine the optimal blobpage size:

- 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 247, Determine blobpage fullness with oncheck -pB on page 247, and optimizing blobspace blobpage size in the topics about table performance considerations in the *HCL OneDB™ 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 203.

Creating an sbspace

Use the onspaces utility or HCL® OneDB® Server Administrator (ISA) to create an sbspace.

About this task

Use the onspaces utility 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 169.

   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 219.
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 249.

Example

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 *HCL OneDB™ Administrator's Reference*. For information about creating smart large objects, see the *HCL OneDB™ DataBlade® API Programmer's Guide* and *HCL OneDB™ ESQL/C Programmer's Manual*.

What to do next

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. It is important to size the metadata area for an sbspace correctly to ensure that the sbspace does not run out of metadata space.

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.

To ensure that this does not happen, 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 *HCL OneDB™ Performance Guide*. Also see Monitoring the metadata and user-data areas on page 253.

Adding a chunk to an sbspace

You can add a chunk to an sbspace or temporary sbspace.

About this task

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.

Example

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 OneDB 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.

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 169 and managing sbspaces in the topics about table performance considerations in your *HCL OneDB™ Performance Guide*.

Creating a temporary sbspace

About this task

For background information and the rules for determining where temporary smart large objects are stored, see Temporary sbspaces on page 174. 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 197.

   For information about SBSPACETEMP, see the configuration parameters topics in the *HCL OneDB™ 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 *HCL OneDB™ 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 :
   ```
   ifx_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
   ```

Results

For information about creating smart large objects, see the *HCL OneDB™ DataBlade® API Programmer's Guide* and *HCL OneDB™ ESQL/C Programmer's Manual*.

Manage the plogspace

You create or move the plogspace with the onspaces utility or equivalent SQL administration API command.

To create the plogspace, run the onspaces -c -P command or the admin() or task() SQL administration API function with the create plogspace argument.

If you want to change the location of the plogspace to a different chunk, create a new plogspace. The physical log is moved to the new plogspace and the old plogspace is dropped.

You can modify the chunk in the plogspace in the following ways:

- Mark the chunk as not extendable. Run the admin() or task() SQL administration API function with the modify chunk extendable off argument
- Change the extend size of the chunk. The default extend size is 10000 KB. Run the admin() or task() SQL administration API function with the modify space sp_sizes argument.

Automatic space management

You can configure the server to add more storage space automatically when more space is required. You use space more effectively and ensure that space is allocated as necessary, while reducing out-of-space errors. You reduce the time required to manually monitor your spaces to determine which storage space might run out of free 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 dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace, the server can add a chunk to the storage space. The server can also extend a chunk in a dbspace, plogspace, or temporary dbspace that is not mirrored.When the server expands a 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 OneDB 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.

By default, the SP_AUTOEXPAND configuration parameter is set to 1 to enable automatic expansion of storage spaces. If you do not want to the server to automatically expand space, 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.

Tip:

In some situations, the database server might not automatically expand a temporary dbspace that is listed in the DBSPACETEMP configuration parameter after you configured the server to automatically expand an existing storage space. If operations (such as an index build or sort) that use the temporary dbspace run out of space, you receive an out of space error. To work around this problem, you must manually add a chunk to the temporary dbspace or use a bigger temporary dbspace.

If you have a storage pool and the database server participates in Enterprise Replication, storage spaces that are necessary for replication are created automatically if needed when you define a replication server.

Creating and managing storage pool entries

You can add, modify, and delete the entries in the storage pool.

About this task

Each entry in the storage pool contains information about a directory, cooked file, or raw device that a database server instance can use if necessary to automatically expand an existing storage space.

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 OneDB can begin allocating space.
- The total space available to OneDB 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 KB. Therefore, the minimum chunk size that you can specify is 1000 KB.
- 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");
```

Example

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 MB, 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 GB, 10 MB, 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");
```

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.

About this task

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");
```

Example

The following command specifies that chunk 12 can be extended:

```
EXECUTE FUNCTION task("modify chunk extendable", "12");
```

Modifying the sizes of an extendable storage space

You can control how an extendable storage space in the storage pool grows by modifying the create size and the extend size. By default, the maximum size of an extendable storage space is unlimited. You can limit the size of an extendable storage space by setting a maximum size.

To modify the create, extend, or maximum size of a storage space:To modify the create or extend size of a storage space:

Run the admin() or task() SQL administration API 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", "max_size");
```

```
EXECUTE FUNCTION task("modify space sp_sizes", "space_name",
 "new_create_size", "new_extend_size");
```

The *space_name* is the name of the storage space.

The *new_create_size* is the minimum size that the server can use to create a new chunk in the specified dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

The *new_extend_size* is the minimum size that the server can use to extend a chunk in the specified unmirrored dbspace or temporary dbspace.

Specify either sizes with a number (for the number of KB) or a percentage (for a percentage of the total space).

The *max_size* is the maximum size, in KB, to which the server can expand the storage space. A value of 0 indicates unlimited.

Example

The following command sets the create size to 60 MB, the extend size to 10 MB, and the maximum size to 200 MB for a space that is named **dbspace3**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace3", "60000",
   "10000", "200000");
```

The following command sets the create size to 60 MB and the extend size 10 MB for a space that is named **dbspace3**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace3", "60000", "10000");
```

The following command sets the create size to 20 percent and the extend size to 1.5 percent for a space that is named **logdbs**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "logdbs", "20", "1.5");
```

Changing the threshold and wait time for the automatic addition of more space

While OneDB 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.

About this task

Specify a threshold for the minimum amount of free KB in a storage space to trigger 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.

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.

About this task

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 storage space.

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;
```

Example

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 OneDB to automatically expand the space or extend a chunk.

Before you begin

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.

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 OneDB storage pool, and loading tables into the space. When the space fills, OneDB automatically expands it.

About this task

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 . 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 OneDB storage pool.

   For example, add the `$ONEDB_HOME/tmp` directory to the storage pool, as follows:

```
DATABASE sysadmin;
EXECUTE FUNCTION task("storagepool add", "$ONEDB_HOME/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 OneDB 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, OneDB automatically creates a cooked file in the `$ONEDB_HOME/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 storage space 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 storage space. Then try to allocate more pages. The allocation must be successful and you must not receive an out-of-space error.

   OneDB 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 `$ONEDB_HOME/tmp` directory to the storage pool, as follows:

   ```
   DATABASE sysadmin;
   EXECUTE FUNCTION task("storagepool add", "$ONEDB_HOME/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 OneDB 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. **Optional:** 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. **Optional:** 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. **Optional:** Change two sizes that are associated with expanding a storage space:
   Choose from:
   - The extend size, which is the minimum size that is used when extending a chunk in a dbspace, temporary dbspace, or the plogspace
   - The extend size, which is the minimum size that is used when extending a chunk in a dbspace or temporary dbspace
   - The create size, which is the minimum size that is 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 MB and 10 MB, respectively, for space number 3:

     ```
     EXECUTE FUNCTION task("modify dbspace sp_sizes",
      "3", "60000", "10000");
     ```

What to do next

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 the onspaces utility 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.

Table 28. Database server modes for dropping chunks

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
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 *HCL OneDB™ Administrator's Reference*.

For information about dropping a chunk from a dbspace with onspaces, see the *HCL OneDB™ 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 on page 231 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 *HCL OneDB™ 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 354.) 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 *HCL OneDB™ Administrator's Reference*.

Dropping a dbspace or blobspace with ON-Monitor (UNIX™)

You can drop a dbspace or blobspace with ON-Monitor.

About this task

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**.

What to do next

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 *HCL OneDB™ 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.

Before you begin

Prerequisite: The storage pool must contain entries (a directory, cooked file, or raw device).

About this task

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",);
  ```

Example

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");
```

Returning empty space to the storage pool

You can return the space from an empty chunk or storage space to the storage pool.

About this task

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");`

Example

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");
```

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 176.

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 *HCL OneDB™ 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 *HCL OneDB™ Virtual-Table Interface Programmer's Guide*.

For reference information about creating an extspace with onspaces, see information about the onspaces utility in the *HCL OneDB™ 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 *HCL OneDB™ 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 245.)

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 240.)

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 240.

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 *HCL OneDB™ Administrator's Reference*.

Using HCL® OneDB® Server Administrator

About this task

To query **sysdatabases** using HCL® OneDB® 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 *HCL OneDB™ 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 on page 241.

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 *HCL OneDB™ 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 *HCL OneDB™ 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
```

HCL® OneDB® Server Administrator

You can perform the following tasks using HCL® OneDB® 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 *HCL OneDB™ 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 **IONEDB_ OPCACHE** environment variable, if set

If you do not set the **IONEDB_ OPCACHE** 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 *HCL OneDB™ 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.

Example

```
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
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

Command	Description
	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 *HCL OneDB™ Performance Guide*.

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 *HCL OneDB™ 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 on page 250.

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.

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 Remote administration with the SQL administration API on page 599 and the *HCL OneDB™ Administrator's Reference*.

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
```

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
```

Monitoring the metadata and user-data areas

About this task

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.
   Example
   For an example, see The oncheck -ce and oncheck -pe options on page 250.
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 219.
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.

Results

For more information, see Size sbspace metadata on page 219.

Multitenancy

You can segregate data, storage space, and processing resources for multiple client organizations by creating multiple tenant databases in a single instance of HCL OneDB™.

For example, assume that you want to provide payroll services to small businesses. You sell the use of the payroll application as a service to small business clients. Instead of providing a separate HCL OneDB™ instance to each client, you can configure a tenant database for each client in a single HCL OneDB™ instance.

When you configure multitenancy, you segregate the following aspects of a database server:

Data

You create a separate tenant database for each client.

Storage spaces

Each tenant database has dedicated storage spaces to store data. Tables, fragments, and indexes that are created in the tenant database must be created in the dedicated storage spaces. Only the tenant database can use the dedicated storage spaces.

You can limit the amount of permanent storage space that is available to a tenant database to conserve system resources.

Temporary storage spaces can be dedicated to a specific tenant database or shared between databases.

You can encrypt tenant storage spaces if the DISK_ENCRYPTION configuration parameter is set. Each encrypted storage space has a separate encryption key.

Users

You can set permissions for client users to access each tenant database. You can grant certain users permission to create, modify, or drop tenant databases. By default, only a DBA or user **informix** can create a tenant database.

Processing resources

You can segregate CPU resources for a tenant database by defining a tenant virtual processor class and creating virtual processors for running the session threads for the tenant database. Otherwise, the session threads for tenant databases have access to all CPU virtual processors.

Session limits

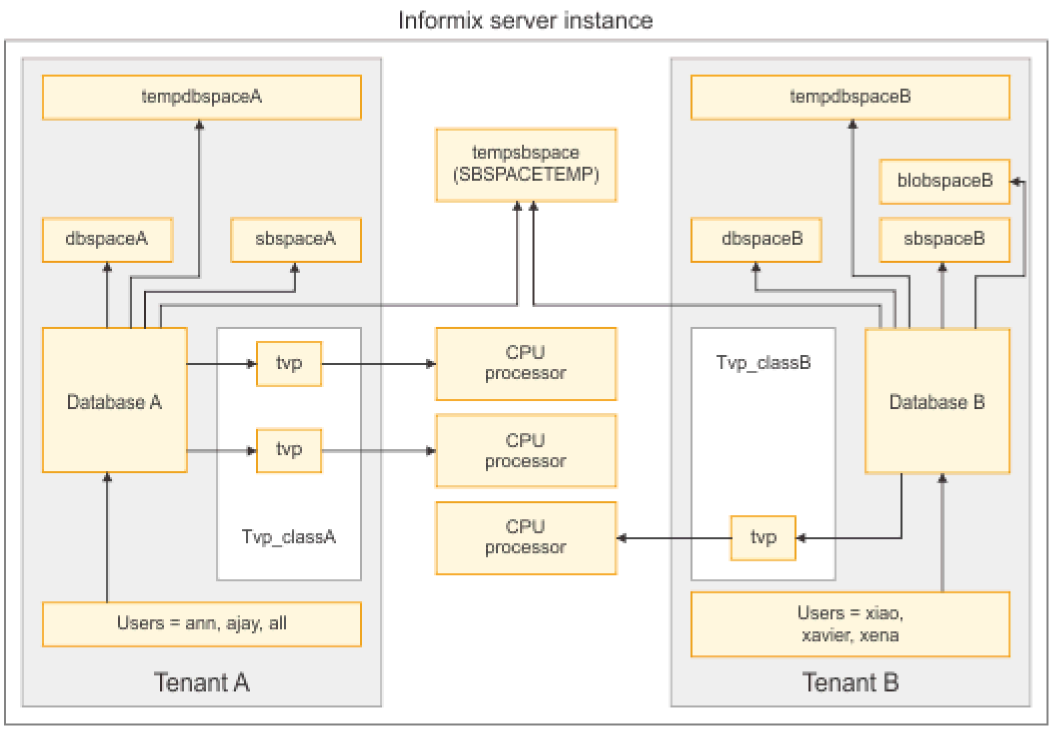
You can set the following limits for tenant sessions:

- The number of locks a tenant session can acquire.
- The amount of memory that can be allocated for a session.
- The amount of temporary storage space that can be allocated for a session.
- The size of transactions within a session, based on the amount of log space that individual transactions would fill.
- The amount of time that a transaction is allowed to run within a session.
- The amount of shared memory for all sessions that are connected to the tenant database.
- The number of client connections to a tenant database.

The following illustration shows a possible configuration for two clients in the HCL OneDB™ server instance. Each client has a database and users who are allowed to access the tenant database. Each tenant database has their own storage spaces.

Both tenant databases share the default temporary sbspace. Tenant A has a tenant virtual processor class with two virtual processors, while Tenant B has a virtual process class with one virtual processor.

Figure 45. Multiple tenants in the HCL OneDB™ server instance



Replication and tenant databases

You can replicate tenant databases with Enterprise Replication and high-availability clusters.

You can run the commands to create, modify, or delete tenant databases through an Enterprise Replication grid.

You cannot run the commands to create, modify, or delete tenant databases from an updatable secondary server in a high-availability cluster.

Backup and restore of tenant databases

You can back up tenant databases as part of a database server backup or by specifying the tenant storage spaces in the backup command. You can restore a single tenant database with ON-Bar by specifying the -T option in the onbar -r command.

If storage space encryption is enabled, you can encrypt all storage spaces that are assigned to a tenant during a restore. Whether storage space encryption is enabled or not enabled, you can decrypt all tenant storage spaces during a restore.

Creating a tenant database

You can create a tenant database to segregate data, storage, and processing resources to a specific client organization.

Before you begin

You must be user **informix**, a DBSA, or have the TENANT privilege to create a tenant database.

About this task

You cannot convert an existing database to a tenant database. You cannot convert a tenant database to a non-tenant database. You cannot run the CREATE DATABASE statement to create a tenant database.

To create a tenant database:

1. Create the storage spaces for the tenant database. All dedicated storage spaces must be empty when you create the tenant database.
   You can create the following types of dedicated spaces for a tenant database:

   dbspaces

   You must create at least one dbspace for the tenant database. The tenant database must be stored in one or more dedicated dbspaces.

   blobspaces

   If the tenant database will contain simple large objects, you must create one or more blobspaces.

   sbspaces

   If the tenant database will contain smart large objects, you must create one or more sbspaces. Smart large objects can include BLOB or CLOB data, or data and table statistics that are too large to fit in a row. Some HCL OneDB™ features, such as Enterprise Replication, spatial data, and basic text searching, require sbspaces.

   temporary dbspaces

   Optional: Create one or more temporary dbspaces to store temporary tables. Otherwise, temporary tables are stored in the temporary dbspaces that are specified by the DBSPACETEMP configuration parameter or environment variable.

   temporary sbspaces

   Optional: Create one or more temporary sbspaces to store temporary smart large objects. Otherwise, temporary smart large objects are stored in the temporary sbspaces that are specified by the SBSPACETEMP configuration parameter.
2. **Optional:** Set limits for the tenant database so that it cannot monopolize system resources.
   Tenant database limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. You can set the following limits for a tenant database:

   Locks available to a session

   Set the session_limit_locks property to specify the maximum number of locks available to a session.

   Logspace available to transactions in a session

   Set the session_limit_logspace property to specify the maximum amount of log space that a session can use for individual transactions.

Memory available to a session

Set the session_limit_memory property to specify the maximum amount of memory that a session can allocate.

Temporary table space available to a session

Set the session_limit_tempspace property to specify the maximum amount of temporary table space that a session can allocate.

Amount of time that a transaction can run

Set the session_limit_txn_time property to specify the maximum amount of time that a transaction can run in a session

Total space available to a tenant database

Set the tenant_limit_space property to specify the maximum amount of storage space available to a tenant user.

3. Set up a storage pool so that storage spaces can grow automatically. You can specify maximum sizes for extendable storage spaces to limit the growth of tenant databases.
4. Provide TENANT privileges to specific users to create, modify, and delete tenant databases.
   Example
   For example, the following command gives the user **jsmith** TENANT privileges:
   ```
   EXECUTE FUNCTION task("grant admin", "jsmith", "tenant");
   ```
5. Create a tenant database and define its properties by running the admin() or task() SQL administration API function with the tenant create argument.
   Example
   For example, the following statement creates a tenant database that is named **companyA**:
   ```
   EXECUTE FUNCTION task('tenant create', 'companyA',
         '{dbspace:"companyA_dbs1,companyA_dbs2", sbspace:"companyA_sbs1",
           vpclass:"tvp_A,num=2", logmode:"ansi"}');
   ```
   The tenant database has two dbspaces, an sbspace, two tenant virtual processors, and the ANSI logging mode.

What to do next

When you explicitly specify storage locations during the creation or altering of tables and indexes in the tenant database, you must specify the dbspaces that are listed in the tenant database definition. Otherwise, the statement fails. If you do not explicitly specify storage for tables or indexes, they are created in the first dbspace that is listed in the tenant definition.

Note: Improve the security of your databases by performing the following tasks:

- Run GRANT and REVOKE statements to control user access to databases.
- Set the DBCREATE_PERMISSION configuration parameter to restrict the ability to create non-tenant databases.

Managing tenant databases

You can view the properties of tenant databases, update the properties of tenant databases, and delete tenant databases.

Viewing tenant database properties

To view the tenant database definition, query the **tenant** table in the **sysadmin** database. For example, the following statement lists the tenant databases and their properties:

```
SELECT hex(tenant_id),tenant_dbsname,tenant_resources::json,
       tenant_create_time,tenant_last_updated
FROM tenant;
```

The **tenant_resources** column, which contains the tenant properties, is of type BSON, so you must cast the column to JSON to view the properties.

Updating the properties of tenant databases

To update properties, run the admin() or task() SQL administration API function with the tenant update argument. The updates take effect for new sessions.

You can append dbspaces, blobspaces, or sbspaces to the existing lists of storage spaces for a tenant database. The storage spaces must be empty. You must have DBA or TENANT privileges to change tenant database properties.

You cannot remove dedicated storage spaces from a tenant database unless you delete the database.

When you specify new values for the following tenant database properties, existing values are replaced.

- dbspacetemp (temporary dbspaces that are assigned to the tenant)
- session_limit_logspace (limit on log space for individual transactions)
- session_limit_memory (limit on memory that is allocated per session)
- session_limit_tempspace (limit on temporary table space per session)
- session_limit_txn_time (limit on the length of time a transaction can run)
- sbspacetemp (temporary sbspaces that are assigned to the tenant)
- session_limit_locks (limit on the number of locks per session)
- tenant_limit_space (limit on total storage space)
- vpclass (virtual processor classes names and quantities)

Deleting tenant databases

To delete a tenant database, run the admin() or task() SQL administration API function with the tenant drop argument. You must have DBA or TENANT privileges to delete tenant databases. You cannot delete a tenant database with the DROP

DATABASE statement. All dedicated storage spaces for the tenant database are emptied and become available. Any tenant virtual processors that are not shared with other tenant databases are dropped.

Restoring a tenant database to a point in time

You can restore a tenant database to a point in time with the ON-Bar utility.

Before you begin

You must meet the following prerequisites before you start a tenant restore:

- The tenant database storage spaces must have a physical backup.
- The database server must be online.
- All tenant storage spaces that are listed in the tenant definition must exist.
- Tenant storage spaces must have unique names for existing backups. For example, if a tenant database that has backups is dropped and storage spaces with the same names are used in a new tenant database, the backups of the dropped tenant database must be removed.
- No other warm restores or tenant restores are in progress.

A tenant database point-in-time restore has the following additional prerequisites if the tenant database is in a high-availability cluster:

- The cluster cannot include shared-disk secondary servers or updatable secondary servers. The cluster can include only read-only HDR secondary servers and remote stand-alone secondary servers.
- All secondary servers must be online.
- You can run the restore command only on the primary server.

  However, during the tenant point in time restore process, internally generated commands are run on the secondary servers to restore the new state of the tenant database spaces on the secondaries. A tenant point in time restore command includes new physical backups of the tenant spaces in their new state. The tenant point in time restore command that is run on the primary server does not return until all secondary servers acknowledge the completion of the automatically generated restores of the new tenant space backups.

About this task

Like other warm restores, the logical logs that are required for the tenant database restore are restored to the temporary spaces that are specified by the DBSPACETEMP configuration parameter. If the DBSPACETEMP configuration parameter is not set, temporary files are created in the root dbspace.

To restore a tenant database to a point in time, run the following command:

```
onbar -r -T tenant_database -t "time" -O
```

Substitute *tenant_database* with the name of the tenant database. Substitute *time* with the time of the last transaction to be restored from the logical logs.

If you omit the -O option, all the permanent tenant spaces must be marked as down before you run the restore command. Temporary spaces are never backed up or restored.

Results

If the restore fails, fix the problem and run the restore again. Until the restore succeeds, the tenant database is blocked from accepting connections. During the restore, the value of the **tenant_state** field is set to `restoring` in the **tenant_resources** column of the **sysadmin:tenant** table. When a tenant database is blocked, the value of the **tenant_state** field is set to `blocked`. You can view the value of the **tenant_state** field by running the following query:

```
SELECT bson_value_lvarchar(tenant_resources, 'tenant_state') AS tenant_state
   FROM sysadmin:tenant
   WHERE tenant_dbsname = 'tenant_dbname';
```

Substitute *tenant_dbname* with the name of the tenant database.

Example

Example

The following command restores a tenant database that is named **tenant1** to the specified point in time:

```
onbar -r -T tenant1 -t "2015-10-10 11:35:57" -O
```

Storage optimization

Data compression and consolidation processes can minimize the disk space that is used by your data and indexes.Data compression and consolidation processes can minimize the disk space that is used by your data.

The following table describes the processes that you can use to reduce the amount of disk space that is used by data in rows, simple large objects in dbspaces, and index keys. You can automate any or all of these processes or do them as needed.

The following table describes the processes that you can use to reduce the amount of disk space that is used by data. You can automate any or all of these processes or do them as needed.

Table 29. Storage optimization processes

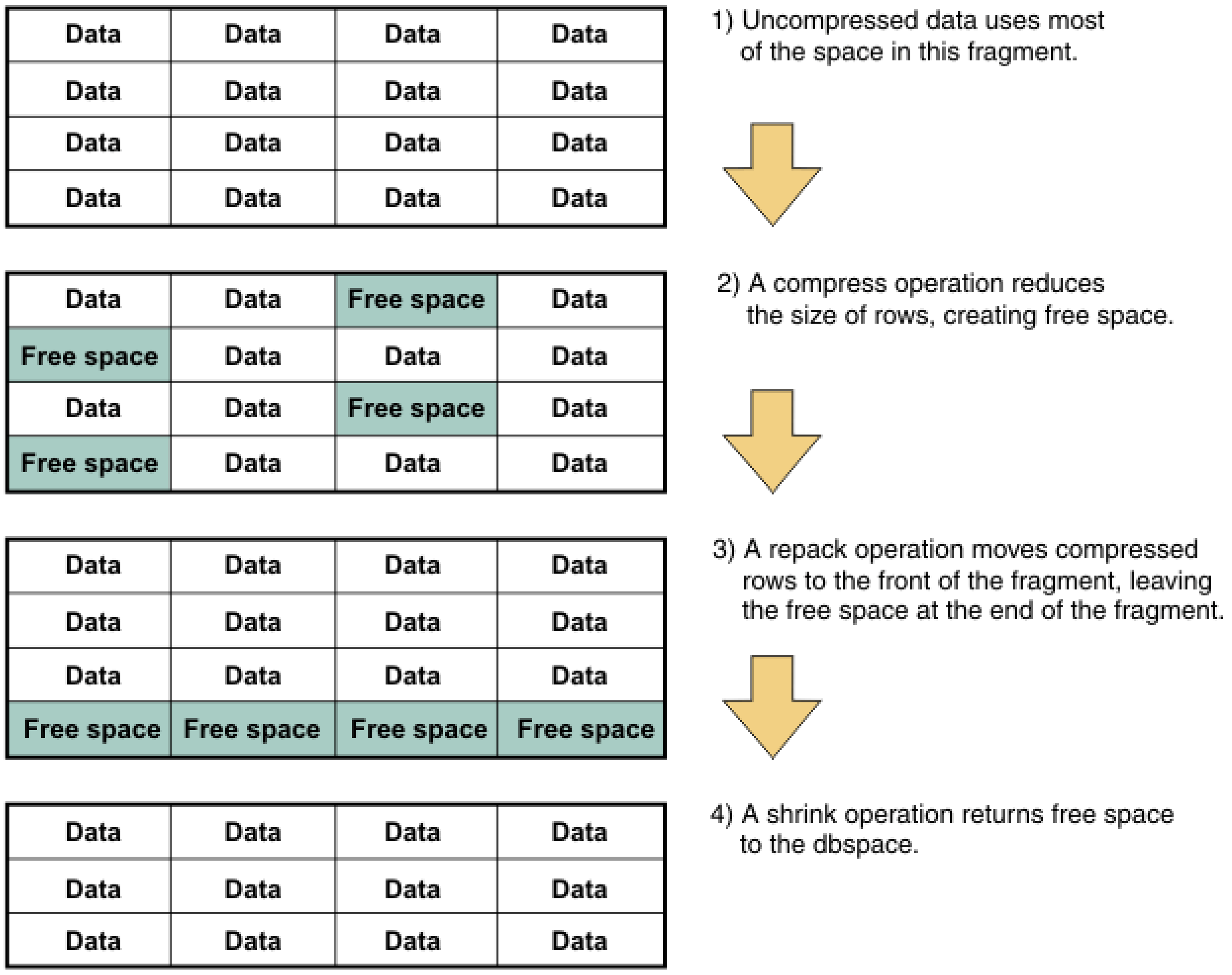
Storage optimization process	Purpose	When to use
Compressing	Compresses data in table or fragment rows, reducing the amount of required disk space Compresses data in tables and fragments, compress simple large objects in dbspaces, and	When you want to reduce the size of 2000 or more rows of data, simple large objects in dbspaces, or 2000 or more index keys When you

Table 29. Storage optimization processes (continued)

Storage optimization process	Purpose	When to use
	compresses keys in indexes. Reduces the amount of required disk space. After you enable compression, new data or index keys is automatically compressed.	want to reduce the size of 2000 or more rows of data in a table
Repacking	Consolidates free space in tables, fragments, and indexes.Consolidates free space in tables and fragments.	After you compress or when you want to consolidate free space
Shrinking	Returns free space to the dbspace.	After you compress or repack or when you want to return free space to the dbspace
Defragmenting	Brings data rows closer together in contiguous, merged extents.Brings data rows or index keys closer together in contiguous, merged extents.	When frequently updated tables or indexes become scattered among multiple non-contiguous extentsWhen frequently updated tables become scattered among multiple non-contiguous extents

The following illustration 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. The process for storage optimization of indexes is the same.

Figure 46. Data in a fragment during the compression and storage optimization process


Data
Free space
1) Uncompressed data uses most of the space in this fragment.
2) A compress operation reduces the size of rows, creating 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.
4) A shrink operation returns free space to the dbspace.

Storage optimization methods

You can optimize individual tables, fragments, or indexes. You can schedule the automatic optimization of all tables and fragments. You can optimize individual tables or fragments. You can schedule the automatic optimization of all tables and fragments.

You can use the COMPRESSED option in the CREATE TABLE statement to enable automatic compression of the table when the table has at least 2000 rows.

You can use the COMPRESSED option in the CREATE INDEX statement to enable automatic compression of the index if the index has 2000 or more keys. Compression is not enabled if the index has fewer than 2000 keys.

You can use the SQL administration API task or admin function to perform any type of storage optimization on a table, fragment, or index.

You can use the SQL administration API task or admin function to perform any type of storage optimization on a table or fragment.

You can enable the **auto_crsd** Scheduler task to automatically compress, repack, shrink, and defragment all tables and table fragments.

Table 30. Methods of storage optimization

Goal	SQL statement	SQL administration API argument	Scheduler task	OAT page
Automatically compress data for a table or fragment	CREATE TABLE with the COMPRESSED option	table compress or fragment compress		**Storage**
Automatically compress data for all tables and fragments			**auto_crsd**	**Server Optimization Policies**
Repack and shrink a table or fragment		table repack shrink or fragment repack shrink		**Storage**
Automatically repack and shrink all tables and fragments			**auto_crsd**	**Server Optimization Policies**
Automatically compress a B-tree index	CREATE INDEX with the COMPRESSED option	index compress		**Storage**
Repack and shrink a B-tree index		index repack shrink		**Storage**
Defragment a table of fragment		defragment		**Storage**
Automatically defragment all tables and fragments			**auto_crsd**	**Server Optimization Policies**

Scheduling data optimization

You can configure the automatic compressing, shrinking, repacking, and defragmenting of all tables and extents by enabling the **auto_crsd** Scheduler task.

About this task

You can enable and configure the **auto_crsd** task by updating Scheduler tables in the **sysadmin** database.

To enable the **auto_crsd** task by updating the Scheduler tables:

1. Connect to the **sysadmin** database as user **informix** or another authorized user.
2. 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:
   Example

```
UPDATE ph_task
  SET tk_enable = 'T'
  WHERE tk_name = 'auto_crsd';
```

3. **Optional:** Change the frequency of when the task is run by running an UPDATE statement on the **ph_task** table to change the value of the **tk_frequency** column. The default value is 7 00:00:00, which indicates that the task runs once a week. For example, the following statement changes the frequency to once a day:
   Example
   ```
   UPDATE ph_task
     SET tk_frequency = '1 00:00:00'
     WHERE tk_name = 'auto_crsd';
   ```
4. **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

   Example
   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';
   ```
5. **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 50 000 rows. A table is compressed when the number of uncompressed rows exceeds 50 000.
   - AUTOREPACK_SPACE: The threshold for repacking a table is the percentage of noncontiguous space. The default is 90%. A table is repacked when more than 90% of the space the table occupies is noncontiguous.
   - AUTOSHRINK_UNUSED: The threshold for shrinking a table or fragment is the percentage of unused, allocated space. The default is 50%. A table or fragment is shrunk when more than 50% of the allocated space is unused.
   - AUTODEFRAG_EXTENTS: The threshold for defragmenting table or fragment extents is the number of extents. The default is 100. A table or fragment is defragmented when the number of extents exceeds 100.

   Example
   For example, the following statement changes the compression threshold to 5000 rows:
   ```
   UPDATE ph_threshold
     SET value = '5000'
     WHERE name = 'AUTOCOMPRESS_ROWS';
   ```

Results
When a threshold for an operation that you enabled is exceeded, the Scheduler runs the operation.

Example: Optimizing data storage on demand

In this example, you learn how to run SQL administration API commands to determine how much space you can save by compressing a table, how to compress the table, and how to optimize storage on demand. You also learn how to uncompress the table and remove the compression dictionaries.

Before you begin

Assume that you have a table named **rock** in a database named **music** that is owned by user **mario**. The **rock** table is not fragmented. You can run the same operations on a table fragment as you can on a whole table, but the syntax is slightly different.

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 whole table.
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Logical and physical logs are large enough to handle normal processing and compression operations. Compression, repacking, and uncompressing, operations can use large amounts of logs.

About this task

To compress both row data and simple large objects in dbspaces:

To compress and uncompress row data:

1. 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 that is 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 HCL OneDB™ uses to compress data in the **rock** table. You run the following command

   ```
   EXECUTE FUNCTION task("table create_dictionary", "rock", "music", "mario");
   ```

   Tip: If you do not create the compression dictionary as a separate step, HCL OneDB™ creates the dictionary automatically when you compress data.
3. You decide that you want to compress data in the **rock** table and simple large objects in dbspaces, 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");
   ```

   You can perform the same operations faster by running them in parallel. You run the following command:

   ```
   EXECUTE FUNCTION task("table compress repack shrink parallel", "rock",
   "music", "mario");
   ```

You can adjust the command by specifying what you want to compress or shrink. For example:

- To compress only row data, specify:

  ```
  EXECUTE FUNCTION task("table compress rows parallel","rock","music","mario");
  ```

- To compress only row data and then repack and shrink the data, specify:

  ```
  EXECUTE FUNCTION task("table compress repack shrink rows parallel",
  "rock","music","mario");
  ```

- To compress only simple large objects in the dbspace, specify:

  ```
  EXECUTE FUNCTION task("table compress blobs parallel","rock","music","mario");
  ```

After the existing rows and simple large objects are compressed, HCL OneDB™ 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.

If the simple large objects or rows are not smaller when compressed, the database server does not compress them.

4. Now suppose that you want to uncompress the data. You run the following command:

   ```
   EXECUTE FUNCTION task("table uncompress", "rock", "music", "mario");
   ```

   ```
   EXECUTE FUNCTION task("table uncompress parallel", "rock", "music", "mario");
   ```

5. You want to remove the compression dictionary.

   a. Verify that Enterprise Replication does not require the dictionary.

      If you do require the dictionaries for Enterprise Replication, do not remove compression dictionaries for uncompressed or dropped tables and fragments.

   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");
      ```

What to do next

To run compression and other storage optimization commands on table fragments, include the fragment argument instead of the table argument and the fragment partition number instead of the table name.

```
EXECUTE FUNCTION task("fragment command_arguments", "partnum_list");
```

Partition defragmentation

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.

Before you defragment a table, index, or partition, be sure that none of the following conflicting operations are in progress:

- An existing defragment operation on the table, index, or dbspace.
- DDL statements, such as DROP TABLE or ALTER FRAGMENT, are being run on the table or partition.
- The table is being truncated.
- The table is being compressed or uncompressed.
- An online index build is running.

You cannot defragment the following objects:

- Pseudo tables, such as virtual-table interface (VTI) tables
- Tables with virtual-index interface (VII) indexes
- Tables with functional indexes
- Temporary tables
- Sort files
- A table that has exclusive access set
- Optical BLOB files

To determine how many extents a table, index, or partition has, you can run the oncheck -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.

You cannot stop a defragment request after you run the command.

If there are problems in completing a defragment request, error messages are sent to the online log file.

Compression

You can compress and uncompress row data in tables and fragments and simple large objects in dbspaces. You can compress B-tree indexes.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, simple large objects, or indexes, consolidating data, and returning free space have the following benefits:

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 more 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:
  - 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

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.

You can compress data and indexes in parallel.

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, compressing data on one replication server does not affect the data on any other replication server.

If you use high-availability clusters, data that is compressed in the source table is compressed in the target table. You cannot perform compression operations on secondary servers, because secondary servers must have the same data and physical layout as the primary server.

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.

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.

You can compress the following types of data:

- 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.
- Simple large objects (TEXT or BYTE data types) that are stored in dbspaces.

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.

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.

HCL OneDB™ 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 rows in some types of tables and fragments. 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 the following database objects:

- Tables or fragments that are in the **sysmaster**, **sysutils**, **sysuser**, **syscdr**, and **syscdcv1** databases
- Catalogs
- Temporary tables
- Virtual-table interface tables
- The tblspace **tblspace**
- Internal partition tables
- Dictionary tables (these tables, one per dbspace, hold compression dictionaries for the fragments or tables that are compressed in that dbspace and metadata about the dictionaries.)

You cannot compress a table while an online index build is occurring on the table.

You cannot compress simple large objects in blobspaces.

You cannot compress indexes.

Compression is not applied to index data, large object 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.

HCL OneDB™ 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.

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.

HCL OneDB™ cannot compress an individual row to be smaller than four bytes long. 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.

B-tree index compression

You can compress detached B-tree indexes. You can also consolidate free space in the index and you can return free space at the end of the index to the dbspace. Before you compress an index, you can estimate the amount of disk space that you can save.

You can compress a detached B-tree index that is on a fragmented or non-fragmented table.

An index must have at least 2000 keys to be compressed.

You cannot compress the following types of indexes:

- An index that is not a B-tree index
- An attached B-tree index
- Virtual B-tree indexes
- An index that does not have at least 2000 keys

The compression operation compresses only the leaves (bottom level) of the index.

You cannot uncompress a compressed index. If you no longer need the compressed index, you can drop the index and then re-create it as an uncompressed index.

You can compress a new index when you create it by including the COMPRESSED option in the CREATE INDEX statement. You compress an existing index with an SQL administration API command.

Compression ratio estimates

The compression ratio depends on the data that is being compressed. Before you compress a table or table fragment, you can estimate the amount of space you can save if data is compressed. Compression estimates are based on samples of row data. The actual ratio of saved space might vary.

The compression algorithm that HCL OneDB™ 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 compression ratio is 90 percent. The maximum 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 achieved because HCL OneDB™ adds a single byte of metadata to each compressed image.

HCL OneDB™ estimates the compression ratios by random sampling of row data and then summing up the sizes of the following items:

- Uncompressed row images
- Compressed row images, based on a new compression dictionary that is temporarily created by the estimate compression command
- Compressed row images, based on the existing dictionary, if there is one. If there is no existing dictionary, this value is the same as the sum of the sizes of the uncompressed row images.

The actual space saving ratios that are achieved might vary from the compression estimates due to a sampling error, the type of data, how data fits in data pages, or whether other storage optimization operations are also run.

Some types of data compress more than other types of data:

- 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 is already compressed by another algorithm and data that is encrypted might not compress well. For example, images and sound samples in rows might already be compressed, so compressing the data again does not save more space.

Compression estimates are based on raw compressibility of the rows. The server generally puts a row onto a single data page. How the rows fit on data pages can affect how much the actual compression ratio varies from the estimated compression ratio:

- When each uncompressed row nearly fills a page and the compression ratio is less than 50 percent, each compressed row fills more than half a page. The server puts each compressed row on a separate page. In this case, although the estimated compression ratio might be 45 percent, the actual space savings is nothing.
- When each uncompressed row fills slightly more than half a page and the compression ratio is low, each compressed row might be small enough to fit in half a page. The server puts two compressed rows on a page. In this case, even though the estimated compression ratio might be as low as 5 percent, the actual space savings is 50 percent.

HCL OneDB™ does not 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 extra 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.

Compression dictionaries

A compression dictionary is a library of frequently occurring patterns in data or index keys and the symbol numbers that replace the patterns.

One compression dictionary exists for each compressed fragment and each compressed non-fragmented table.One compression dictionary exists for each compressed fragment, each compressed non-fragmented table, each compressed simple large object in a dbspace, and each compressed index partition.

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.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, or an index that has at least 2,000 keys. Typically, approximately 100 KB of space is required for storing the 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).

HCL OneDB™ attempts to capture the best compressible patterns (the frequency of the pattern that is 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.

Tools for moving compressed data

You can use HCL OneDB™ data migration utilities to move compressed data between databases.

The dbexport utility uncompresses compressed data. Therefore, you must recompress the data after you use the dbimport utility to import the data.

BLOBspace Blob Compression

A *partition BLOB* (binary large object) column is a TEXT or BYTE column that is stored in a DBspace, in the same RSAM partition as the home row. A *BLOBspace BLOB* column is the same data type (TEXT or BYTE), but the BLOB data is stored in a BLOBspace.

All TEXT and BYTE BLOBs may be compressed, whether they are stored in the partition or a BLOBspace. If a table containing TEXT or BYTES columns is created with the COMPRESSED keyword, the BLOB data will be automatically compressed along with the home row data once the number of rows has reached a certain threshold (2000 by default). Separate compression dictionaries are built for the home rows and each BLOB column.

- By using the "compress" sysadmin task() command rather than the auto compress feature, it is possible to compress only the BLOB data in a table, or to compress only the home row data without compressing BLOBs.
- ER is able to replicate BLOB data whether it is compressed or uncompressed.

Methods for viewing compression information

You can display compression statistics, information about compression dictionaries, and the compression dictionary.

The following table describes the different methods that you can use to view compression information.

Table 31. Methods to view compression information

Method	Description
oncheck -pT or oncheck -pt command	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 about the progress of currently running compression operations.
onstat -g ppd option	Displays information about 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.
`UNLOAD TO` `'compression_dictionary_file'` `SELECT * FROM`	View the compression dictionary in a file.

Table 31. Methods to view compression information (continued)

Method	Description
`sysmaster:syscompdicts_full;` SQL statement	

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 *HCL OneDB™ Guide to SQL: Syntax*
dbload utility	Yes	Yes	*HCL OneDB™ Migration Guide*
dbimport utility	Yes	Yes	*HCL OneDB™ Migration Guide*
programs	Yes	Yes	*HCL OneDB™ ESQL/C Programmer's Manual*
Insert MERGE, using an EXTERNAL source table	Yes	Yes	*HCL OneDB™ Guide to SQL: Syntax*

Important: The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been loaded into a database.

Moving data with external tables

You can use external tables to load and unload database data.

About this task

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 HCL® OneDB® internal data format
- Use the database server to convert data between delimited ASCII, fixed-ASCII, and HCL® OneDB® 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 the HCL® OneDB® 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 HCL® OneDB® 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.

About this task

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
   ```

Results

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 HCL® OneDB® 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 HCL® OneDB® 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.

About this task

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;
   ```

Results

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.

About this task

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;
```

Results

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 HCL OneDB™ instance to another without writing the data to an intermediate file.

Before you begin

You can use a named pipe to unload data from one HCL OneDB™ 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 HCL OneDB™ 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
```

About this task

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
```

For more information, see *HCL OneDB™ 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.

HCL OneDB™ 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 32. System catalog tables that describe external table files

Table name	Description
sysexternal	Stores name and attributes of each external table file
sysextdfiles	Stores file-path and directory of each external table file
sysextcols	Stores attributes of each column in FIXED-type external tables

See the *HCL OneDB™ 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, HCL OneDB™ is unable to determine the number of rows in the external table because the data exists outside the database in data files. HCL OneDB™ 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
- External tables

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 HCL® OneDB® 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 *HCL OneDB™ Error Messages* or use the finderr utility. For information about the violations table error messages, see your *HCL OneDB™ 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 logging type of the table is RAW, the database server can use light append (or EXPRESS) mode to load data and to process check constraints. However, if the database server crashes while inserting data rows into a RAW table in EXPRESS mode, this unlogged light append operation is not rolled back, and the table might be left in an unknown state.
- 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 tables of RAW or STANDARD logging types, see the *HCL OneDB™ Backup and Restore Guide*.

Logging and log administration

Logging

These topics describe logging of HCL® OneDB® 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 295.

Database server processes that require logging

As HCL OneDB™ operates, processing transactions, tracking data storage, and ensuring data consistency, HCL OneDB™ 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 Logical log on page 300 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 *HCL OneDB™ 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 288.

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 335.)

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 *HCL OneDB™ 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 *HCL OneDB™ Database Design and Implementation Guide*. For information about primary key and foreign key constraints, see the *HCL OneDB™ 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 566.

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 390.

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 *HCL OneDB™ 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 293 and Manage the database-logging mode on page 295.

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 on page 288 and Database logging in an X/Open DTP environment on page 295.

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 HCL OneDB on page 179.

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.

ALTER ACCESS_METHOD
ALTER FRAGMENT
ALTER FUNCTION
ALTER INDEX
ALTER PROCEDURE
ALTER ROUTINE
ALTER SECURITY LABEL COMPONENT
ALTER SEQUENCE
ALTER TABLE
ALTER TRUSTED CONTEXT
ALTER USER
CLOSE DATABASE
CREATE ACCESS_METHOD
CREATE AGGREGATE
CREATE CAST
CREATE DATABASE
CREATE DISTINCT TYPE
CREATE EXTERNAL TABLE
CREATE FUNCTION
CREATE FUNCTION FROM
CREATE INDEX
CREATE OPAQUE TYPE
CREATE OPCLASS
CREATE PROCEDURE
CREATE PROCEDURE FROM
CREATE ROLE
CREATE ROUTINE FROM
CREATE ROW TYPE
CREATE SCHEMA
CREATE SECURITY LABEL
CREATE SECURITY LABEL COMPONENT
CREATE SECURITY POLICY
CREATE SEQUENCE
CREATE SYNONYM

CREATE TABLE
CREATE TEMP TABLE
CREATE TRIGGER
CREATE TRUSTED CONTEXT
CREATE USER
CREATE VIEW
CREATE XADATASOURCE
CREATE XADATASOURCE TYPE
DROP ACCESS_METHOD
DROP AGGREGATE
DROP CAST
DROP DATABASE
DROP FUNCTION
DROP INDEX
DROP OPCLASS
DROP PROCEDURE
DROP ROLE
DROP ROUTINE
DROP ROW TYPE
DROP SECURITY
DROP SEQUENCE
DROP SYNONYM
DROP TABLE
DROP TRIGGER
DROP TRUSTED CONTEXT
DROP TYPE
DROP USER
DROP VIEW
DROP XADATASOURCE
DROP XADATASOURCE TYPE
GRANT
GRANT FRAGMENT
RENAME COLUMN
RENAME DATABASE
RENAME INDEX
RENAME SECURITY
RENAME SEQUENCE
RENAME TABLE
RENAME TRUSTED CONTEXT
RENAME USER
REVOKE
REVOKE FRAGMENT

TRUNCATE
UPDATE STATISTICS
SAVE EXTERNAL DIRECTIVES
SET CONSTRAINTS
SET Database Object Mode
SET INDEXES
SET TRIGGERS
START VIOLATIONS TABLE
STOP VIOLATIONS

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.

DELETE
FLUSH
INSERT
LOAD
MERGE
PUT
SELECT INTO TEMP
UNLOAD
UPDATE

The following SQL statements generate logs in special situations.

Table 33. 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.
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

Some SQL statements are not logged.

The following SQL statements do not produce log records, regardless of the database logging mode.

- ALLOCATE COLLECTION
- ALLOCATE DESCRIPTOR
- ALLOCATE ROW
- CLOSE
- CONNECT
- DATABASE
- DEALLOCATE COLLECTION
- DEALLOCATE DESCRIPTOR
- DEALLOCATE ROW
- DECLARE
- DESCRIBE
- DISCONNECT
- FETCH
- FREE
- GET DESCRIPTOR
- GET DIAGNOSTICS
- INFO
- LOCK TABLE
- OPEN
- OUTPUT
- PREPARE
- RELEASE SAVEPOINT
- SAVEPOINT
- SELECT
- SET AUTOFREE
- SET COLLATION
- SET CONNECTION
- SET DATASKIP
- SET DEBUG FILE
- SET DEFERRED_PREPARE
- SET DESCRIPTOR
- SET ENCRYPTION PASSWORD
- SET ISOLATION
- SET LOCK MODE
- SET LOG
- SET OPTIMIZATION
- SET PDQPRIORITY

SET ROLE

SET SESSION AUTHORIZATION

SET STATEMENT CACHE

SET TRANSACTION

SET Transaction Mode

SET USER PASSWORD

UNLOCK TABLE

WHENEVER

SET ENVIRONMENT

SET EXPLAIN

For temporary tables in temporary dbspaces, nothing is logged, not even the SQL statements that are always logged for other types of tables. If you include temporary (nonlogging) dbspaces in the value of the DBSPACETEMP configuration parameter, the database server places nonlogging tables in these temporary dbspaces first.

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 286.

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 299. 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 18: How the database server uses shared memory on page 109 and Flush the logical-log buffer on page 138.

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 288. 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 549.

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 *HCL OneDB™ 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. Manage the database-logging mode on page 295, 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 *HCL OneDB™ Guide to SQL: Syntax*.

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 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 Logging on page 286. To find out the current logging mode of a database, see Monitor the logging mode of a database on page 299.

For information about using SQL administration API commands instead of some ondblog, see Remote administration with the SQL administration API on page 599 and the *HCL OneDB™ Administrator's Reference*.

Change the database-logging mode

You can use ondblog to add or change logging. Then use ON-Bar to back up the data. When you use ON-Bar, the database server must be in online, administration, or quiescent mode.

You can use ondblog HCL® OneDB® Server Administrator (ISA) to add or change logging. Then use ON-Bar to back up the data. When you use ON-Bar, the database server must be in online, administration, or quiescent mode.

For information about ON-Bar, see the *HCL OneDB™ 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 34. 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 299. 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 299. 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 *HCL OneDB™ 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 *HCL OneDB™ 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

About this task

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 *HCL OneDB™ 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 database logging mode with ISA

HCL® OneDB® 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 297.

Modify database logging mode with ON-Monitor (UNIX™)

You can use ON-Monitor to change the log-buffering mode between unbuffered and buffered.

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 *HCL OneDB™ Guide to SQL: Syntax*. For more information, see Table types for HCL OneDB on page 179.

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 HCL OneDB™ 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 or 2.

You can use the onmode -wf command to change the value of TEMPTAB_NOLOG.

Related information

TEMPTAB_NOLOG configuration parameter on page

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 564
onstat -g sql	Monitor SQL statements, listed by session ID and database.	Performance monitoring in the *HCL OneDB™ Performance Guide*
onstat -g spf	Display detailed information about SQL queries.	
onstat -g stm	Monitor memory usage of prepared SQL statements.	Memory utilization in the *HCL OneDB™ 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 *HCL OneDB™ 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 on page 299.

Monitor the logging mode with ISA

To use HCL® OneDB® Server Administrator (ISA) to display the logging status and buffering mode for your databases, select **SQL > Schema**.

Logical log

The information in Logging on page 286, and these topics explains how the database server uses the logical log. For information about how to perform logical-log tasks, see Manage logical-log files on page 309, and Manage the database-logging mode on page 295.

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 on page 301.
- Monitoring the logical-log file status

  See Identification of logical-log files on page 301.
- Allocating an appropriate amount of disk space for the logical log

  See Size of the logical-log file on page 302.
- Allocating additional log files whenever necessary

  See Allocate logical log files on page 318.
- Backing up the logical-log files to media

See Back up logical-log files on page 312 and Freeing of logical-log files on page 304.

- Managing logging of blobspaces and sbspaces

  See Log blobspaces and simple large objects on page 305 and Log sbspaces and smart large objects on page 306.

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.

To improve performance by reducing 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.

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 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 35. 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

Table 35. Logical-log file-numbering sequence (continued)

Log file number	First rotation unique ID number	Second rotation unique ID number	Third rotation unique ID number
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 36. 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.
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 316.

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 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 306.

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 *HCL OneDB™ 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 322.
- 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 logical log files on page 318.

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 340.
- The file contains an open transaction.

  The open transaction is the long transaction explained in Controlling long transactions on page 326.
- The file is not backed up.

  If the logical-log file is not backed up, processing resumes when you use ON-Bar 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 *HCL OneDB™ 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 314.

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 166, 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 312.

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 313 and the *HCL OneDB™ 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 220 onspaces -ch: Change sbspace default specifications on page
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) on page
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 171 PUT Clause on page
mi_lo_create DataBlade® API function	Turns off logging for a smart large object when it is initially loaded	*HCL OneDB™ DataBlade® API Function Reference*
mi_lo_alter DataBlade® API function	Turns on logging after the load is complete	*HCL OneDB™ DataBlade® API Function Reference*
ifx_lo_create function	Turns off logging for a smart large object when it is initially loaded	*HCL OneDB™ ESQL/C Programmer's Manual*
ifx_lo_alter function	Turns on logging after the load is complete	*HCL OneDB™ 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 *HCL OneDB™ 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.

Manage logical-log files

You must manage logical-log files even if none of your databases uses transaction logging. See Logical log on page 300 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 on page 310.
- If you do not want to use the default values, change the LOGSIZE and LOGBUFF configuration parameters.
- Add the estimated number of logical-log files.

  See Allocate logical log files on page 318.

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 Remote administration with the SQL administration API on page 599 and the *HCL OneDB™ 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 High-availability cluster configuration on page 372 or in the *HCL OneDB™ 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 RTO 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 RTO 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 RTO 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 The Scheduler on page 577.

If an RTO 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 322.

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 316.

For information about LOGSIZE, LOGFILES, and NETTYPE, see the topics about configuration parameters in the *HCL OneDB™ 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 *HCL OneDB™ Backup and Restore Guide*.

Backing up blobspaces

About this task

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.

Results

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.

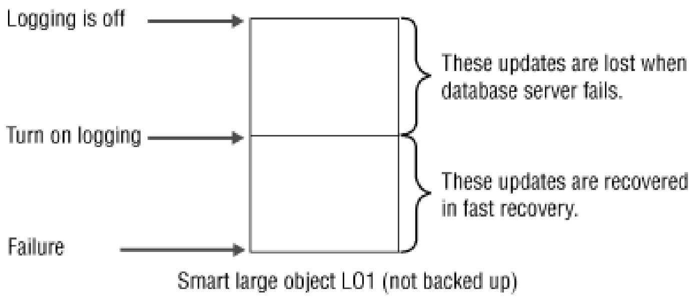
Back up sbspaces

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 334.

Figure 47. Turn on logging in an sbspace


Logging is off
Turn on logging
Failure
These updates are lost when database server fails.
These updates are recovered in fast recovery.
Smart large object LO1 (not backed up)

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 302 and Monitor logging activity on page 316.

Tip: For information using ON-Bar or ontape to back up storage spaces and logical logs, see the *HCL OneDB™ 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 on page 315 or Free a log file with status U-B-L on page 316.

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 Changing database server operating modes on page 72. 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.

Results

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

About this task

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.
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.

Results

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 315 or Free a log file with status U-B-L on page 316.

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 337.

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 *HCL OneDB™ 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 *HCL OneDB™ 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 logical log files

When you initialize or restart the database server, it creates the number of logical-log files that are specified by the LOGFILES configuration parameter. The size of the logical log files is specified by the LOGSIZE configuration parameter.

You can manually add logical log files or configure the database server to add logical log files as needed. The database server updates the value of the LOGFILES configuration parameter dynamically when logical log files are added.

The following configuration parameters also affect logical log files. You can update the value of these configuration parameters while the server is running, unless otherwise noted.

AUTO_LLOG

Automatically adds logical logs to improve performance and limits the total size of logical log files.

DYNAMIC_LOGS

Automatically adds logical logs to prevent transaction blocking.

LOGBUFF

Sets the size of the three logical-log buffers in shared memory. You must restart the database server when you change the value of the LOGBUFF configuration parameter.

LTXEHWM

Sets the percentage of available log space that, when filled, triggers the database server to give the long transaction currently that is being rolled back exclusive access to the logical log.

LTXHWM

Sets the percentage of available log space that, when filled, triggers the database server to check for a long transaction.

Dynamically add a logical-log file to prevent transaction blocking

The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file to prevent transaction blocking.

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.

Size and number of dynamically added log files

The purpose of enabling dynamic logs with the DYNAMIC_LOGS configuration parameter 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.

Location of dynamically added logical log files

If the DYNAMIC_LOGS configuration parameter is set to 2, the default location of dynamically added log files is the dbspace that contains the newest log file.

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 320.

Monitor events for dynamically added logs

You can monitor the event alarms that are triggered when the database server dynamically adds logical log files to prevent transaction blocking. The DYNAMIC_LOGS configuration parameter value must be 1 or 2.

When each alarm is triggered, a message is written to the message log.

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 37. Event alarms for dynamically added log files

Class ID	Severity	Class message	Message
26	3	Dynamically added log file *log_number*	This message shows 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 shows 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 38. The DYNAMIC_LOGS settings

DYNAMIC_ LOGS value	Meaning	Event alarm	Wait to add log?	Dynamic log added?
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	No	No	No

Table 38. The DYNAMIC_LOGS settings (continued)

DYNAMIC_ LOGS value	Meaning	Event alarm	Wait to add log?	Dynamic log added?
	`transaction_address`, but the dynamic log feature is turned off.			

Dynamically add logical logs for performance

You can set the AUTO_LLOG configuration parameter to enable the database server to dynamically add logical logs to improve performance.

When you set the AUTO_LLOG configuration parameter, you also specify a dbspace in which to create new logical log files and the size of all logical log files at which the server stops adding logs for performance.

The AUTO_LLOG and DYNAMIC_LOGS configuration parameters add logical logs under different conditions that do not directly interact. When the AUTO_LLOG configuration parameter is enabled, logical logs are added to improve performance. When the DYNAMIC_LOGS configuration parameter is enabled, logical logs are added under more urgent conditions, such as when a long transaction threatens to block the server by using all available log space. The settings of the two configuration parameters do not constrain each other. For example, the maximum size that is specified in the AUTO_LLOG configuration parameter does not affect the amount of log space that can be added by the DYNAMIC_LOGS configuration parameter. Similarly, the value of AUTO_LLOG configuration parameter does not affect the amount of log space that you can add manually.

Adding logical-log files manually

You can use an onparams command or ON-Monitor to add logical-log files.

About this task

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.

   Choose from:
   - 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 *HCL OneDB™ Backup and Restore Guide*.

Results

For more information about using onparams to add a logical-log file, see the *HCL OneDB™ 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 322, 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.

About this task

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.

Results

For information about using onparams to drop a logical-log file, see the *HCL OneDB™ 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 326.

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 322.
- Increase the LOGSIZE value in the `onconfig` file if you want the database server to create larger log files.

Move logical-log files

You might want to move logical-log files for performance reasons or to make more space in the dbspace.

To find the location of logical-log files, run the onstat -l command. 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:

- Optionally dropping logical-log files from their current dbspace.
- Adding the logical-log files to their new dbspace

Restriction: You cannot move logical log files into dbspaces that have non-default page sizes.

The database server must be in online, administration, quiescent, or fast-recovery mode.

You can change the location for new logical logs by setting the AUTO_LLOG configuration parameter to 1 and the name of the dbspace. The AUTO_LLOG configuration parameter enables the database server to add logical logs as needed to improve performance.

Example

The following procedure provides an example of how to move six logical-log files from the root dbspace to another dbspace, **dbspace_1**:

1. Add six new logical-log files to **dbspace_1** by running the following command:
   ```
   onparams -a -d dbspace_1
   ```
2. Take a level-0 backup of all storage spaces to free all log files except the current log file. For example, you can run the following command to back up all log files, including the current log file:
   ```
   onbar -l -b -c
   ```
3. Run the onmode -l command to switch to a new current log file.
4. Drop all six logical-log files in the root dbspace by running the onparams -d -l command with the log file number for each log file. You cannot drop the current logical-log file.
5. Create a level-0 backup of the root dbspace and **dbspace_1**. For example, you can run the following command:
   ```
   onbar -b root dbspace_1
   ```

Display logical-log records

Use the onlog utility to display and interpret logical-log records. For information about using onlog, see the *HCL OneDB™ Administrator's Reference*.

Controlling long transactions

There are multiple ways to control long transactions:

- Adjust the long-transaction high-watermark setting
- Adjust the exclusive access, long-transaction high-watermark setting
- Adjust the size of log files
- Set limits on logspace available to individual transactions
- Set limits on the amount of time a transaction can run

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.

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 10 logical logs and LTXHWM is set to 98. A transaction begins in log file 1 and update activity fills logs 1 - 9. The database server dynamically adds log file 11 after log file 10. If the transaction does not complete, this process continues until the database server adds 40 log files. When the database server adds the 50th log, the transaction is 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 by the database server. 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 have enough disk space, and you never want to block other users while a long transaction is rolling back.

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 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.

Set limits on logspace available to transactions

The SESSION_LIMIT_LOGSPACE configuration parameter limits how much log space a session can use for each transaction, which prevents transactions above a specific size from occurring, and prevents large transactions from a single session from monopolizing system resources.

The database server terminates a transaction that exceeds the log space limit, and produces an error in the database server message log.

The size limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Set limits on the amount of time a transaction can run

The SESSION_LIMIT_TXN_TIME configuration parameter limits how much time a transaction can run in a session, which prevents transactions that require a large amount of time from occurring, and prevents long transactions from a single session from monopolizing system resources.

The database server terminates a transaction that exceeds the time limit, and produces an error in the database server message log.

The time limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

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 Manage the physical log on page 336.

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 332.

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 334 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 *HCL OneDB™ 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 305.

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 166.

Size and location of the physical log

When the database server initializes disk space, it places the physical log in the root dbspace. The initial size of the physical log is set by the PHYSFILE configuration parameter.

After you initialize the database server for the first time, you can change the size or location of the physical log with the onparams utility.

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. For best performance, create the plogspace to store the physical log and allow the database server to expand the size of the physical log as needed to improve performance.

Recommendation: Locate critical dbspaces on fault-tolerant storage devices. If the storage that the physical log is in is not fault-tolerant, use HCL OneDB™ mirroring for the dbspace that contain the physical log. This protects the database if the storage device fails. However, if you mirror the plogspace, it cannot be expanded.

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 332.

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 330.

If failover occurs, and the secondary server becomes the primary server, checkpoint discrepancies between the two servers can affect re-connection attempts. If a checkpoint on the new secondary server does not exist on the new primary server, attempts to connect the secondary server to the primary server fail. The secondary server must be fully restored before it can connect to the primary server.

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.)

Tip:

- If automatic checkpoints are triggered too frequently because of physical log activity, you can increase the physical log size or use a plogspace to automatically tune the physical log resources.
- If automatic checkpoints are triggered too frequently because of logical log activity, you can set the AUTO_LLOG parameter in the `onconfig` file to allow the server to automatically increase the logical log space to reduce checkpoint frequency.

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 *HCL OneDB™ 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 *HCL OneDB™ 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.

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 335.

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 294), 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 `$ONEDB_HOME/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

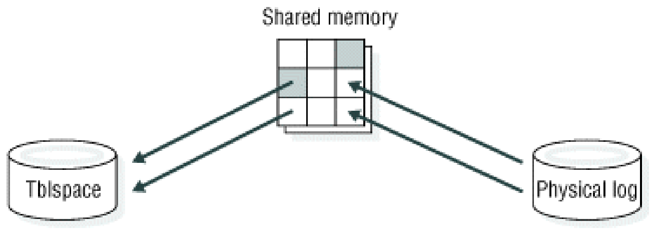
Fast recovery returns the database server to a consistent state as part of shared-memory initialization. All committed transactions are restored, and all uncommitted transactions are rolled back.

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 48. Writing all remaining before-images in the physical log back to disk



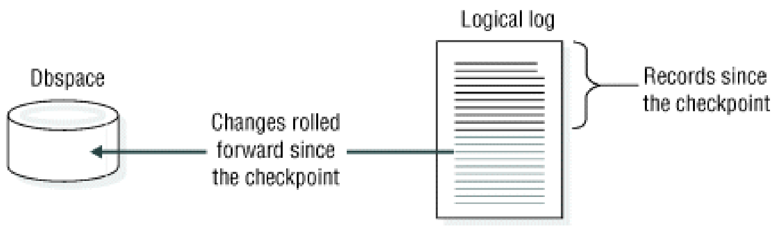
The server locates the checkpoint record in the logical log

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 49. Rolling forward the logical-log records written since the most recent checkpoint


Logical log
Dbspace
Records since the checkpoint
Changes rolled forward since the checkpoint

The server rolls back uncommitted transactions

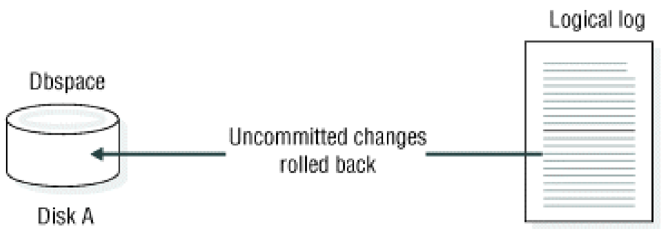
After rolling the logical-log records forward, the database server rolls back all logical-log records for transactions that were not committed at the time the system failed. All databases are logically consistent because all committed transactions are rolled forward and all uncommitted transactions are 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 556.

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 50. Rolling back all incomplete transactions



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 Physical logging, checkpoints, and fast recovery on page 328 for background information.

Change the physical-log location and size

You can 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 that are allocated for the physical log in the root dbspace. You might improve performance by moving the physical log to another dbspace.

You can move the physical log to a dbspace or the plogspace. When the physical log is in the plogspace, the database server increases the size of the physical log as needed to improve performance. When the physical log is in a dbspace, you must manually increase the size of the physical log.

To move the physical log to the plogspace, create the plogspace by running the onspaces -c -P command or the SQL administration API admin() or task() function with the create plogspace argument. To change the location of the plogspace, create a new plogspace. The physical log is moved to the new plogspace and the old plogspace is dropped.

Prerequisites to moving the physical log to a dbspace:

- Log in as user **informix** or **root** on UNIX™ or as a member of the **Informix-Admin** group on Windows™.
- Determine whether adequate contiguous space in the target chunk is available by running the oncheck -pe command.

  The space that is allocated for the physical log must be contiguous. When you change 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. The dbspace must use the default page size.

To move the physical log to a dbspace, run the onparams -p -s command or the SQL administration API admin() or task() function with the create dbspace argument.

The following example changes the size and location of the physical log. The new physical-log size is 400 KB, and the log is in the **dbspace6** dbspace:

```
onparams -p -s 400 -d dbspace6
```

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 *HCL OneDB™ 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**)
Command line Command line or ISA	onparams -p	Moves or resizes the physical log.
Command line	onmode -l	Advances to the next logical-log file.

Utility	**Command**	**Additional information**
Command line or ISA		
ISA	**Logs > Logical**	Click **Advance Log File**.

For more information about and an example of onstat -l output, see the *HCL OneDB™ Administrator's Reference*.

For information about using SQL administration API commands instead of some onparams and onmode commands, see Remote administration with the SQL administration API on page 599 and the *HCL OneDB™ Guide to SQL: Syntax*.

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 *HCL OneDB™ 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 Remote administration with the SQL administration API on page 599 and the *HCL OneDB™ 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.

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 *HCL OneDB™ Performance Guide*.

Fault tolerance

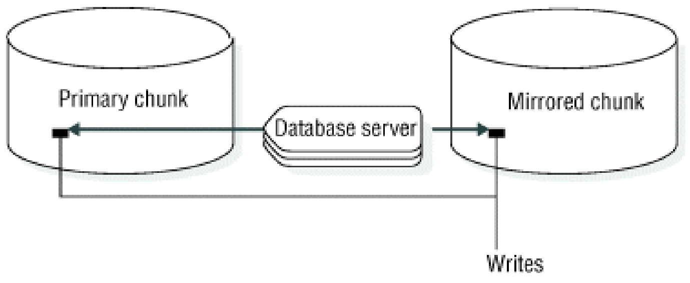
Mirroring

These topics describe the database server mirroring feature. For instructions on how to perform mirroring tasks, see Using mirroring on page 347.

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 51. 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 345.

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 345.) In addition, database server mirroring uses split reads to improve read performance. (See Disk reads from mirror chunks on page 345.) 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 *HCL OneDB™ 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 Using mirroring on page 347.

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 Using mirroring on page 347.

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 *HCL OneDB™ Administrator's Reference*. For information about how to display these status flags, see Monitor disk usage on page 240.

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 353.

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.

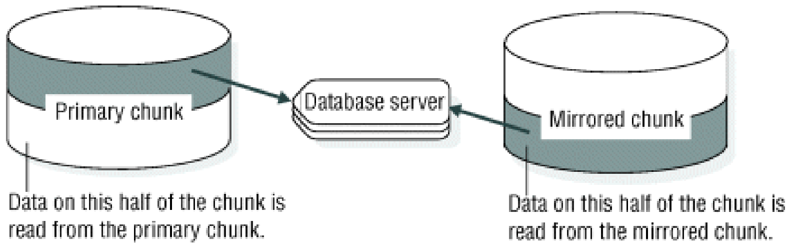
Disk writes to mirror chunks

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 52. Split read reducing the maximum distance over which the disk head must travel


Primary chunk
Database server
Mirrored chunk
Data on this half of the chunk is read from the primary chunk.
Data on this half of the chunk is read from the mirrored chunk.

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 345.

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 *HCL OneDB™ Administrator's Reference*.

A display of disk-space use, provided by one of the methods explained under Monitor chunks on page 240, 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.

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

About this task

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 on page 347.
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 348.
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 349.

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 349.

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 197.

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 344.

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 240.

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 on page in the *HCL OneDB™ 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.

About this task

Start mirroring for unmirrored dbspaces using onspaces

About this task

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

About this task

To start mirroring with HCL® OneDB® 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.

About this task

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

About this task

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

About this task

To start mirroring for new storage spaces with HCL® OneDB® 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.

About this task

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

About this task

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

About this task

To add mirror chunks with HCL® OneDB® 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.

About this task

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.

Swap mirror chunk

You can use the swap_mirror command to swap a mirror chunk and a primary chunk, making the original primary chunk the mirror, and the original mirror chunk the new primary. This operation is especially useful as an on-line method of migrating chunks to a newer, faster set of disk drives.

To swap a single mirrored chunk, use the following command:

execute function task("modify chunk swap_mirror",<chunk number>).

To swap all the chunks in a mirrored space, use the following command:

execute function task("modify space swap_mirrors","<space name>").

Note: You cannot swap a chunk if either its primary or mirror is down.

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 348.)

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

About this task

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.

About this task

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

About this task

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

About this task

To recover a mirror chunk with HCL® OneDB® 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.

About this task

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

About this task

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.

About this task

Select a dbspace or blobspace that is mirrored and press **CTRL-B** or **F3**.

Ending mirroring using ISA

About this task

To end mirroring with HCL® OneDB® Server Administrator (ISA):

1. Select **Storage > Chunks**.
2. Select the dbspace name and click **Stop Mirroring**.

Consistency checking

HCL® OneDB® 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 HCL® OneDB® 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 39. 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 *HCL OneDB™ 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 HCL® OneDB® 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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*.

Retain consistent level-0 backups

After you perform the checks described in Verify consistency on page 355 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 355.
- 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 *HCL OneDB™ 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 353.

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 HCL® OneDB® Technical Support whenever one occurs. The diagnostic information collected is intended for the use of HCL® OneDB® 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 HCL® OneDB® 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 *HCL OneDB™ 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

HCL® OneDB® 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.

ONDBSPACEDOWN setting	Result	Action
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 *HCL OneDB™ 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 *HCL OneDB™ Administrator's Reference*.

No bad-sector mapping

HCL OneDB™ 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.

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 database servers, 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.

Strategies for high availability and scalability

HCL OneDB™ 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 HCL OneDB™ technologies and components to support those strategies.

Components supporting high availability and scalability

HCL OneDB™ database software can be customized to create systems that provide uninterrupted services, minimize maintenance and downtime, automatically redirect client connection requests to the most appropriate database servers, and distribute both processing and storage across hardware.

High-availability clusters

A high-availability cluster consists of a primary server that contains the master copy of data, and is securely networked to at least one secondary server that is synchronized with the primary server or has access to the primary server's data. Transactions are sent to secondary servers after they are committed on the primary server, so database data is reliable. If the primary server fails, a secondary server can become the primary server, and client connections can be redirected to the new primary server.

High-availability cluster servers are configured with identical hardware and software. High-availability cluster servers can be in close proximity or geographically remote from each other, and applications can securely connect to any of the cluster servers.

There are three types of secondary servers:

- Shared-disk (SD) secondary servers, which share disk space with the primary server.
- High-availability data replication (HDR) secondary servers, which maintain synchronously or asynchronously updated copies of the entire primary server and can be accessed quickly if the primary server fails.
- Remote stand-alone (RS) secondary servers, which maintain asynchronously updated copies of the entire primary server, and can serve as remote-backup servers in disaster-recovery scenarios.

Enterprise Replication

Using Enterprise Replication, you can maintain complete or partial copies of your data across multiple servers by replicating transactions. Data is replicated asynchronously through transactions captured from the logical log. If a remote database server or network failure occurs, a local database server can service local users, and store transactions to be replicated until remote servers become available. At each database server, Enterprise Replication reads the logical logs to capture locally originating transactions, and then replicates those transactions to other database servers in the Enterprise Replication domain.

Shard clusters

HCL OneDB™ can horizontally partition (*shard*) a table or collection across multiple database servers. The set of database servers that data is sharded across is called a *shard cluster*, and each of the database servers in the set is a *shard server*. Distributing rows or documents across a shard cluster reduces the size of the associated index on each shard server, and distributes performance across your hardware. As your database grows in size, you can scale up by adding more database servers. Horizontal partitioning is also known as sharding.

Rows or documents that are inserted on one shard server can be replicated or sent to other shard servers, depending on the sharding rules you specify. Queries that are performed on a shard server can select data from other shard servers in a shard cluster. When data is sharded based on a replication key that specifies certain segmentation characteristics, queries can skip shard servers that do not contain relevant data. This query optimization is another benefit that comes from data sharding.

Connection management

The Connection Manager is a utility that can monitor the workload and status of database servers in high-availability clusters, Enterprise Replication domains, grids, and server sets, and then use a redirection policy to send client connection requests to the most appropriate database server. Connection Managers can also act as proxy servers to handle client/server communication and circumvent connection issues that are related to firewalls.

Connection Managers can control failover for high-availability clusters, automatically promoting secondary servers to the role of the primary server if the original primary server fails.

If a partial network failure occurs, Connection Managers can prioritize connections between application servers and the primary server of a high-availability cluster, to better define failover.

Grids

A grid is a set of interconnected replication servers, where SQL commands can be propagated from one server to all the others. Grids provide an easier way to administer a large group of servers, update database schemas, run stored procedures and user-defined routines, and administer replication.

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.

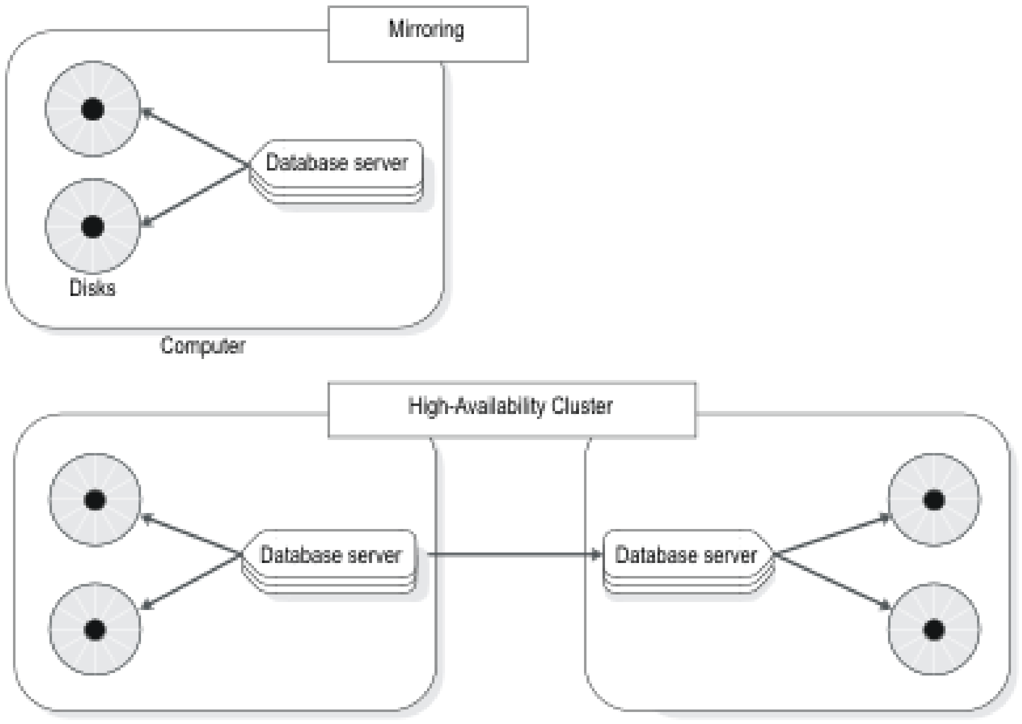
Clustering versus mirroring

Clustering and mirroring are transparent methods for increasing fault tolerant.

Mirroring, described Mirroring on page 342, 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 53. A comparison of mirroring and clustering



Clustering versus two-phase commit

The two-phase commit protocol, described in detail in Multiphase commit protocols on page 549, 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 *HCL OneDB™ Enterprise Replication Guide* and Enterprise Replication as part of the recoverable group on page 402.

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.

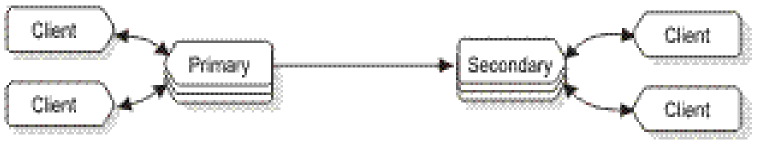
User-defined types, user-defined routines, and DataBlade® modules have special installation and registration requirements. For instructions, see How data initially replicates on page 391.

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.

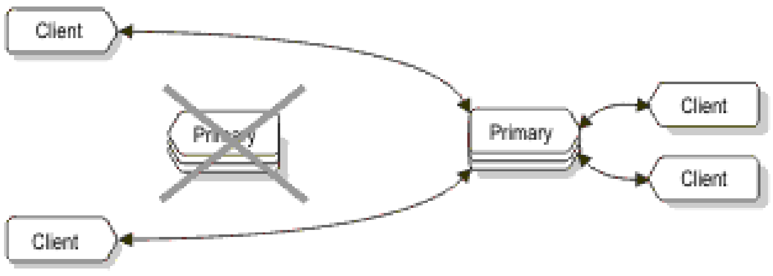
As the following figure illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.

Figure 54. 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 55. Database servers and clients in a data replication configuration after a failure



Transparent scaling and workload balancing strategies

HCL OneDB™ servers scale easily and they dynamically balance workloads to ensure optimal use of resources.

HCL OneDB™ can address these objectives:

- Periodically increase capacity on page 368
- Geographically disperse processing and increase capacity on page 369
- Balance workload to optimize resource use on page 370

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 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 40. Scalability with shared-disk secondary servers

Advantages	Potential disadvantages
• 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

  If 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 41. Scalability with remote stand-alone secondary servers

Advantages	Potential disadvantages
• Data redundancy. This secondary server maintains a copy of the data. • 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.	• The primary and secondary servers require the same hardware, operating system, and version of the database server product.

Geographically disperse processing and increase 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. Remote database server outages are tolerated. If database server or network failure occurs, 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 local 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.

You can also use Enterprise Replication to set up a shard cluster, where your data is horizontally partitioned (*sharded*) across multiple servers. As your capacity requirements grow, you can add additional database servers to the shard cluster, increasing your overall capacity.

Table 42. 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. • You can add additional shard servers to an established shard cluster as your capacity needs increase.	• Conflicts are possible. • Transaction failures are possible. If one occurs, 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. HCL OneDB™ 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

HCL OneDB™ 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	• Very high availability. Applications can access this server quickly if they cannot connect to a primary server.	• Local to the primary • Requires an exact replica of the data (including table and database schemas).

Goal	Strategy	Advantages	Potential disadvantages
	availability data replication server) • (Can also use RSS and ER)	• Data is replicated synchronously. • Increased scalability • No requirement to change applications	• 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

High-availability cluster configuration

These topics describe how to plan, configure, start, and monitor high-availability clusters for HCL OneDB™, 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 HCL® OneDB® Enterprise Replication, see the *HCL OneDB™ Enterprise Replication Guide*.

High availability and scalability on page 363, explains what a high-availability cluster is, how it works, and how to design client applications for a cluster environment.

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 Manage the database-logging mode on page 295.
- 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 534 and Design data replication group clients on page 410.
- 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 534.

Configuring clusters

Configure clusters by securing confirming the hardware, operating-system, and database requirements. You also set up a security protocol and the secure connection.

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.

The Connection Manager also supports Distributed Relational Database Architecture™ (DRDA®) connections. For more information, see Distributed Relational Database Architecture (DRDA) communications on page 57.

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 HCL OneDB™ 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 HCL OneDB™ executable image. In this situation, you cannot add a server on a Linux™ 64-bit operating system because that operating system requires a different HCL OneDB™ 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.
- 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.

  You can encrypt storage spaces on high-availability servers. The encryption state of corresponding storage spaces on HDR and RS primary and secondary servers can be different. The encryption state of corresponding storage spaces on SD primary and secondary servers must be the same.
- The secondary servers must not use disk compression.

  If you use the HCL OneDB™ 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 Overview of database server configuration and administration on page 3.
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 *HCL OneDB™ 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 410.

UNIX™ Only:

You must use symbolic links for the chunk path names, as explained in Allocating raw disk space on UNIX on page 199.

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 412.

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 *HCL OneDB™ Backup and Restore Guide*.

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.

High-availability cluster configuration parameters

Set the HA_ALIAS configuration parameter on each database server that participates in a high-availability cluster.

Set TEMPTAB_NOLOG configuration parameter to 1 or 2 on each secondary server in a high-availability cluster. Secondary servers in a high-availability cluster must disable logical logging on temporary tables.

Set the following configuration parameters to the same value on both database servers in a HDR pair:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT

Related information

TEMPTAB_NOLOG configuration parameter on page

Cluster transaction coordination

You can configure your high-availability cluster so that when a client session issues a commit, the server blocks the session until the transaction is applied in that session, on a secondary server, or across the cluster. Set the CLUSTER_TXN_SCOPE configuration parameter or run the `SET ENVIRONMENT CLUSTER_TXN_SCOPE` statement to configure this behavior.

Multistep client operations that are performed on different high-availability cluster servers or in different sessions with high-availability cluster servers can fail because of asynchronous log processing. If a client application loads data onto a cluster server, and then attempts to process the same data on a second cluster server before the data is replicated to the second server, the operation fails. The client transaction must be applied on the second server before its data can be further processed.

Cluster transaction coordination causes client applications to wait for either cluster-wide or secondary-server application of transactions before the transaction commits are returned. This process prevents operation failures and ensures that the steps of multistep processes occur in serial order.

The different scopes for cluster transaction are:

- SESSION: When a client session issues a commit, the database server blocks the session until the effects of the transaction commit are returned to that session. After control is returned to the session, other sessions at the same database server or on other database servers in the cluster might be unaware of the transaction commit and the transaction's effects.
- SERVER: When a client session issues a commit, the database server blocks the session until the transaction is applied at the database server from which the client session issued the commit. Other sessions at that database server are aware of the transaction commit and the transaction's effects. Sessions at other database servers in the cluster might be unaware of the transaction's commit and its effects. This behavior is default for high-availability cluster servers.
- CLUSTER: When a client session issues a commit, the database server blocks the session until the transaction is applied at all database servers in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY. Other sessions at any database server in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY, are aware of the transaction commit and the transaction's effects.

Cluster transaction coordination was introduced in HCL OneDB™ version 11.70.xC6. Before HCL OneDB™ version 11.70.xC6, high-availability cluster servers had the following default behaviors:

- Primary servers had a cluster transaction scope of SERVER.
- Read-only secondary servers were in the dirty-read isolation level, and could read uncommitted data.
- Updatable secondary servers had a cluster transaction scope of SESSION.

Setting a CLUSTER_TXN_SCOPE value to CLUSTER does not change the behavior that is specified by the DRINTERVAL configuration parameter value. When a client application commits a transaction on a primary server, the primary server sends the HDR secondary server logical log buffers at maximum intervals that are specified by the DRINTERVAL configuration parameter. After the primary server sends logical log buffers to the HDR secondary server, it returns control to a session, but the session still does not receive a commit until the transaction is applied on all cluster servers.

Configuring secure connections for high-availability clusters

For a high-availability cluster to function, the database servers must establish trusted connection with each other. Secure connections between cluster servers by using a trusted-host file on each cluster server and including the connection security option in `sqlhosts` file entries.

About this task
The secure ports that are specified in `sqlhosts` files are used only for communication between database servers. Client applications cannot connect to secure ports.

To configure a trusted environment for replication, complete the following steps for each cluster server:

1. Edit the `sqlhosts` file on each host that contains a cluster server:
    a. Add an entry for each cluster server that is running on that host, and include the s=6 option.
    b. Add an entry for each other cluster server that participates in the cluster, and do not include the s=6 option.
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.
   Do not specify a non-network protocol such as onipcshm, onipcstr, or onipcnmp.
3. Specify trusted-host information.
   Trusted-host information can be specified in the following ways:
   Choose from:
    - Create a `hosts.equiv` file in the `$ONEDB_HOME/etc` directory, and then manually add entries to the file.
    - Create a trusted-host file in the `$ONEDB_HOME/etc` directory, and then manually add entries to the file. You must set the REMOTE_SERVER_CFG configuration parameter to the trusted-host file's name and set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1`.
    - Run the admin() or task() function with the **cdr add trustedhost** argument, and specify trusted-host information. Trusted-host information that is specified by the **cdr add trustedhost** argument propagates to all servers in the high-availability cluster. Do not run this function if you have manually entered trusted-host information on any of the database servers in a high-availability cluster or Enterprise Replication domain.
4. Create a server alias for running utilities and client applications.

Example

For example, set the **ONEDB_SERVER** environment variable to the alias to run utilities such as onstat and client applications such as DB-Access.

Remote standalone secondary servers

These topics provide an overview of setting up and configuring remote standalone (RS) secondary servers in a high availability environment.

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 effect 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, HCL OneDB™ 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 or 2.

Related information

TEMPTAB_NOLOG configuration parameter on page

Index page logging

You must enable index page logging to use an RS secondary server.

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 *HCL OneDB™ 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.

About this task

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 ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure.You can also set up an RS secondary server using standard ON-Bar command 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 *HCL OneDB™ User-Defined Routines and Data Types Developer's Guide*. For information about how to install DataBlade® modules, see the *HCL OneDB™ 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 on page 381. 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.
6. 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. For this example, **ServerA**'s DBSERVERNAME and HA_ALIAS configuration parameters are both set to `ServerA`, and **ServerB**'s DBSERVERNAME and HA_ALIAS configuration parameters are both set to `ServerB`.

   Using the database server's HA_ALIAS configuration parameter value, **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 on page 381. 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 7 on page 381.

   Important: You must complete steps 5 on page 381 through 6 on page 381 during the same session. If you must shut down and restart the secondary database server after step 5 on page 381, you must redo step 5 on page 381.
7. 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.

Results

Table 43. 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_ha_alias password*	
4.	ON-Bar command onbar -b -L 0	
5.		ON-Bar command onbar -r -p or onbar -r -p -e
6.		onmode command onmode -d RSS *primary_ha_alias 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 7 on page 381 must be performed).
7.		ON-Bar command onbar -r -l

Table 43. Steps to start a primary with an RS secondary server for the first time (continued)

Step	On the primary	On the RS secondary
		This step is required only when the secondary database server prompts you to recover the logical-log files from the tape backup.

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.

About this task

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 on page . Alternatively you can initialize the old primary server by running the following command:

   ```
   oninit -PHY
   ```

   See The oninit utility on page 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 424 and *HCL OneDB™ 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.

About this task

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 `$ONEDB_HOME/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.

To see information about the data being sent to the log-staging directory set for a RS secondary server, run the onstat -g rss verbose command on the RS secondary server.

If the write to the staging file fails, the RS secondary server raises event alarm 40007.

See *HCL OneDB™ 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.

Results

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 *HCL OneDB™ 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 *HCL OneDB™ 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
```

Flow control for remote standalone secondary servers

Flow control provides a way to limit log activity on the primary server so that remote standalone (RS) secondary servers in the cluster do not fall too far behind on processing transactions. Enabling flow control ensures that logs on RS secondary servers remain current if the servers are on a busy or intermittent network.

Set the RSS_FLOW_CONTROL configuration parameter on the primary server to enable flow control. All RS secondary servers in the cluster are affected by the primary server's RSS_FLOW_CONTROL configuration parameter setting. When flow control is active, users connected to the primary server may experience slower response time.

Logs are always sent to the RS secondary server in the order in which they were received.

To check if flow control is active for a RS secondary server, use the onstat -g rss verbose command, and compare the `RSS flow control` value to the `Approximate Log Page Backlog` value. If the `Approximate Log Page Backlog` is higher than the first value of `RSS flow control`, flow control is active. If the `Approximate Log Page Backlog` is lower than the second value of `RSS flow control`, flow control is disabled.

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 HCL OneDB™.

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 373.

Setting up a shared disk secondary server

You set up a shared disk system by configuring the primary server, configuring the SD secondary server, and starting the SD secondary.

Setting up the SD primary server

About this task

To set up the primary server:

1. Set the SDS_TIMEOUT configuration parameter to specify the amount of time in seconds that the primary server waits for a log position acknowledgment to be sent from the SD secondary server.
2. Configure the alias name of the SD primary server by running the following command:

   ```
   onmode -d set SDS primary ha_alias
   ```

   The server name that is specified by *ha_alias* becomes the primary server of the shared disk environment and the source of logs for the SD secondary server.

Setting up the SD secondary server

To set up the SD secondary server:

1. Set the following configuration parameters in the configuration file.
   - HA_ALIAS: set to define an alias for server-to-server communication in a high-availability cluster.
   - SDS_ENABLE: set to 1 (enable) on the secondary server to enable the shared disk environment.
   - SDS_PAGING: set to 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: set to the temporary dbspace for the SD secondary server that is dynamically created when the server is first started.

    - SDS_LOGCHECK: set to the number of seconds to delay a failover if network communications between the primary and secondary servers is temporarily unavailable.
    - TEMPTAB_NOLOG: set to 1 or 2 to prevent logical logging on temporary tables.
    - UPDATABLE_SECONDARY: set to a positive integer if you want to enable client applications to perform update, insert, and delete operations on the secondary server.
2. Set the following configuration parameters to match those on the primary server:
    - ROOTNAME
    - ROOTPATH
    - ROOTOFFSET
    - ROOTSIZE
    - PHYSFILE
    - LOGFILES
    - LOGSIZE
    - DISK_ENCRYPTION

   You can set other configuration parameters to match those of the primary server except for DBSERVERALIASES, DBSERVERNAME, and SERVERNUM.
3. Add an entry to the `sqlhosts` file to identify the primary server:

   ```
   #dbservername   nettype   host   servicename   options
    name           protocol  name   port
   ```
4. Start the SD secondary server using the oninit command.

   The SD secondary server processes any open transactions as a fast recovery in quiescent mode. There is increased memory usage in the LGR memory pool during fast recovery.
5. Examine the `online.log` file on the secondary server to verify that it completed processing open transactions and is in online mode.
6. Allow client applications to connect to the SD secondary server.

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 *HCL OneDB™ Administrator's Reference*.

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 374 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 Configure SMX connections on page 415 topic for additional information about setting up and configuring encryption between primary servers and SD secondary servers.

Flow control for shared-disk secondary servers

Flow control provides a way to limit log activity on the primary server so that shared-disk (SD) secondary servers in the cluster do not fall too far behind on processing transactions.

Set the SDS_FLOW_CONTROL configuration parameter on the primary server to enable flow control. All SD secondary servers in the cluster are affected by the primary server's SDS_FLOW_CONTROL configuration parameter setting. When flow control is active, users connected to the primary server may experience slower response time.

Logs are always sent to the SD secondary server in the order in which they were received.

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 44. Secondary server setup information

Secondary server type	See
HDR secondary	See High-availability cluster configuration on page 372, and information about starting an HDR pair using external backup and restore in the *HCL OneDB™ Backup and Restore Guide*.
RS secondary	See Remote standalone secondary servers on page 379.
SD secondary	See Shared disk secondary servers on page 387

How data initially replicates

About this task

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.

Results

For detailed instructions about replicating data, see Starting HDR for the First Time. The *HCL OneDB™ Backup and Restore Guide* explains how to start replication using ON-Bar.

You must perform the initial backup with a storage-space backup.

Replication of primary-server data to secondary servers

All secondary server types use logs to replicate primary-server replicate data. The primary server sends its entire logical log to HDR and RS secondary servers, but only the log page's position to SD secondary servers.

Index page logging can be used by all secondary servers, but is required for replication to RS secondary servers.

Databases must use transaction logging to be replicated.

Warning: If the primary server and secondary server disconnect from each other, and are allowed to independently run as standard servers or primary servers, then high-availability data replication might have to be reestablished.

Replication to HDR secondary servers

There are three synchronization modes that the primary database server can use to replicate data to an HDR secondary server:

- *Fully synchronous mode*, where transactions require acknowledgement of completion on the HDR secondary server before they can complete.

  Data integrity is highest when you use fully synchronous mode, but system performance can be negatively affected if client applications use unbuffered logging and have many small transactions.
- *Asynchronous mode*, where transactions do not require acknowledgement of being received or completed on the HDR secondary server before they can complete.

  System performance is best when you use asynchronous mode, but if there is a server failure, data can be lost.
- *Nearly synchronous mode*, where transactions require acknowledgement of being received on the HDR secondary server before they can complete.

  Nearly synchronous mode can have better performance than fully synchronous mode and better data integrity than asynchronous mode. If used with unbuffered logging, SYNC mode, which is turned on when DRINTERVAL is set to -1, is the same as nearly synchronous mode.

The synchronization mode is controlled by the combination of DRINTERVAL configuration parameter value, HDR_TXN_SCOPE configuration parameter value, and database logging type.

The following two figures illustrate replication from a primary server to an HDR secondary server.

Figure 56. How data replicates from a primary to HDR secondary server

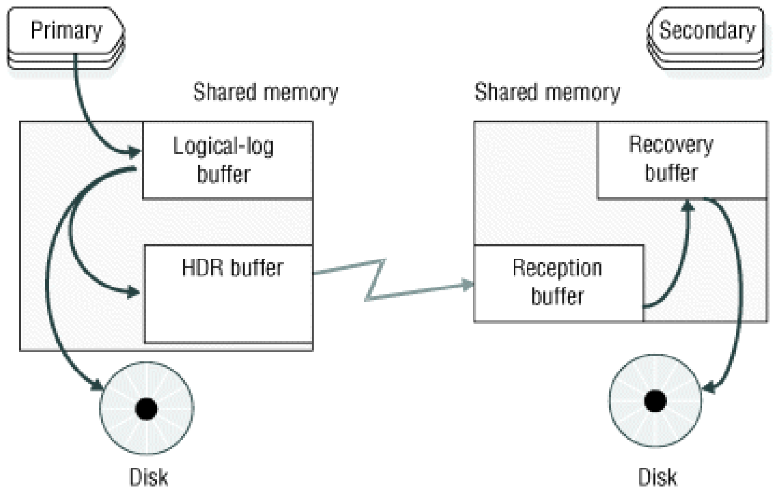
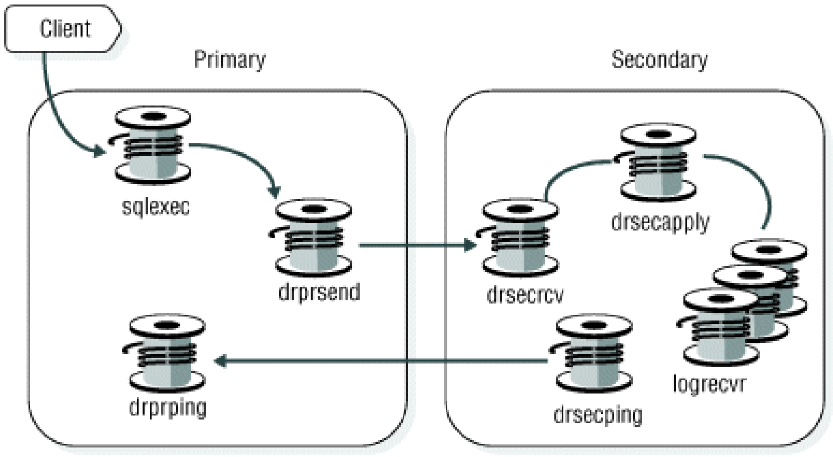


Figure 57. Threads that manage data replication



The contents of the primary server's logical-log buffer are copied to the shared-memory *data-replication buffer* and flushed to disk. If the primary server is using fully synchronous or nearly synchronous mode, it must receive an acknowledgement from the HDR secondary server before it can complete the logical-log flush. The primary server starts a **drprsend** thread to transmit the data-replication buffer across the network to the secondary server's **drsecrcv** thread, which then writes the data into the shared-memory *reception buffer*. The **drsecapply** thread copies the reception buffer to the recovery buffer. Both HDR and RS secondary servers use **logrecvr** threads to apply logical-log records their dbspaces. You can adjust the number of **logrecvr** threads by changing the value of the OFF_RECVRY_THREADS configuration parameter.

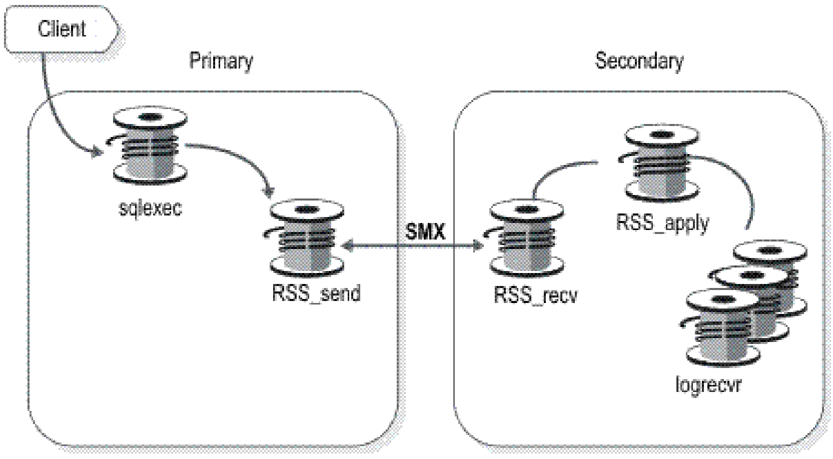
The **drprping** and **drsecping** threads send and receive messages to monitor the connection between two servers.

Replication to RSS secondary servers

Because checkpoints between a primary server and an RS secondary server are asynchronous, RS secondary servers require index page logging.

The following figure illustrated replication from a primary server to an RS secondary server.

Figure 58. Threads that manage data replication for RS secondary servers



If the primary server can verify that it is connected to an RS secondary server, the **RSS_send** thread copies a page from either the disk or the logical-log buffer to the data-replication buffer. The **RSS_Send** thread uses a Server Multiplexer Group (SMX) connection to send the data-replication buffer to the RS secondary server's **RSS_recv** thread. The **RSS_recv** thread then writes the data into the *reception buffer*. The **RSS_apply** thread copies the reception buffer to the recovery buffer.

Unlike with HDR fully synchronous mode or nearly synchronous mode, the primary server does not require acknowledgment from the secondary server before sending the next buffer. The primary server sends up to 32 unacknowledged data-replication buffers before the **RSS_send** thread waits for the **RSS_Recv** thread to receive an acknowledgment from the RS secondary server.

Replication to SD secondary servers

SD secondary servers read logical log pages from disk and then apply the data to their memory data buffers.

Fully synchronous mode for HDR replication

HDR fully synchronous mode ensures that any transaction committed on a primary server was also committed on the HDR secondary server, which can protects transactional consistency if a failure occurs.

After the primary database server writes the logical-log buffer contents to the HDR buffer, it sends the 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.

To track synchronization, both the primary and HDR secondary server store the following information in their reserved pages:

- 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 that was sent or received
- The page number of the last logical-log record that was sent or received

To view this information, run the onstat -g dri ckpt command.

Checkpoints between database servers in an HDR 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 is specified by the DRTIMEOUT configuration parameter, the primary database server assumes that a failure occurred.

HDR fully synchronous mode has the following requirements:

- The DRINTERVAL configuration parameter on the primary and HDR secondary server must be set to `0`.
- The DRTIMEOUT configuration parameter on the primary and HDR secondary server must be set to the same value.

Administration can be easier if the operating-system times on the primary and HDR secondary servers are synchronized.

To turn on fully synchronous data replication, set the DRINTERVAL configuration parameter to `0`, and then use one of the following methods:

- Set the HDR_TXN_SCOPE configuration parameter to FULL_SYNC.
- Run SET ENVIRONMENT HDR_TXN_SCOPE 'FULL_SYNC';

Log records are applied in the order in which they were received. When the log transmission buffer contains many log records, the application of those log records on the HDR secondary server requires more time, and performance can be negatively affected. If this situation occurs, consider using nearly synchronous mode for HDR data replication.

Nearly synchronous mode for HDR replication

When you use nearly synchronous mode for HDR replication, the primary server flushes the logical-log buffer to disk after receiving acknowledgement that the HDR secondary server received a transmitted transaction. The primary server does not wait for acknowledgement that the transaction was committed on the HDR secondary server.

When the log transmission buffer contains many log records, the application of those log records on the HDR secondary server requires more time. Nearly synchronous mode for HDR replication provides better performance than fully synchronous mode, and better data integrity than asynchronous mode.

The primary server stores the following near-synchronization information in its reserved page:

- The number of unprocessed data replication buffers queued to the **drprsend** thread.
- The log unique value, the page number for the most recently paged log.
- The pointer to the thread-control block (TCB), the thread id in parentheses, and the log sequence number (LSN) of the commit that was performed by that thread.
- The LSNs of commits that are waiting for acknowledgement of being received on the HDR secondary.

To view this information, run the onstat -g dri que command.

HDR nearly synchronous mode has the following requirements:

- The DRINTERVAL configuration parameters on the primary and HDR secondary server must be set to `-1`, or the DRINTERVAL configuration parameter on the primary server must be set to `0`.
- The DRTIMEOUT configuration parameters on the primary and HDR secondary server must be set to the same value.
- The operating-system time on the primary and HDR secondary servers must be synchronized.

To turn on nearly synchronous data replication, set the DRINTERVAL configuration parameter to `0`, and then use one of the following methods:

- Set the HDR_TXN_SCOPE configuration parameter to NEAR_SYNC.
- Run `SET ENVIRONMENT HDR_TXN_SCOPE 'NEAR_SYNC';`

Asynchronous mode for HDR replication

Asynchronous HDR replication means that the primary server does not wait for a response from the HDR secondary server before flushes the logical log to disk. Asynchronous HDR replication can increase replication speed, but transactions can be lost.

There are multiple ways to turn on asynchronous mode for HDR replication:

- Set the DRINTERVAL configuration parameter to a positive integer value.
- Set the DRINTERVAL configuration parameter to 0, and set the HDR_TXN_SCOPE configuration parameter to ASYNC.
- Run the following statement:

  ```
  SET ENVIRONMENT HDR_TXN_SCOPE 'ASYNC';
  ```

In asynchronous mode, the primary database server flushes the logical-log to disk after it copies the contents of the logical-log buffer to the data-replication buffer. 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 since the records were sent to the HDR secondary database server exceeds the value of the primary server's DRINTERVAL configuration parameter.

To reduce the risk of lost transactions in a cluster that uses asynchronous replication, use unbuffered logging for all the databases. Unbuffered logging reduces the amount of time between transaction-record writing and transfer. If your primary server uses buffered logging, and you receive an `error -7350 Attempt to update a stale version of a row` message, switch to unbuffered logging.

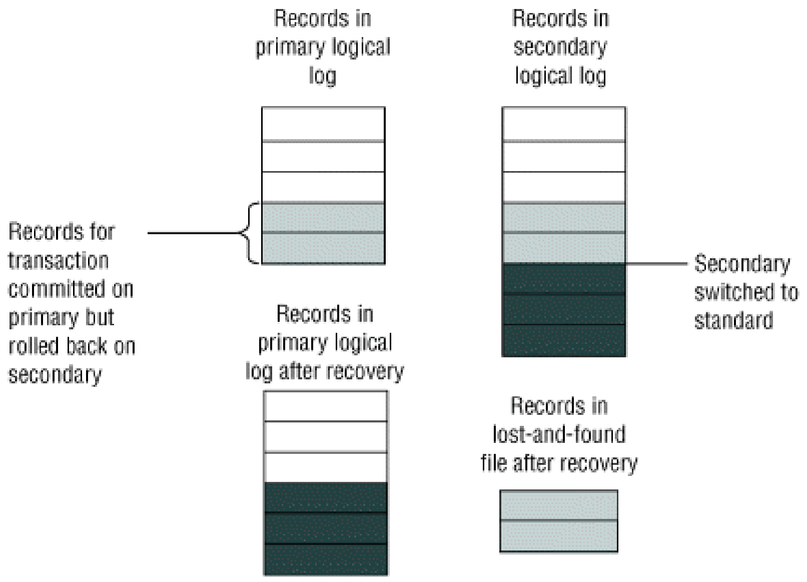
If a failover does occur, but the primary server is restarted with data replication, transactions that were committed on the primary server and not committed on the secondary server are stored in a file that is specified by the DRLOSTFOUND configuration parameter.

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 59. Using a lost-and-found file


Records in primary logical log
Records in secondary logical log
Records for transaction committed on primary but rolled back on secondary
Secondary switched to standard
Records in primary logical log after recovery
Records in lost-and-found file after recovery

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.

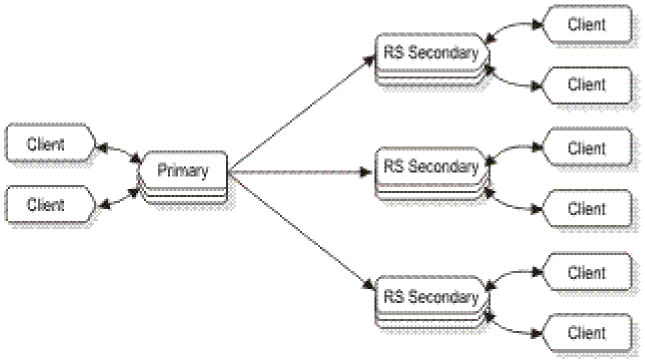
Data replication configuration examples

These topics describe some examples of how a data replication environment can be configured.

Remote standalone secondary configuration examples

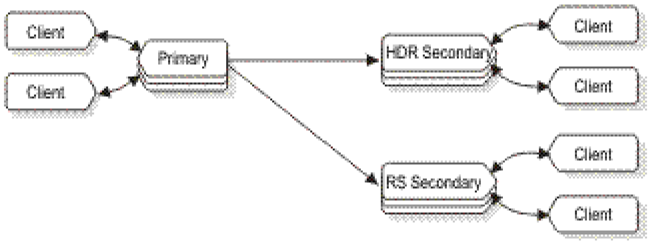
The following figure illustrates an example of a configuration consisting of multiple RS secondary servers. This configuration would be useful in a situation where the primary server is located a long distance from the RS secondary servers or if the network speed between the primary server and the RS secondary server is slow or erratic. Because RS secondary servers use fully duplexed communication protocols, and do not require synchronous checkpoints processing, the primary-server's performance is usually unaffected.

Figure 60. Primary server with three RS secondary servers

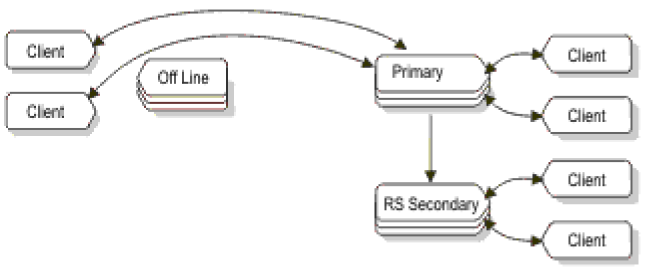


The next illustration shows an example of a configuration of an RS secondary server along with an HDR secondary server. In this example, the HDR secondary provides high availability while the RS secondary provides additional disaster recovery if both the primary and HDR secondary servers are lost. The 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.

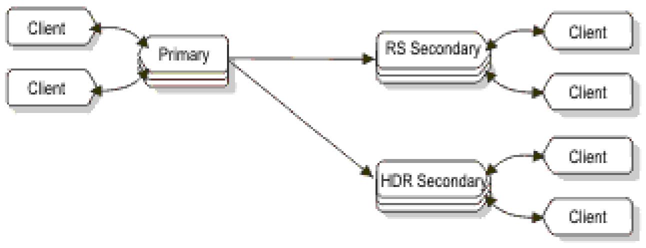
Figure 61. Primary server with HDR secondary and RS secondary servers



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 62. Failover of primary server to HDR secondary server

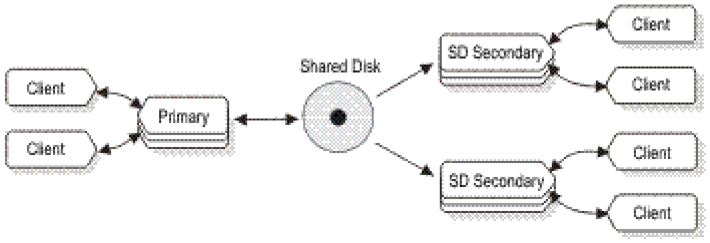


If the original primary is going to be offline for an extended period of time, then the RS secondary server can be converted to an HDR secondary server. Then when the original primary comes back online, it can be configured as an RS secondary server, as in the following illustration:
Figure 63. RS secondary server assuming role of HDR secondary server

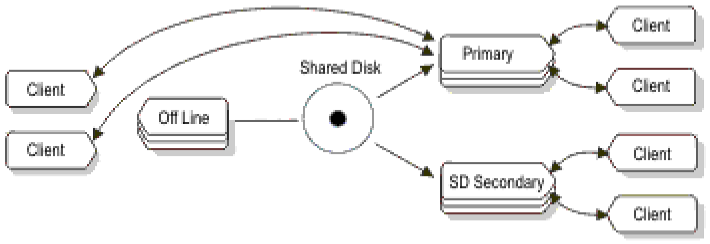


Shared disk secondary configuration examples

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 64. Primary server configured with two SD secondary servers

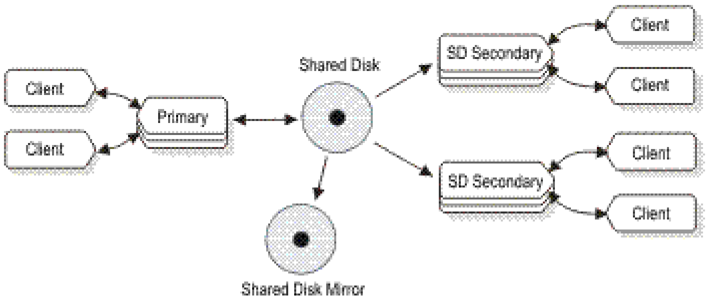


Because both of the SD secondary servers are reading from the same disk subsystem, they are both equally able to assume the primary server role. The following figure illustrates a situation in which the primary server is offline.
Figure 65. SD secondary server assuming role of primary server



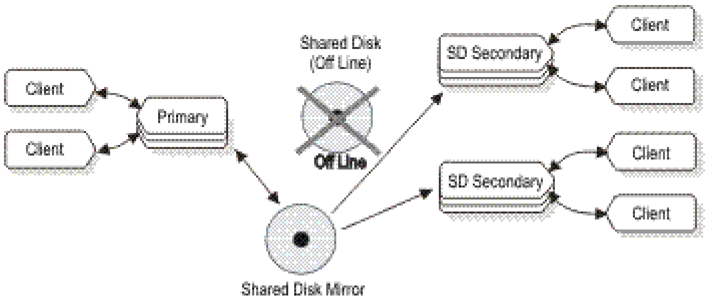
There are several ways to protect against hardware failure of the shared disk. Probably the most common way is to configure the disk array based on RAID technology (such as RAID-5). Another way to protect against disk failure is to use SAN (Storage Area Technology) to include some form of remote disk 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 66. Primary server and SD secondary servers with mirrored disks



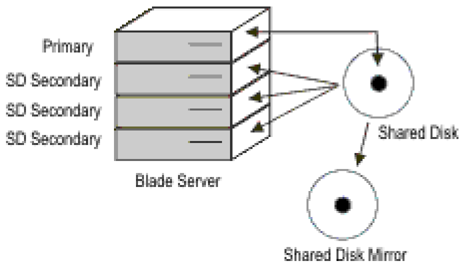
In the event of a disk failure, the servers can be reconfigured as in the following illustration:

Figure 67. Shared disk mirror after failure of primary shared disk



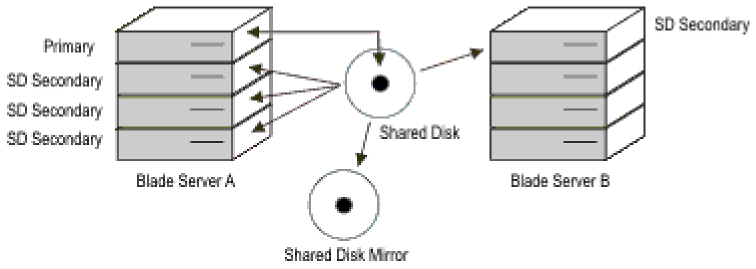
In addition to configuring a mirrored disk subsystem as in the previous illustration, you might want to configure additional servers. For example, you might want to use the primary and two SD secondary servers within a single blade server enclosure. By placing the server group within a single blade server, 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 68. Primary and SD secondary servers in a blade server



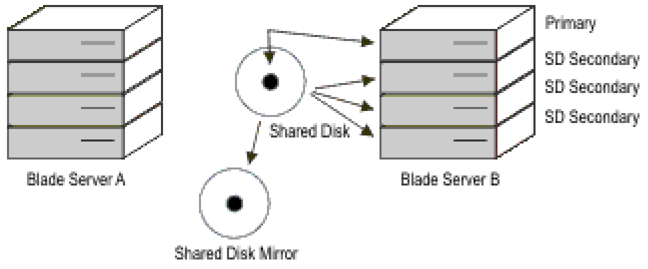
You might decide to avoid the possible failure point of a single blade server by using multiple blade servers, as in the following illustration.

Figure 69. Multiple blade server configuration to prevent single point of failure



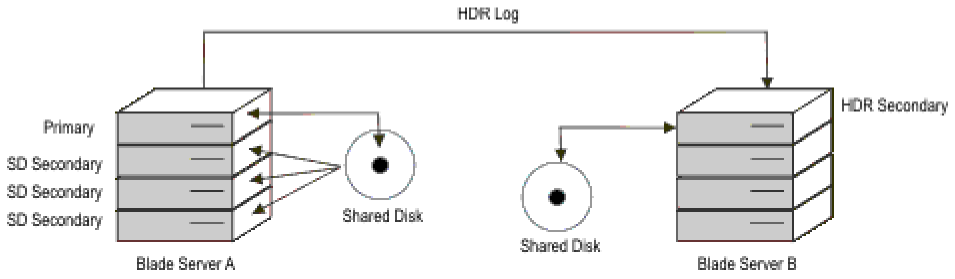
In the previous illustration, if Blade Server A fails, it would be possible to transfer the primary server role to the SD secondary server on Blade Server B. Since it is possible to bring additional SD secondary servers online very quickly, it would be possible to dynamically add additional SD secondary servers to Blade Server B as in the following illustration.

Figure 70. Failover after failure of blade server



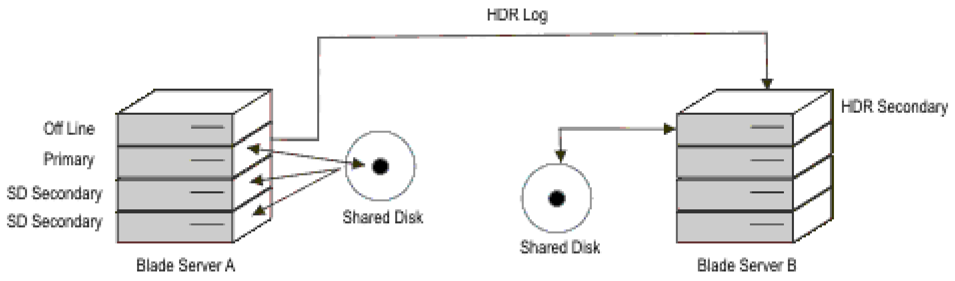
Because of limits on the distance between the primary and mirrored disks that disk mirroring can support, you might be concerned about using shared disks and relying on shared disk mirroring to provide disk availability. For example, you might want significant distance between the two copies of the disk 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 71. HDR secondary server in blade server configuration



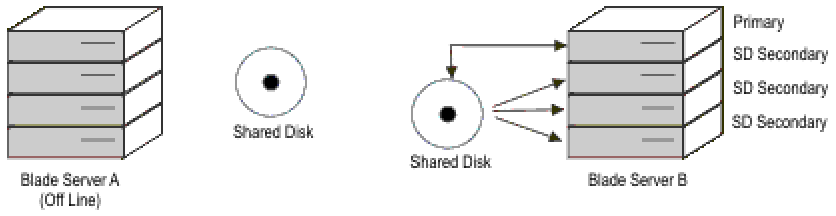
In the configuration shown in the previous illustration, if the primary node fails, but the shared disks are intact and the blade server is still functional, it is possible to transfer the primary server role from the first server in Blade Server A to another server in the same blade server. Changing 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 72. 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.

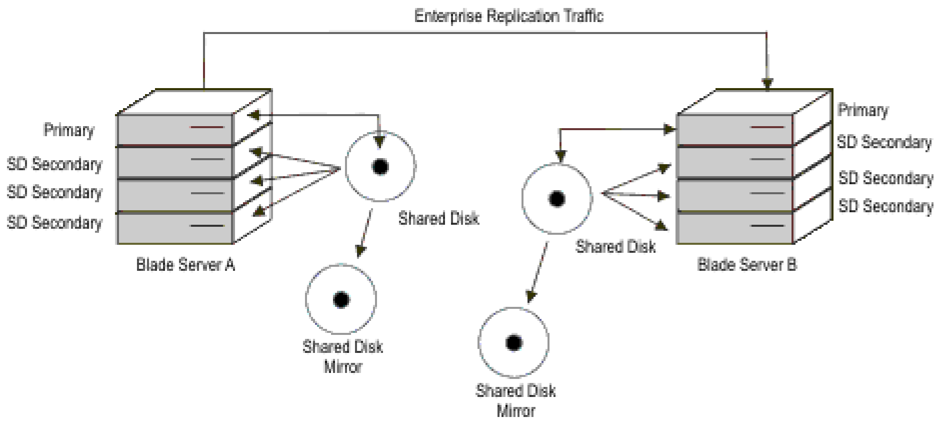
Figure 73. Failure of entire blade server



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 74. 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.

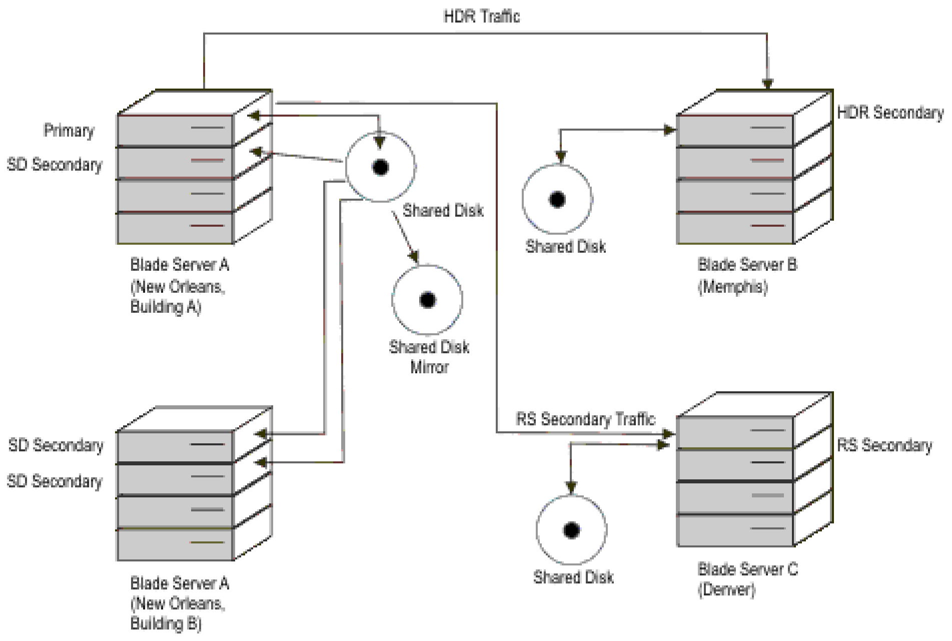
In general, an HDR Secondary server provides backup for SD secondary servers and provides support for a highly available system which is geographically remote from the main system. RS secondary servers provide additional availability for the HDR secondary and are viewed as a disaster-availability solution. 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.

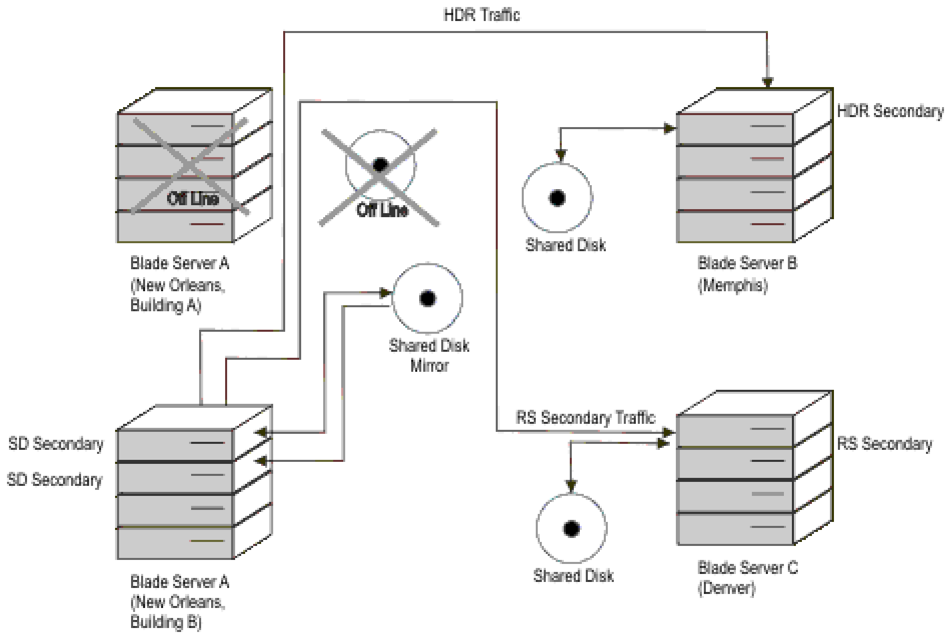
However, even a two-tiered approach might not be enough. A hurricane in one region can create tornadoes hundreds of miles away. To protect against this, consider adding a third tier of protection, such as an RS secondary server located one or more thousand miles away. This three-tier approach provides for additional redundancy that can significantly reduce the risk of an outage.

Figure 75. Configuration for three-tiered server availability



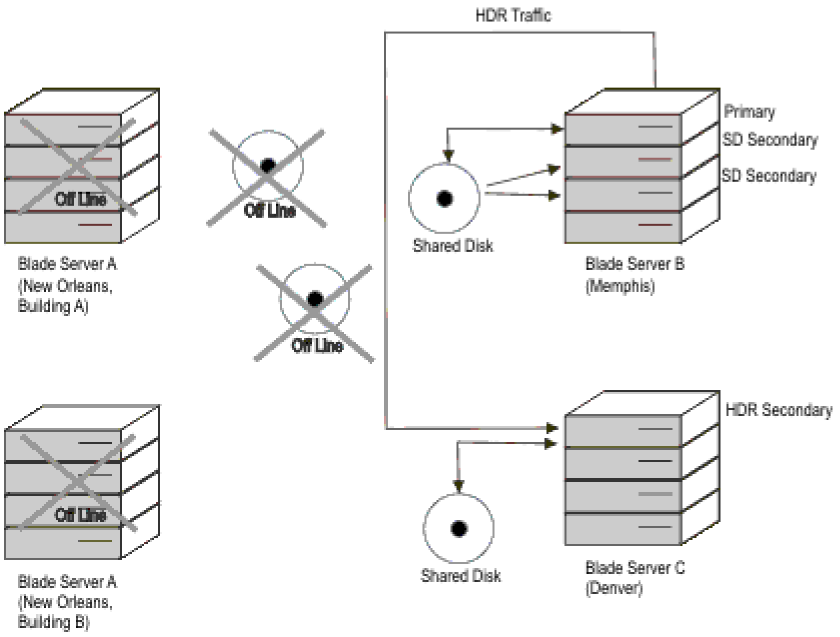
Now suppose that a local outage occurred in Building-A on the New Orleans campus. Perhaps a pipe burst in the machine room causing water damage to the blade server and the primary copy of the shared disk subsystem. You can switch the role of primary server to Building-B by running onmode -d make primary 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 76. 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 77. Second tier of protection



An even larger outage which affected both sites would require switching to the most remote system.

Figure 78. Third tier of protection

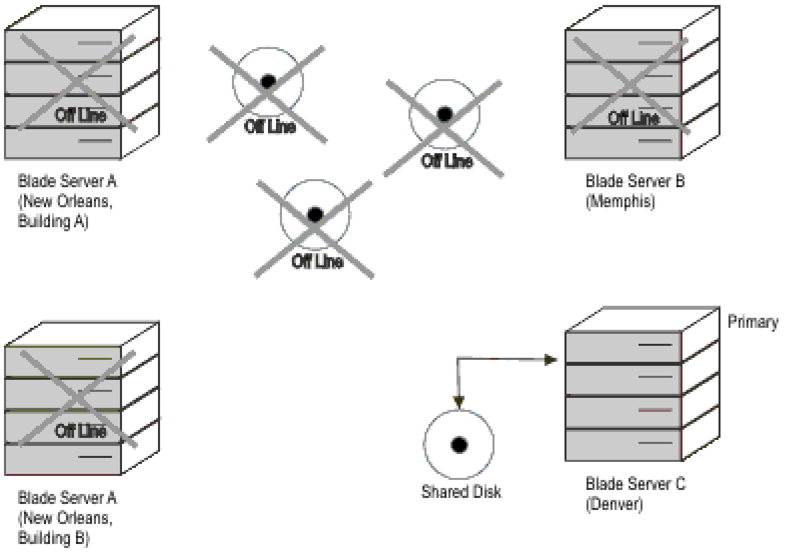


Table 45. Suggested configurations for various requirements

Requirement	Suggested configuration
You periodically must increase reporting capacity	Use SD secondary servers

Table 45. Suggested configurations for various requirements (continued)

Requirement	Suggested configuration
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 a local HDR secondary server that is running fully synchronous mode or nearly synchronous mode for data replication, and also using 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
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, HCL OneDB™ 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 ENCRYPT_SMX configuration parameter on page 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
```

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 427 for information about **committed read** and **committed read last committed** isolation levels on secondary servers.

Use of temporary dbspaces for sorting and temporary tables

About this task

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 165 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 204.
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.

Results

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 or 2 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 the configuration parameters for an HDR replication pair

About this task

Certain configuration parameters must be set to the same value on both database servers in a HDR replication pair (as listed under Database server configuration requirements for clusters on page 374.) Configuration parameters that can have different values on each database server can be changed individually.

To make changes to `onconfig` files:

1. Bring each database server offline with the onmode -k option.
   If automatic failover by Connection Managers or automatic switchover from DRAUTO settings of 1 or 2 are enabled, bring the HDR secondary server offline first.
2. Change the parameters on each database server.
3. Starting with the database server that was last brought offline, bring each database server back online.
   Example

   For example, if you brought the HDR secondary database server offline last, bring the HDR secondary database server online first. Table 1 lists the procedures for bringing the primary and secondary database servers back online.

Results

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.

Changing the logging mode of databases

About this task

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.

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.

Tip: When adding a dbspace on the primary server of a high-availability cluster that has one or more SD secondary servers, the `online.log` of an SD secondary server might show this error: "Assert Failed: Page Check Error". If that happens, shut down and restart that SD secondary server. After restarting that SD secondary server, the newly added dbspace will be available and fully functional.

Renaming chunks

About this task

If you use symbolic links for chunk path names, you can rename chunks while HDR is operating. For instructions on renaming chunks, see the *HCL OneDB™ 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 *HCL OneDB™ 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 high-availability cluster servers, if the status of a chunk (down or 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.

About this task

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.
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 completed.
4. Run onmode -m on the secondary database server to verify that a checkpoint was also completed on the secondary database server.

Results

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 197.

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 353.

When you drop a chunk from the primary database server, HCL OneDB™ 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, HCL OneDB™ 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 354.

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 Manage the physical log on page 336.

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 Manage logical-log files on page 309. HCL OneDB™ 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 the configuration parameters for an HDR replication pair on page 411.

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.

Configure 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.Server Multiplexer Group (SMX) is a communications interface that supports multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.

Reduce latency between servers

You can reduce latency between high-availability servers by increasing the number of pipes that are used for SMX connections between the servers. Set the SMX_NUMPIPES configuration parameter to the number of pipes.

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.

Encrypt SMX connections

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.

Set the wait time for SMX activity between servers

You can set the SMX_PING_INTERVAL and SMX_PING_RETRY configuration parameters to adjust the interval that secondary server in a high-availability cluster waits for activity from the primary server. Use the SMX_PING_INTERVAL configuration parameter to specify the number of seconds in a timeout interval, where a secondary server waits for activity from the primary server in an SMX connection.

Use the SMX_PING_RETRY configuration parameter to specify the maximum number of times that a secondary server repeats the timeout interval that is specified by the SMX_PING_INTERVAL configuration parameter if a response from the primary server is not received. If the maximum number is reached without a response, the secondary server prints an error message in the `online.log` and closes the Server Multiplexer Group (SMX) connection.

Compress data through SMX connections

You can specify the level of compression that the database server uses before sending data from the source database server to the target database server with the SMX_COMPRESS configuration parameter. Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data.

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 379). 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.

Encrypting data traffic between HDR database servers

Before you begin

About this task

You can use HCL OneDB™ 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 ENCRYPT_SMX configuration parameter on page for more information.

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 the configuration parameters for an HDR replication pair on page 411.
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.

Example

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 *HCL OneDB™ 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 341.

For more information about LRU tuning, see the *HCL OneDB™ 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 one of the following database server types:

- Standalone server
- Remote standalone (RS) secondary server
- High-availability data replication (HDR) secondary server
- Shared-disk (SD) secondary server
- Enterprise Replication server

Using the ifxclone utility, the database administrator can quickly, easily, and securely create a clone server from a running HCL OneDB™ 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.

Data is transferred from the source server to the target server over the network using 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.

About this task

The general steps to create a clone of a server are as follows:

1. Set the following environment variables on the target server:
   Choose from:
   - ONEDB_HOME
   - ONEDB_SERVER
   - ONCONFIG
   - ONEDB_ SQLHOSTS
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:
      Result
      ```
      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:
      Result
      ```
      $ 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.

Example

Use the following steps to clone a server using the ONCONFIG and ONEDB_ SQLHOSTS 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:

```
#dbservername  nettype   hostname     servicename   options
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:

```
#dbservername  nettype   hostname     servicename   options
 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
   ```

What to do next

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 HCL OneDB™ password to create the instance.

Creating a clone of a primary server

Use the ifxclone utility to create a clone of a primary server.

About this task

The general steps to create a clone of a server are as follows:

1. Verify that the ENABLE_SNAPSHOT_COPY configuration parameter on the source server is set to `1`.
2. Verify that the CDR_AUTO_DISCOVER configuration parameter is set to 1, and that the REMOTE_SERVER_CFG file is set on all cluster servers.
3. Set the following environment variables on the target server:
   Choose from:
   - ONEDB_HOME
   - ONEDB_SERVER
   - ONCONFIG
   - ONEDB_ SQLHOSTS
4. On the target server, create all of the chunks that exist on the source server. You can use the --createchunkfile option (-k) to automatically create cooked chunks on the target server. Follow these steps to create the chunks:

   a. On the source server, run the onstat -d command to display a list of chunks

   b. On the target server, log-in as user **informix** and use the commands touch, chown, and chmod to create the chunks.
      Example
      For example, to create a chunk that is 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.

5. Create an `sqlhosts` file on the target server's host, and add the target server's connectivity information to the file.
6. On the source server, run the admin() or task() function with the cdr add trustedhost argument, including the target server's trusted-host information.
7. While still logged in as user **informix**, run the ifxclone utility with the --autoconf option (-a) and other appropriate parameters on the target system on which the clone server is started.

Note: If trusted-host information was added manually, do not use the --autoconf option; you must configure trusted hosts and `sqlhosts` file information manually.

Example

Example: Cloning a primary server

For this example, you have the following system:

- **server_1** is the source server and has the following `sqlhosts` file entries:

```
#dbservername  nettype   hostname          servicename  options
 server_1      onsoctcp  host1.example.com  123
```

- **server_1** has the following configuration parameter settings:
  - ENABLE_SNAPSHOT_COPY 1
  - CDR_AUTO_DISCOVER 1
  - REMOTE_SERVER_CFG authfile.server_1
- **server_1** has the following trusted-host file entries

```
#trustedhost
host1
host1.example.com
host2
host2.example.com
```

- **server_2** is the target server, is on **host2.example.com**, and uses 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. You can use the --createchunkfile option (-k) to automatically create cooked chunks on the target server.
2. Run the ifxclone utility on the target server as user **informix**:

```
ifxclone -T -S server_1 -I host1.example.com -P 123 -t server_2
         -i host2.example.com -p 456 -a -k
```

```
ifxclone -T -S server_1 -I host1.example.com -P 123 -t server_2
         -i host2.example.com -p 456 -a
```

The ifxclone utility modifies the `sqlhosts` file on the source server and creates a copy of the file on the new target server:

```
#dbservername  nettype   hostname          servicename  options
 server_1      onsoctcp  host1.example.com  123
 server_2      onsoctcp  host2.example.com  456
```

The ifxclone utility also propagates the trusted-host file on the source server to the target server.

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.

You cannot use the dbexport utility on HDR secondary servers or shared disk (SD) secondary servers. The dbexport utility is supported on a remote standalone (RS) secondary server only if the server is set to stop applying log files. Use the STOP_APPLY configuration parameter to stop application of log files.

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
- BSON
- BYTE (stored in the table)
- CHAR
- CLOB
- DATE
- DECIMAL
- DATETIME
- FLOAT
- INT
- INT8
- INTERVAL
- JSON
- 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
- ondblog
- ON-Monitor
- onparams
- onspaces

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.

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 410 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 *HCL OneDB™ 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 *HCL OneDB™ Guide to SQL: Reference* and *HCL OneDB™ 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 *HCL OneDB™ 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.

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 46. Mode changes on the primary database server

On the primary	On the secondary	To restart HDR
From any mode to offline: (onmode -k)	Secondary displays: DR: Receive error. HDR is turned off. The mode remains read-only.	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:

Table 46. Mode changes on the primary database server (continued)

On the primary	On the secondary	To restart HDR
	If DRAUTO is set to `0`, the mode remains read-only. If DRAUTO is set to `1`, the secondary server switches to standard type and can accept updates. (If DRAUTO is set to `2` , the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.)	• 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 546.
To online, quiescent, or administration mode: (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 47. 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 545.
Quiescent (onmode -s)	Primary does not receive errors. HDR remains on.	Use onmode -m on the secondary.

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:
     ```
     $ONEDB_HOME/bin/hdrmksec.sh ServerB
     ```
   - On Windows™ systems:
     ```
     hdrmksec.bat ServerB
     ```

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 *HCL OneDB™ 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:

```
HCL OneDB   -- 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.

```
HCL OneDB     -- 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.

```
HCL OneDB     -- 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 *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ 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 417 for details.

See Configure SMX connections on page 415 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_ha_alias password* command on the primary server. During secondary server setup, you include the password when you run the onmode -d RSS *rss_ha_alias 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.

Example

Examples

The following commands establish an RS secondary server named **ServerB** using a password of `s#cure`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

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`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

```
onmode -d change RSS ServerB s@fest
```

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_ha_alias password* command on the primary server. During secondary server setup, you include the password when you run the onmode -d RSS *rss_ha_alias 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.

Example

Examples

The following commands establish an RS secondary server named **ServerB** using a password of `s#cure`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

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`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

```
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. Configure the cluster environment for the following results:

- 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 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 distributed transactions (transactions that span multiple database servers) are running, any transaction that is running on the primary server at the time of server failure is terminated.

When failover occurs, whether it is manual or performed by the Connection Manager, the database server that receives failover must have the most advanced log-replay position of all active servers in the cluster. If the database server that receives failover does not have the most advanced log replay position, all transactions in the cluster are terminated and rolled back. Because the primary and SD secondary server read from the same physical disk, failover to an SD secondary server should occur first. If the failover server is an HDR secondary server, SD secondary servers are shut down.

For best Connection Manager failover performance, use the FOC ORDER=ENABLED setting in the Connection Manager configuration file, and set the HA_FOC_ORDER configuration parameter on the cluster's primary server.

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.

Connection management through the Connection Manager

Connection Managers can control automatic failover for high-availability clusters, monitor client connections and direct requests to appropriate database servers, act as proxy servers and handle client/server communication, and prioritize connections between application servers and the primary server of a high-availability cluster. Connection Managers support high-availability clusters, replicate sets, server sets, and grids.

Automatic failover for database servers

If the Connection Manager detects that a primary server of a high-availability cluster has failed, it can promote a secondary server to the role of the primary server.

If you use multiple Connection Managers to manager failover for a cluster, you can enforce a consistent failover policy by setting the `onconfig` file HA_FOC_ORDER configuration parameter on the cluster's primary server. The value of the `onconfig` file HA_FOC_ORDER configuration parameter replaces the value of the `FOC` parameter's `ORDER` attribute in the configuration file of each Connection Manager that connects to the primary server.

Rule-based connection redirection and load balancing

Client applications can connect to a Connection Manager as if they are connecting to a database server. The Connection Manager gathers workload statistics from each server in the connection unit and uses service level agreements (SLAs) to manage and direct client connection requests to appropriate servers. When a client application makes a connection request through a redirect-mode SLA, the Connection Manager returns a database server's IP address and port number to the client application. The client application then uses the information to connect to the specified database server.

Connection Manager redirection takes place in the communication layer, so additional action is not required by client applications. Connection Managers can use redirection policies that are based on workload, latency, apply failures, or the apply backlog. Redirection can also be configured to occur round-robin.

Redirection-policy SLAs do not support connections from application that are compiled with Data Server Driver for JDBC and SQLJ version 3.5.1 or before, or with HCL OneDB™ Client Software Development Kit (Client SDK) 3.00 or before.

Proxy-server connection management

Proxy-mode SLAs and redirect-mode SLAs are similar; in both cases, the Connection Manager gathers workload statistics from connection-unit servers, and controls which servers receive client connection request servers. When a client application makes a connection request through a proxy-mode SLA, client/server communication travels through the Connection Manager. When a database server is behind a firewall, Connection Managers can act as proxy servers, and handle client/server communication.

Proxy-policy SLAs do not have the same version restrictions that redirect-policy SLAs have. Connections from application that are compiled with any version of Data Server Driver for JDBC and SQLJ, or with any version of HCL OneDB™ Client Software Development Kit (Client SDK) are supported.

Failover prioritization for application servers

You can install Connection Managers on the same hosts as application servers, and then prioritize the connections between each application server and the primary server of a high-availability cluster. This can help the highest priority application server maintain a connection to the cluster's primary server if a portion of the network fails.

Configuring connection management

To configure connection management, you must install software, set environments and connectivity information, create Connection Manager configuration files, and run the oncmsm utility.

About this task

To configure and start connection management, complete the following steps:

1. Install at least one Connection Manager as part of the HCL OneDB™ Client Software Development Kit (Client SDK) installation.
    a. If Connection Managers are installed on hosts where database servers are not installed, set each Connection Manager's host **ONEDB_HOME** environment variable to the directory the Connection Manager is installed into.
2. Modify connectivity information in the `sqlhosts` files that are on all client, database server, and Connection Manager hosts.
    a. If `sqlhosts` files are in host directories other than `$ONEDB_HOME/etc`, set host **ONEDB_ SQLHOSTS** environment variables to the appropriate `sqlhosts` file location.
    b. If Connection Managers or clients are installed on hosts where database servers are not installed, create a `sqlhosts` file on each host, and then set each Connection Manager's or client's host **ONEDB_ SQLHOSTS** environment variable to the location of the `sqlhosts` file.
3. If Connection Managers, application servers, or database servers are on an untrusted network, complete the following steps:
    a. Create a password file.
    b. Encrypt the password file by running the onpassword utility.
    c. Distribute the password file to the database servers that Connection Managers connect to.
    If the database servers are on other operating systems, distribute the unencrypted password file, and then encrypt it on the other operating systems.
4. On Connection Manager hosts, create a configuration file for each installed Connection Manager.
    a. If the configuration file is in a directory other than `$ONEDB_HOME/etc`, set the **CMCONFIG** environment variable to specify the file's location.
    b. If you define a service-level agreement that uses a transaction-latency or apply-failure redirection policy, start quality of data (QOD) monitoring by running the cdr define qod and cdr start qod commands.
5. Set `onconfig` parameters on the cluster database servers.
    a. If Connection Managers control failover for a high-availability cluster, set the DRAUTO configuration parameter to `3` on all managed cluster database servers, and set the HA_FOC_ORDER configuration parameter on the primary server of each cluster to a failover order.
    b. On each database server that uses multiple ports, set the DBSERVERALIASES configuration parameter to the alias names listed in Connection Manager and database server `sqlhosts` file entries.
6. **Optional:** Configure the `cmalarmprogram` script that is installed with Connection Managers.
7. Run the oncmsm utility to start Connection Managers.

Creating Connection Manager configuration files

About this task

To configure a Connection Manager, you must create a configuration file, and then load the configuration file by running the oncmsm utility.

A Connection Manager configuration file consists of two parts:

- The header, which contains Connection Manager parameters that are specific to the Connection Manager.
- The body, which consists of one or more connection unit sections that contain parameters and attributes that are specific to the defined connection units.

The following steps apply to all connection-unit types:

1. Create an ASCII text file in the `$ONEDB_HOME/etc` directory of the host the Connection Manager is installed on.
2. On the first line of the file, specify the NAME parameter, followed by a name for the Connection Manager. Connection Manager names must be unique in the domain of the connection units that are managed.
   Example
   For example:
   ```
   NAME my_connection_manager_1
   ```
3. Specify optional header parameters.
   Example
   For example:
   ```
   NAME my_connection_manager_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm_1.log
   ```
4. Create the body of the configuration file by specifying at least one connection unit type followed by the name of the connection unit, and then opening and closing braces.
   Example
   For example:
   ```
   NAME my_connection_manager_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

   CLUSTER my_cluster
   {
   }
   ```
5. Set the connection unit's ONEDB_SERVER parameter to the `sqlhosts` file entries for connection-unit participants.
   Example
   For example:
   ```
   NAME my_connection_manager_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

   CLUSTER my_cluster
   {
       ONEDB_SERVER group_name
   }
   ```
6. If the Connection Manager manages connection requests, specify SLA parameters, SLA names, and DBSERVER attributes.
   Example

For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

CLUSTER my_cluster
{
   ONEDB_SERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=HDR,SDS,RSS
}
```

7. Specify the FOC parameter and PRIORITY attribute. If the Connection Manager manages failover, specify the ORDER attribute, as well.

   Example

   For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

CLUSTER my_cluster
{
   ONEDB_SERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=HDR,SDS,RSS
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

8. Specify optional SLA parameter attributes.

   Example

   For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

CLUSTER my_cluster
{
   ONEDB_SERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=(HDR,SDS,RSS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

9. If the Connection Manager manages more than one connection unit, add the other connection units to the body of the configuration file.

   Example

   For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_1.log
```

```
CLUSTER my_cluster
{
   ONEDB_SERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=(HDR,SDS,RSS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_2
{
   ONEDB_SERVER group_name_2
   SLA sla_3 DBSERVERS=PRI
   SLA sla_4 DBSERVERS=(HDR,SDS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

Results

Parameters and format of the Connection Manager configuration file

The following example shows the format of the Connection Manager configuration file, and shows which parameters and attributes can be set for each connection-unit type.

```
#******************************** HEADER ********************************

NAME on page 460 connection_manager_instance_name

# Optional Parameters
MACRO on page 458 name_1=value
MACRO on page 458 name_2=value
MACRO on page 458 name_n=value_n
.
.
.
LOCAL_IP on page 456 ip_address_list
LOG on page 456 value
LOGFILE on page 457 path_and_filename
DEBUG on page 448 value
CM_TIMEOUT on page 446 seconds
EVENT_TIMEOUT on page 449 seconds
SECONDARY_EVENT_TIMEOUT on page 461 seconds
SQLHOSTS on page 475 value
SSL_LABEL on page 476 certificate_name

#********************************* BODY ********************************

# Replicate set connection-unit example

REPLSET on page 461 unit_name_1
{
   ONEDB_SERVER on page 455 sqlhosts_group_names_list
   #Optional Parameters and Attributes
   SLA on page 463 sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA on page 463 sla_name_2 DBSERVERS=value_list ...
   SLA on page 463 sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# High-availability cluster connection-unit example

CLUSTER on page 446 unit_name_2
{
   ONEDB_SERVER on page 455 sqlhosts_group_name
   FOC on page 449 ORDER=value \
       PRIORITY=value \
       TIMEOUT=value

   #Optional Parameters and Attributes
   SLA on page 463 sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA on page 463 sla_name_2 DBSERVERS=value_list ...
   SLA on page 463 sla_name_n DBSERVERS=value_list ...
.
.
.
   CMALARMPROGRAM on page 445 path_and_filename
}

# Server set connection-unit example

SERVERSET on page 462 unit_name_3
{

   ONEDB_SERVER on page 455 sqlhosts_group_names_and_standalone_servers_list

   #Optional Parameter and Attributes
   SLA on page 463 sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA on page 463 sla_name_2 DBSERVERS=value_list ...
   SLA on page 463 sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# Grid connection-unit example

GRID on page 454 unit_name_4
{
   ONEDB_SERVER on page 455 server_list

   #Optional Parameter and Attributes
   SLA on page 463 sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA on page 463 sla_name_2 DBSERVERS=value_list ...
   SLA on page 463 sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# Connection Unit n
connection_unit_type unit_name_n
{
   ONEDB_SERVER on page 455 values
   .
   .
   .
}
#***********************************************************************
```

Tip: For increased readability, break long configuration-file lines by using a backslash (\ ) line-continuation character.

The following example shows a macro definition that uses two text-file lines, but is read as a single line:

```
MACRO servers=node1,node2,node3,node4,node5,node6,node7,node8, \
              node9,node10,node11,node12,node13,node14,node15
```

CMALARMPROGRAM Connection Manager configuration parameter

The CMALARMPROGRAM parameter specifies the path and file name of a program or script to run if failover processing encounters an error.

Syntax

“ `CMALARMPROGRAM`*path_and_filename* ”

Usage

The CMALARMPROGRAM parameter is optional, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

If the Connection Manager cannot find a server capable of receiving failover, it searches for ORDER-attribute servers at increasing intervals, up to 60 seconds, for a maximum of two days. If failover processing fails after eight attempts, the Connection Manager calls the program that is specified by the CMALARMPROGRAM parameter. The first eight failover attempts take approximately one minute.

Before you can use the `cmalarmprogram` script, you must edit the file, and set the script parameters.

The `ALARMADMIN` and `ALARMPAGER` parameters determine the level of Connection Manager event alarms that are sent to specified email addresses.

Table 48. Connection Manager event-alarm levels

Level of Connection Manager event alarm	Alarm type
0 (default)	None
1	Unimportant informational alarms
2	Informational alarms
3	Alarms requiring attention
4	Emergency alarms
5	Fatal error alarms

Example

Example

In the following example, `cmalarmprogram.sh` is called if failover processing fails after eight attempts:

```
CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
```

CM_TIMEOUT Connection Manager configuration parameter

The CM_TIMEOUT parameter specifies the number of seconds that a cluster of database servers waits to receive events from the failover-arbitrator Connection Manager. If the specified period elapses with no events received by the cluster, the primary server promotes the Connection Manager with the highest priority to the role of failover arbitrator.

Syntax

“ **CM_TIMEOUT** { 60 | *seconds* } ”

Usage

The CM_TIMEOUT parameter is optional, and applies to CLUSTER connection units.

If the CM_TIMEOUT parameter is not specified, the timeout is 60 seconds.

Example

Example

In the following example, if 100 seconds elapse without the servers of a high-availability cluster receiving any events from the current failover arbiter, the primary server of the cluster promotes an active Connection Manager to the role of failover arbiter:

```
CM_TIMEOUT 100
```

CLUSTER Connection Manager configuration parameter

The CLUSTER parameter specifies that a connection unit is composed of a high-availability cluster, and specifies a name for that connection unit. A high-availability cluster is a primary server plus one or more secondary servers, which can be high-availability data replication secondary servers, shared-disk secondary servers, or remote stand-alone secondary servers.

Syntax

“ **CLUSTER** *connection_unit_name* ”

Usage

Each CLUSTER parameter value must be unique within the Connection Manager configuration file.

CLUSTER parameter values cannot use multibyte characters.

CLUSTER connection units can use the following redirection policies:

- Round-robin
- Secondary apply backlog
- Workload

Each CLUSTER connection-unit definition that references a specific high-availability cluster must have a unique PRIORITY attribute value. For example, the following high-availability cluster, composed of **server_1** and **server_2**, must have unique PRIORITY values in each definition.

my_connection_manager_1's configuration file:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_1.log

CLUSTER my_cluster
{
   ONEDB_SERVER server_1,server_2
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

my_connection_manager_2's configuration file:

```
NAME my_connection_manager_2
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm_2.log

CLUSTER my_cluster
{
   ONEDB_SERVER server_1,server_2
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Example

Example 1: Specifying a high-availability cluster as a CLUSTER connection unit

In the following example, a high-availability cluster composed of the servers in **my_server_group** is specified as the CLUSTER connection unit **my_cluster**:

```
CLUSTER my_cluster
{
   ONEDB_SERVER my_server_group
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED
```

```
        PRIORITY=1
}
```

DEBUG Connection Manager configuration parameter

The DEBUG parameter specifies whether logging of SQL and ESQL/C error messages is enabled or disabled.

Syntax

```
“ DEBUG { 0  | 1 } ”
```

Table 49. Values for the DEBUG Connection Manager configuration parameter

DEBUG parameter value	Description
`1`	Enables logging of SQL and ESQL/C error messages.
`0` (default)	Disables logging of SQL and ESQL/C error messages.

Usage

The DEBUG parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Debug messages are created in the location that is specified by the `LOGFILE` parameter in the Connection Manager configuration file. If the `LOGFILE` parameter is not specified, then the Connection Manager creates `$ONEDB_HOME/tmp/`*`connection_manager_name.pid`*`.log`.

Connection Manager debug logging is enabled in the configuration file; you cannot enable debug mode from the command line.

Example

Example

In the following example, a `my_log` is created in `$ONEDB_HOME/tmp` and logging of SQL and ESQL/C error messages is enabled.

```
LOGFILE $ONEDB_HOME/tmp/my_log
DEBUG 1
```

EVENT_TIMEOUT Connection Manager configuration parameter

The EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no primary-server events before the active-arbiter Connection Manager starts failover processing. The Connection Manager waits for primary-server events or notifications from secondary servers that the primary server is offline. A primary-server event is an indication from the primary server that the server is still functioning, such as a sent performance-statistics or administration messages, or node-changes.

Syntax

“ `EVENT_TIMEOUT` { `-1` | `60` | *seconds* } ”

Table 50. Values for the EVENT_TIMEOUT Connection Manager configuration parameter

EVENT_TIMEOUT parameter value	Description
`-1`	Connection Manager waits indefinitely
`0` to `30`	Connection Manager waits 30 seconds.
> `30`	Connection Manager waits the specified number of seconds.

Usage

The EVENT_TIMEOUT parameter is optional, and applies to CLUSTER connection units.

If EVENT_TIMEOUT is not specified in the Connection Manager configuration file, the Connection Manager waits 60 seconds for primary-server events or notifications from secondary servers that the primary server is offline before it starts failover processing.

Example

Example

In the following example, the active-arbiter Connection Manager begins failover processing after 200 seconds elapse with no primary-server events.

```
EVENT_TIMEOUT 200
```

FOC Connection Manager configuration parameter

The FOC parameter and attributes specify the failover configuration for high-availability clusters, and specify the priority of the connection between the Connection Manager and the primary server.

Syntax

“ **FOC** { **ORDER** = **ENABLED** | **ORDER** = **DISABLED** } ”

“ **PRIORITY** = *value* ”

“ [**TIMEOUT** = *seconds*] ”

The FOC parameter is required by CLUSTER connection units.

Attributes of the FOC Connection Manager configuration parameter

Table 51. Attributes of the FOC parameter

Attribute of the FOC parameter	Description
`ORDER`	Disables automatic failover or specifies that the value of the primary server's HA_FOC_ORDER configuration parameter is used for automatic failover. If the ORDER attribute is not specified, the value of the primary server's HA_FOC_ORDER configuration parameter is used for automatic failover. If the ORDER attribute is not specified, and the primary server's HA_FOC_ORDER configuration parameter is not set, the failover order is SDS, HDR, and then RSS.
`PRIORITY`	Specifies the priority of connections between Connection Managers and the primary server of a cluster. The value must be a positive integer and unique among all Connection Manager CLUSTER units specified for a high-availability cluster. The lower the number, the higher the priority. For CLUSTER connection units, a PRIORITY value must be specified, even if ORDER is set to DISABLED.
`TIMEOUT`	Specifies the number of additional seconds the Connection Manager waits for primary-server events before the Connection Manager begins failover processing. The TIMEOUT attribute value applies after the EVENT_TIMEOUT parameter 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 can begin.

Table 51. Attributes of the FOC parameter (continued)

Attribute of the FOC parameter	Description
	If the TIMEOUT attribute is not specified, failover begins immediately after the amount of time that is specified by the EVENT_TIMEOUT parameter value is exceeded.

Values for the ORDER attribute of the FOC Connection Manager configuration parameter

Table 52. Values for the ORDER attribute of the FOC Connection Manager configuration parameter.

ORDER attribute value	Description
`ENABLED`	Specifies that the Connection Manager can perform automatic failover. The Connection Manager uses the value of the primary server's `onconfig` file HA_FOC_ORDER configuration parameter to determine failover sequence. If the ORDER attribute is set to ENABLED, but the primary server's HA_FOC_ORDER configuration parameter is not specified, the failover order is SDS, HDR, and then RSS.
`DISABLED`	Disables automatic failover processing by the Connection Manager.

ORDER Usage

The ORDER attribute specifies if automatic failover is enabled. If failover cannot complete, the Connection Manager reattempts failover at increasing intervals, up to 60 seconds, for a maximum of two days.

To modify the number of seconds that must elapse before the active-arbiter Connection Manager starts failover processing, set the EVENT_TIMEOUT parameter.

If you configure failover, you can set the CMALARMPROGRAM parameter to specify a program or script to run if failover processing cannot complete after eight tries.

Important: If automatic failover is enabled, and failover processing begins, you must not manually restart the failed primary server until failover processing completes.

PRIORITY Usage

If you install Connection Managers on application-server hosts, you can use the PRIORITY attribute to prioritize the connections between the application servers and the primary server of a cluster.

If the network connection between a specific Connection Manager and primary database server fails, PRIORITY determines whether the Connection Manager starts failover. If failover would promote a database server that cannot communicate with an active, higher-priority Connection Manager, then failover does not occur. If failover would promote a database server that cannot communicate with an active, lower-priority Connection Manager, failover can occur.

Example

Example 1: Configuring multiple Connection Managers for failover

In the following example, the HA_FOC_ORDER configuration parameter in the **my_primary_server_1** `onconfig` file is set:

```
HA_FOC_ORDER SDS,HDR,RSS
```

Three Connection Managers are installed and have the following configuration files:

`cm_configuration_file_1`

```
NAME my_connection_manager_1

CLUSTER my_cluster_1
{
   ONEDB_SERVER my_server_group_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=1
   CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
}
```

`cm_configuration_file_2`

```
NAME my_connection_manager_2

CLUSTER my_cluster_1
{
   ONEDB_SERVER my_server_group_1
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
}
```

`cm_configuration_file_3`

```
NAME my_connection_manager_3

CLUSTER my_cluster_1
{
   ONEDB_SERVER my_server_group_1
   SLA sla_3 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=3
   CMALARMPROGRAM $ONEDB_HOME/etc/cmalarmprogram.sh
}
```

When the connection managers are initialized, they each search their hosts's `sqlhosts` file for a **my_server_group_1** entry and connect with the servers that are in the group. The value of the HA_FOC_ORDER configuration parameter in the `onconfig` file of the primary server in **my_server_group_1** is set for each of the Connection Manager, so that all the Connection Managers have a consistent failover policy. The value of the primary server's HA_FOC_ORDER configuration parameter replaces the values of the HA_FOC_ORDER configuration parameters in the `onconfig` files of all secondary servers in the cluster. This process maintains failover-order consistency if the primary server fails and then a Connection Manager restarts.

If a Connection Manager detects that the primary server failed, it first attempts to convert the most suitable SD secondary server to the primary server. If no SD secondary server is available, the Connection Manager attempts to convert the HDR secondary server into the primary server. If the HDR secondary server is not available, the Connection Manager attempts to convert the most suitable RS secondary server to the primary server.

If failover processing fails after eight attempts, the Connection Manager calls `$ONEDB_HOME/etc/cmalarmprogram.sh`.

Example

Example 2: Configuring multiple Connection Managers to prioritize the connections between application servers and the primary server of a cluster

In the following example, the HA_FOC_ORDER configuration parameter in the **my_primary_server_2** `onconfig` file is set:

```
HA_FOC_ORDER HDR,RSS
```

The two Connection Managers, **my_important_app_cm** and **my_non_essential_app_cm**, are installed and configured:

`cm_configuration_file_4`

```
NAME important_app_cm
LOCAL_IP 192.0.2.0, 192.0.2.256

CLUSTER my_cluster
{
   ONEDB_SERVER my_server_group_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

`cm_configuration_file_5`

```
NAME non_essential_app_cm
LOCAL_IP 192.0.2.257, 192.0.2.258

CLUSTER my_cluster
{
   ONEDB_SERVER my_server_group_1,
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

- The network state is:
- **important_app_cm** can connect to the primary server and RS secondary server, but cannot connect to the HDR secondary server.
- **non_essential_app_cm** can connect to the primary server and the HDR server, but cannot connect to the RS secondary server.

If the network between **non_essential_app_cm** and **my_primary_server** goes down, **non_essential_app_cm** attempts failover to the HDR secondary server. Because **important_app_cm** cannot connect to the HDR secondary server, and has higher priority than **non_essential_app_cm**, the failover attempt is blocked.

If, instead, the network between **important_app_cm** and **my_primary_server** goes down, **important_app_cm** attempts failover to the HDR secondary server. Because **important_app_cm** cannot connect to the HDR secondary server, it then attempts failover to the RS secondary server. **non_essential_app_cm** cannot connect to the RS secondary server, but it has a lower priority than **important_app_cm**, so failover occurs.

In both situations, the connection between **important_app_cm** and the primary server is prioritized over the connection between **non_essential_app_cm** and the primary server. Because the Connection Managers are on the application-server hosts, the connections between the application servers and the primary server are, essentially, prioritized.

GRID Connection Manager configuration parameter

The GRID parameter specifies that a connection unit is an Enterprise Replication grid, and specifies the name of the grid. A grid is a set of replication servers that you can administer as a unit.

Syntax

“ **GRID***unit_name* ”

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

GRID connection units can use the following redirection policies:

- Apply failure
- Round-robin
- Transaction latency
- Workload

Grid names can use multibyte characters.

Example

Example

In the following example, the grid connection unit named **my_grid** is defined:

```
GRID my_grid
{
   ONEDB_SERVER my_server_group_1,my_server_group_2
   SLA sla_1 DBSERVERS=ANY
}
```

ONEDB_SERVER Connection Manager configuration parameter

The ONEDB_SERVER parameter specifies database servers or Enterprise Replication nodes that the Connection Manager connects to after it starts. The values of the ONEDB_SERVER parameter are also used for providing database server information for service-level agreements when a Connection Manager's SQLHOSTS parameter is set to `REMOTE` or `LOCAL +REMOTE`.

Syntax

CLUSTER connection unit

“ **ONEDB_SERVER** { *server_name* | *sqlhosts_group_name* } ”

GRID connection unit

“ **ONEDB_SERVER** { | *sqlhosts_group_name* } ”

REPLSET connection unit

“ **ONEDB_SERVER** { | *sqlhosts_group_name* } ”

SERVERSET connection unit

“ **ONEDB_SERVER** { | *sqlhosts_group_name* | *stand_alone_server_name* } ”

Usage

The ONEDB_SERVER parameter is required, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

For Enterprise Replication domains, list group names for the nodes that receive client requests from the Connection Manager.

For server-set replication domains, list group names for the nodes that receive client requests from the Connection Manager. List standalone server names, as well.

For high-availability clusters, list the group name for the cluster, or list the names of database servers.

Example

Example

In the following example, the Connection Manager connects to all database servers that are members of the **my_server_group** group in the Connection Manager `sqlhosts` file.

```
NAME my_connection_manager


CLUSTER my_cluster
{
   ONEDB_SERVER my_server_group
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

LOCAL_IP Connection Manager configuration parameter

The LOCAL_IP parameter specifies IP addresses to monitor on the computer that is running the Connection Manager. The LOCAL_IP parameter is used with the FOC parameter's PRIORITY attribute to determine if database-failover occurs during a partial network failure.

Syntax

" **LOCAL_IP** { | *ip_address* } "

Usage

The LOCAL_IP parameter is optional, and applies to CLUSTER connection units.

You must list the IP address of each network interface card to be monitored.

Example

Example

In the following example, the Connection Manager monitors the network connection between its host and the primary server of a cluster through the network interface cards that have IP addresses of 192.0.2.0 and 192.0.2.1:

```
LOCAL_IP 192.0.2.0,192.0.2.1
```

LOG Connection Manager configuration parameter

The LOG parameter specifies logging for Connection Manager modes.

Syntax

```
“LOG{0 | 1 | 2 | 3}”
```

Table 53. Values for the LOG Connection Manager configuration parameter

LOG parameter value	Description
`0` (default)	Specifies that logging is turned off.
`1`	The Connection Manager logs proxy-mode and redirect-mode SLA information.
`2`	The Connection Manager logs proxy-mode SLA information only. Data send-and-receive activities between clients and the Connection Manager is logged.
`3`	The Connection Manager logs proxy-mode SLA information only. Data content between clients and the Connection Manager is logged.

Usage

The LOG parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Set the LOGFILE parameter to specify where log files are stored. If the LOGFILE parameter is not set, the Connection Manager creates a log file in the `$ONEDB_HOME/tmp` directory, with a name of *`connection_manager_name`*`.`*`process_ID`*`.log`.

Example

Example

In the following example, the Connection Manager logs proxy-mode and redirect-mode SLA information to `$ONEDB_HOME/tmp/my_cm.log`.

```
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm.log
```

LOGFILE Connection Manager configuration parameter

The LOGFILE parameter specifies the name and location of the Connection Manager log file.

Syntax

" **LOGFILE***path_and_filename* "

Usage

The LOGFILE parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The path and file name of the log file display when the Connection Manager is started, and the log file is continuously updated with status information while the Connection Manager is running. If multiple Connection Managers are installed on the same host, specify a different path and file name for each Connection Manager.

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 the LOGFILE parameter is not set, the Connection Manager creates a log file in the `$ONEDB_HOME/tmp` directory, with a name of *`connection_manager_name.process_ID`*`.log`.

Example

Example

In the following example, the Connection Manager logs proxy-mode SLA information about data send-and-receive activities between clients and the Connection Manager. The information is logged to `$ONEDB_HOME/tmp/my_cm.log`.

```
LOG 2
LOGFILE $ONEDB_HOME/tmp/my_cm.log
```

MACRO Connection Manager configuration parameter

The MACRO parameter specifies the name of a macro and a value that can be reused in the Connection Manager configuration file.

Macro definition format

" **MACRO***name=value* "

Macro use format

“ ${*macro_name*} ”

Usage

The MACRO parameter is optional, and applies to the Connection Manager configuration file.

A macro can contain spaces, but not line breaks.

The MACRO parameter can be set multiple times to create multiple macros.

After a macro is defined, it can be used in other macros. For example:

```
MACRO WA=wa_server_1,wa_server_2,wa_server_3,wa_server_4
MACRO OR=or_server_1,or_server_2,or_server_3,or_server_4
MACRO ID=id_server_1,id_server_2,id_server_3,id_server_4
MACRO PNW=${WA),${OR),${ID)
```

Example

Example 1: Using a macro in a service-level agreement

In the following example, the macro **CA** contains the names of eight servers.

```
NAME my_connection_manager_1
MACRO CA=ca_server_1,ca_server_2,ca_server_3,ca_server_4, \
         ca_server_5,ca_server_6,ca_server_7,ca_server_8

CLUSTER my_cluster_1
{
   ONEDB_SERVER group_name_1
   SLA sla_1 DBSERVERS=${CA}
   FOC ORDER=ENABLED PRIORITY=1
}
```

The macro expands in the following way:

```
{
   ONEDB_SERVER group_name_1
   SLA sla_1 DBSERVERS=ca_server_1,ca_server_2,ca_server_3,ca_server_4, \
         ca_server_5,ca_server_6,ca_server_7,ca_server_8
   FOC ORDER=ENABLED PRIORITY=1
}
```

Example

Example 2: Using multiple macros in service-level agreements

In the following example, the macros **WA**, **OR**, and **ID** each contain the names of servers. Macro **ID** also contains parentheses to create a redirection-policy group.

```
NAME my_connection_manager_2
MACRO WA=wa_server_1,wa_server_2,wa_server_3
MACRO OR=or_server_1,or_server_2,or_server_3
MACRO ID=(id_server_1,id_server_2,id_server_3)

CLUSTER my_cluster_2
{
   ONEDB_SERVER group_name_2
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=${WA},${OR}
   SLA sla_3 DBSERVERS=${ID} POLICY=ROUNDROBIN
   FOC ORDER=ENABLED PRIORITY=1
}
```

The macros expand in the following ways:

```
{
   ONEDB_SERVER group_name_2
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=wa_server_1,wa_server_2,wa_server_3,or_server_1,or_server_2,or_server_3
   SLA sla_3 DBSERVERS=(id_server_1,id_server_2,id_server_3) POLICY=ROUNDROBIN
   FOC ORDER=ENABLED PRIORITY=1
}
```

NAME Connection Manager configuration parameter

The NAME parameter specifies the name of the Connection Manager instance.

Syntax

“ **NAME***connection_manager_name* ”

Usage

The NAME parameter is required, and applies to the Connection Manager instance.

The name of the Connection Manager instance is necessary for monitoring, shutting down, or reloading the Connection Manager.

Connection Manager names must be unique in the domain of the connection units that are managed.

Example

Example

In the following example, a Connection Manager instance is named **my_connection_manager**.

```
NAME my_connection_manager
```

REPLSET Connection Manager configuration parameter

The REPLSET parameter specifies that a connection unit is an Enterprise Replication replicate set, and specifies the name of the replicate set. A replicate set combines several replicates to form a set that can be administered together as a unit.

Syntax

“ **REPLSET***unit_name* ”

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

REPLSET connection units can use the following redirection policies:

- Apply failure
- Round-robin
- Transaction latency
- Workload

Replicate set names can use multibyte characters.

Example

Example

In the following example, the replicate-set connection unit named **my_replicate_set** is defined:

```
REPLSET my_replicate_set
{
   ONEDB_SERVER my_group_1,my_group_2
   SLA sla_1 DBSERVERS=ANY
}
```

SECONDARY_EVENT_TIMEOUT Connection Manager configuration parameter

The SECONDARY_EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no secondary-server events before the Connection Manager disconnects from a secondary server. A secondary-server event is an indication from a secondary server that the server is still functioning, such as a sent performance-statistics or administration messages.

Syntax

“ **SECONDARY_EVENT_TIMEOUT** { **-1** | **60** | *seconds* } ”

Table 54. Values for the SECONDARY_EVENT_TIMEOUT Connection Manager configuration parameter

SECONDARY_EVENT_TIMEOUT parameter value	Description
`-1`	The Connection Manager waits indefinitely
`0` to `30`	The Connection Manager waits 30 seconds.
> `30`	The Connection Manager waits the specified number of seconds.

Usage

The SECONDARY_EVENT_TIMEOUT parameter is optional, and applies CLUSTER connection units.

If the SECONDARY_EVENT_TIMEOUT parameter is not specified in the Connection Manager configuration file, the Connection Manager waits 60 seconds for secondary-server events before it disconnects from the server.

Example

Example

In the following example, the Connection Manager disconnects from a secondary server after 300 seconds elapse with no secondary-server events.

```
SECONDARY_EVENT_TIMEOUT 300
```

SERVERSET Connection Manager configuration parameter

The SERVERSET parameter specifies that a connection unit is a server set, and specifies the name of the server set. A server set contains unrelated, standard servers that do not use replication or failover.

Syntax

" **SERVERSET***unit_name* "

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

SERVERLSET connection units can use the following redirection policies

- Round-robin
- Workload

Server set names cannot use multibyte characters.

Example

Example

In the following example, the server-set connection unit named **my_server_set** is defined:

```
SERVERSET my_server_set
{
   ONEDB_SERVER my_group_1,my_group_2
   SLA sla_1 DBSERVERS=ANY
}
```

SLA Connection Manager configuration parameter

The SLA parameter defines service-level agreements that direct client requests to database servers.

Syntax

SLA*sla_name* { <CLUSTER SLA syntax> <GRID or REPLSET SLA syntax> <SERVERSET SLA syntax> } [**MODE=** { **REDIRECT** | **PROXY** }] [**USEALIASES=** { **ON** | **OFF** }] [**WORKERS=** { 4 | *Number_of_threads* }] [**HOST=** { *host_name* | *ip_address* }] [**NETTYPE** = { *drsoctcp* | *onsoctcp* }] [**SERVICE=** { *port_number* | *service_name* }] [**SQLHOSTSOPT="** { | *option* } **"**]

CLUSTER SLA syntax fragment

" **DBSERVERS=** { **ANY** [**POLICY=** { **ROUNDROBIN** | **WORKLOAD** [**+SECAPPLYBACKLOG:** *pages*] }] | { | *group* | **HDR** | **PRI** | **primary** | **RSS** | **SDS** | *server* } ▯ | { | { | *group* | **HDR** | **PRI** | **primary** | **RSS** | **SDS** | *server* } | ({ | *group* | **HDR** | **PRI** | **primary** | **RSS** | **SDS** | *server* }) } [**POLICY=** { **ROUNDROBIN** | **WORKLOAD** [**+SECAPPLYBACKLOG:** *pages*] }] } "

GRID or REPLSET SLA syntax fragment

" **DBSERVERS=** { **ANY** [**POLICY=** { { | [*weight**] **FAILURE** | [*weight**] **LATENCY** | [*weight**] **WORKLOAD** } | **ROUNDROBIN** }] | { | *group* } | ▯ { | { | *group* } | ({ | *group* }) } [**POLICY=** { { | [*weight**] **FAILURE** | [*weight**] **LATENCY** | [*weight**] **WORKLOAD** } | **ROUNDROBIN** }] } "

SERVERSET SLA syntax fragment

" **DBSERVERS=** { **ANY** [**POLICY=** { **WORKLOAD** | **ROUNDROBIN** }] | { | *group* | *server* } | ▯ { | { | *group* | *server* } | ({ | *group* | *server* }) } [**POLICY=** { **WORKLOAD** | **ROUNDROBIN** }] } "

(explicit id unique_823_Connect_42_asdf) unique_823_Connect_42_asdf

SLA parameter attributes

Table 55. The attributes of the SLA Connection Manager configuration parameter

Attribute name	Description
`DBSERVERS`	Specifies servers, server aliases, server groups, or server types for directing connection requests. Use the ANY keyword, the SDS or RSS cluster keywords, or enclose a group of values in parentheses to enable a redirection policy for that group.
`HOST`	Specifies a database server's host. The value in the SLA is used, rather than the value in the Connection Manager's host `sqlhosts` file.
`MODE`	Specifies whether connection requests go through the Connection Manager or if the Connection Manager provides connection information to the source of a connection request. The default value is REDIRECT.
`NETTYPE`	Specifies the network protocol of a database server. The value in the SLA is used, rather than the value in the Connection Manager's host `sqlhosts` file.
`POLICY`	Specifies how the Connection Manager redirects client connection requests to the servers specified in the DBSERVER attribute. Redirection policy applies to the ANY keyword, the SDS and RSS cluster keywords, and to a group of values that are enclosed in parentheses. Parentheses within parentheses are ignored. The default value is WORKLOAD. Workload, apply-failure, and transaction-latency policies can be given relative weights.
`SERVICE`	Specifies a database server's port number or service name. The value in the SLA is used, rather than the value in the Connection Manager's host `sqlhosts` file.
`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 Connection Manager's host `sqlhosts` file.
`USEALIASES`	Specifies whether the Connection Manager can redirect client connection requests to database server aliases specified by the DBSERVERALIASES configuration parameter. The default value is ON.
`WORKERS`	Specifies the number of worker threads that are 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 worker threads.

Table 55. The attributes of the SLA Connection Manager configuration parameter (continued)

Attribute name	Description
	The default value is 4.

DBSERVERS attribute values

Table 56. Values of the DBSERVERS attribute.

Attribute value	Value
`ANY`	Specifies that connection requests can be sent to any available database server in the specified cluster, grid, or replicate set. For a SERVERSET connection unit, `ANY` specifies that connection requests can be sent to any available database server specified by the SERVERSET connection-unit's ONEDB_SERVER parameter. You do not need to enclose ANY in parentheses to apply a redirection policy to it. If no redirection policy is specified, ANY uses the workload redirection policy.
group	Specifies a group entry in the Connection Manager's host `sqlhosts` file. Connection requests can be sent to the members of the group.
`HDR`	Is a cluster keyword that specifies that connection requests can be sent to the high-availability data replication server. HDR is supported only by CLUSTER connection units.
`PRI` or `PRIMARY`	Is a cluster keyword that specifies that connection requests can be sent to the primary database server. PRI and PRIMARY are supported only by CLUSTER connection units.
`RSS`	Is a cluster keyword that specifies that connection requests can be sent to remote standalone secondary servers. RSS is supported only by CLUSTER connection units. You do not need to enclose RSS in parentheses to apply a redirection policy to the servers it specifies. If no redirection policy is specified, RSS uses the workload redirection policy.
`SDS`	Is a cluster keyword that specifies that connection requests can be sent to shared-disk secondary servers. SDS is supported only by CLUSTER connection units. You do not need to enclose SDS in parentheses to apply a redirection policy to the servers it specifies. If no redirection policy is specified, SDS uses the workload redirection policy.
server	Specifies server or alias entry in the Connection Manager's host `sqlhosts` file. Connection requests can be sent to the server.

MODE attribute values

Table 57. Values of the MODE attribute.

Attribute value	Value
`PROXY`	Specifies that the Connection Manager acts as a proxy server for client connections. Use proxy mode for the following cases: • A firewall is preventing a client application from connecting to database servers. • You do not want to recompile applications are compiled with Data Server Driver for JDBC and SQLJ version 3.5.1 or before, or with HCL OneDB™ Client Software Development Kit (Client SDK) 3.00 or before. Because a proxy-server Connection Manager handles all client/server communication, configure multiple Connection Manager instances, to avoid a Connection Manager becoming a single point of failure. **Note:** For proxy mode, you must set your operating system to allow the maximum number of file descriptors. For example, use the ulimit command on UNIX™ operating systems.
`REDIRECT` (default)	Specifies that client connections use redirect mode, which configures the Connection Manager to return the appropriate database server name, IP address, and port number to the requesting client application. The client application then uses the returned IP address and port number to connect to the specified database server.

POLICY attribute values

Table 58. Values of the POLICY attribute.

Attribute value	Value
`FAILURE`	Specifies that connection requests are directed or proxied to the replication server with the fewest apply failures. The apply-failure policy is supported by the following connection units: • REPLSET • GRID To use the apply-failure policy, you must enable quality of data (QOD) monitoring by running the cdr define qod and cdr start qod

Table 58. Values of the POLICY attribute. (continued)

Attribute value	Value
	commands. To use the apply-failure policy for a grid, the grid must have a replication-enabled table.
`LATENCY`	Specifies that connection requests are directed or proxied to the replication server with the lowest transaction latency. The transaction-latency policy is supported by the following connection units: • REPLSET • GRID The attribute value does not indicate a specific transaction latency period; the Connection Manager uses a formula with relative values to decide where to redirect client connection requests. To use the transaction-latency policy, you must enable quality of data monitoring by running the cdr define qod and cdr start qod commands. To use the transaction-latency policy for a grid, the grid must have a replication-enabled table.

`ROUNDROBIN`	Specifies that connection requests are directed or proxied in a repeating, ordered fashion (round-robin) to a group of servers.
	
	If you use a round-robin policy, the DBSERVERS attribute values are used to create round-robin groups. Servers that are specified more than one time in a DBSERVERS-attribute group value are treated as single participants in a round-robin group. For example, if a SLA has the following definition:
	
	`SLA sla_1 DBSERVERS=(server_1,server_3,server_1,server_2) \`
	`          POLICY=ROUNDROBIN`
	
	The round-robin group participants are:
	
	• **server_1**
	• **server_2**
	• **server_3**
	
	The round-robin policy is supported by the following connection units:
	
	• CLUSTER
	• REPLSET
	• GRID
	• SERVERSET

Table 58. Values of the POLICY attribute. (continued)

Attribute value	Value
	The round-robin policy is supported by the following software: • All HCL OneDB™ server versions. • Connection Managers from HCL OneDB™ Client Software Development Kit (Client SDK)

`SECAPPLYBACKLOG:`*`number_of_pages`*	Specifies that if a secondary server's apply backlog exceeds *number_of_pages*, the Connection Manager does not redirect or proxy new connections to server. For example:
	`SLA sla_1 DBSERVERS=(server_1,server_2,server_3) \`
	`        POLICY=SECAPPLYBACKLOG:500`
	The Connection Manager sends connection requests to whichever of **server_1**, **server_2**, or **server_3** has an apply backlog below 500 pages and the lowest workload.
	To view the apply backlogs of all servers in a cluster, run the onstat -g cluster command.
	To view the apply backlog for a specific secondary server, run one of the following commands:
	• onstat -g dri
	• onstat -g sds
	• onstat -g rss
	The apply-backlog policy is supported by CLUSTER connection units.
	The apply-backlog policy is supported by the following software:
	• HCL OneDB™ server versions 11.70.xC8 and later, and 12.10.xC2 and later.
	• Connection Managers from HCL OneDB™ Client Software Development Kit (Client SDK) 3.70.xC8 and later, and 4.10xC2 and later.

`WORKLOAD` (default)	Specifies that connection requests are directed or proxied to the database server with the lowest workload. Workload calculations are based on the number of virtual processors a server has and the number of threads in the server's ready queue. The WORKLOAD policy is supported by the following connection units:

Table 58. Values of the POLICY attribute. (continued)

Attribute value	Value
	• CLUSTER • GRID • REPLSET • SERVERSET

USEALIASES attribute values

Table 59. Values of the USEALIASES attribute.

Attribute value	Value
`ON` (default)	Specifies that the Connection Manager can direct client connection requests to server aliases specified by a database server's DBSERVERALIASES configuration parameter.
`OFF`	Specifies that the Connection Manager cannot direct client connection requests to server aliases specified by a database server's DBSERVERALIASES configuration parameter.

HOST attribute values

Table 60. Values of the HOST attribute.

Attribute value	Value
host_name	Specifies a host name or host alias for a database server.
ip_address	Specifies a TCP/IP address for a database server.

NETTYPE attribute values

Table 61. Values of the NETTYPE attribute.

Attribute value	Value
`drsoctcp`	Specifies TCP/IP protocol for Distributed Relational Database Architecture™
`onsoctcp`	Specifies sockets with TCP/IP protocol.

SERVICE attribute values

Table 62. Values of the SERVICE attribute.

Attribute value	Value
port_number	Specifies a port number.
service_name	Specifies a service name.

Usage

The SLA parameter is optional, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Client applications use the SLA name to connect to the database servers or database-server types that are specified by the value of the DBSERVERS attribute. 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.

Example

Example 1: Connection request redirection from a service-level agreement

The following example shows a simple Connection Manager configuration. The configuration specifies a single SLA.

```
NAME my_connection_manager_1

CLUSTER my_cluster_1
{
   ONEDB_SERVER my_server_group_1
   SLA sla_1 DBSERVERS=SDS,HDR,PRI
   FOC ORDER=ENABLED PRIORITY=1
}
```

`CONNECT TO @sla_1` connection requests as are directed in the following way:

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

Example 2: Defining multiple service-level agreements

The following example shows a simple Connection Manager configuration. The configuration specifies two SLAs.

```
NAME my_connection_manager_2

CLUSTER my_cluster_2
{
   ONEDB_SERVER my_server_group_2
   SLA sla_1 DBSERVERS=server_1
   SLA sla_2 DBSERVERS=server_2,server_3
   FOC ORDER=ENABLED PRIORITY=1
}
```

This example configures the Connection Manager for a high-availability cluster and defines two SLAs:

- `CONNECT TO @sla_1` connection requests are directed to **server_1**.
- `CONNECT TO @sla_2` connection requests are directed to **server_2**. If **server_2** is not available, connection requests are directed to **server_3**.

Example

Example 3: Redirection policies in service-level agreements

The following example shows a Connection Manager configuration that uses a workload-balancing redirection policy. The configuration specifies a single SLA.

```
NAME my_connection_manager_3

CLUSTER my_cluster_3
{
   ONEDB_SERVER my_server_group_3
   SLA sla_1 DBSERVERS=(server_1,server_2) \
             POLICY=WORKLOAD
   SLA sla_2 DBSERVERS=(server_3,server_4,server_5) \
             POLICY=ROUNDROBIN
   SLA sla_3 DBSERVERS=(server_6,server_7,server_8) \
             POLICY=ROUNDROBIN+SECAPPLYBACKLOG:400
   FOC ORDER=ENABLED PRIORITY=1
}
```

- `CONNECT TO @sla_1` connection requests are directed to whichever of **server_1** and **server_2** has the lowest workload. WORKLOAD is the default redirection policy, so specifying `POLICY=WORKLOAD` in the SLA is not required.
- `CONNECT TO @sla_2` connection requests are directed round-robin to **server_3**, **server_4** and **server_5**.
- `CONNECT TO @sla_3` connection requests are directed round-robin to **server_6**, **server_7** and **server_8**. If a server's apply backlog is 400 pages or greater, that server is ignored in the round-robin order and does not receive connection requests until its apply backlog falls below 400 pages.

Example

Example 4: Proxy and redirect mode in service-level agreements

Example

In redirect mode, the Connection Manager responds to client redirection requests by returning a specified database server's IP address and port number to the client application. The client application then uses the IP address and port number to connect to the database server. In proxy mode, the Connection Manager acts as a proxy server, and client requests are routed through the Connection Manager. Redirect mode is the default if no SLA mode is specified.

```
NAME my_connection_manager_4

CLUSTER my_cluster_4
{
   ONEDB_SERVER my_server_group_4
   SLA sla_1 DBSERVERS=ANY \
             MODE=REDIRECT #Default value, so is not required for the SLA definition
   SLA sla_2 DBSERVERS=ANY \
             MODE=PROXY
   FOC ORDER=ENABLED PRIORITY=1
}
```

- `CONNECT TO @sla_1` connection requests result in the Connection Manager returning the IP address and port number for a cluster server to the client application.
- `CONNECT TO @sla_2` connection requests are directed through the Connection Manager to a cluster server.

Example

Example 5: Adjusting timeout values for the Connection Manager and cluster servers

The following example shows a Connection Manager configuration where default timeout values are changed:

```
NAME my_connection_manager_5
CM_TIMEOUT 300
EVENT_TIMEOUT 45
SECONDARY_EVENT_TIMEOUT 50

CLUSTER my_cluster_5
{
   ONEDB_SERVER my_server_group_5
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED PRIORITY=1
}
```

- If a cluster does not receive any events from the Connection Manager within 300 seconds, the primary server of the cluster promotes the next available Connection Manager to the role of failover arbitrator.
- If the Connection Manager does not receive any events from the primary server within 45 seconds, the Connection Manager begins failover processing, and attempts to promote a secondary server to the primary server.
- If the Connection Manager does not receive any events from a secondary server within 50 seconds, the Connection Manager disconnects from the secondary server.

Example

Example 6: Macros and workload balancing in service-level agreements

The following example shows a Connection Manager configuration that uses defined macros in SLAs.

```
NAME my_connection_manager_6
MACRO CA=ca_server_1,ca_server_2,ca_server_3
MACRO NY=ny_server_1,ny_server_2,ny_server_3

REPLSET my_replicate_set_1
{
   ONEDB_SERVER my_er_group_1,my_er_group_2
   SLA sla_1 DBSERVERS=${CA}
   SLA sla_2 DBSERVERS=(${NY}) \
             POLICY=ROUNDROBIN
}
```

In this example, two macros are defined:

- `CA`, which is composed of **ca_server_1**, **ca_server_2**, and **ca_server_3**.
- `NY`, which is composed of **ny_server_1**, **ny_server_2**, and **ny_server_3**.

The Connection Manager redirects client connection requests as follows:

- `CONNECT TO @sla_1` connection requests are directed to **ca_server_1**. If **ca_server_1** is unavailable, connection requests are directed to **ca_server_2**. If **ca_server_2** is also unavailable, connection requests are directed to **ca_server_3**.
- `CONNECT TO @sla_2` connection requests are directed round-robin to **ny_server_1**, **ny_server_2**, and **ny_server_3**.

Example

Example 7: Quality of data redirection policies in service-level agreements

The following example shows a Connection Manager configuration that uses transaction-latency and apply-failure redirection policies to direct connection requests in a grid. Quality-of-data (QOD) monitoring is turned on with the cdr define qod and cdr start qod commands.

```
NAME my_connection_manager_7

GRID my_grid_1
{
   ONEDB_SERVER my_server_group_1,my_server_group_2,my_server_group_3
   SLA sla_1 DBSERVERS=ANY \
             POLICY=LATENCY
   SLA sla_2 DBSERVERS=ANY \
             POLICY=FAILURE
   SLA sla_3 DBSERVERS=ANY \
             POLICY=2*Failure+LATENCY
}
```

- `CONNECT TO @sla_1` connection requests are directed to the server with the lowest transaction latency.
- `CONNECT TO @sla_2` connection requests are directed to the server with the lowest number of apply failures.
- `CONNECT TO @sla_3` connection requests are directed to the server with the lowest number of apply failures and lowest transaction latency. The smallest apply-failure count is twice as important as low transaction latency in the Connection Manager's calculations.

Example

Example 8: sqlhosts connectivity information in service-level agreements

The following example shows a Connection Manager configuration that uses attribute values instead of the values in its host `sqlhosts` file for directing connection requests.

```
NAME my_connection_manager_8

SERVERSET my_server_set
{
   ONEDB_SERVER server_1,server_2,server_3
   SLA sla_1 DBSERVERS=server_1 \
             NETTYPE=onsoctcp
             HOST=host_1 \
             SERVICE=port_1 \
   SLA sla_2 DBSERVERS=server_2 \
             NETTYPE=onsoctcp
             HOST=host_2 \
             SERVICE=port_2 \
}
```

The Connection Manager uses the values of the HOST, SERVICE, and NETTYPE attributes, rather than the values in its host `sqlhosts` file for directing connection requests.

Example

Example 9: Controlling connection-requests with aliases

The following example shows a Connection Manager configuration that specifies which database-server aliases SLAs can use.

The `onconfig` file for **server_1** has the following parameter setting:

```
DBSERVERALIASES server_1_alias_1,server_1_alias_2
```

The `sqlhosts` file that **server_1** uses has the following entries:

```
#dbservername      nettype   hostname   servicename  options
my_group_9         group     -          -            e=server_1_alias_2
server_1           onsoctcp  my_host_1  my_port_1    g=my_group_9
server_1_alias_1   onsoctcp  my_host_1  my_port_2    g=my_group_9
server_1_alias_2   onsoctcp  my_host_1  my_port_3    g=my_group_9
```

The Connection Manager configuration file for **server_1** has the following entries:

```
NAME my_connection_manager_9

SERVERSET my_server_set_2
{
   ONEDB_SERVER my_group_9
   SLA sla_1 DBSERVERS=server_1
   SLA sla_2 DBSERVERS=server_1 \
             USEALIASES=OFF
   SLA sla_3 DBSERVERS=server_1_alias_1
   SLA sla_4 DBSERVERS=server_1_alias_1 \
             USEALIASES=OFF
}
```

The Connection Manager directs client requests in the following ways:

- `CONNECT TO @sla_1` and `CONNECT TO @sla_3` requests can be directed to **server_1**, through **my_port_1**, **my_port_2**, or **my_port_3**.
- `CONNECT TO @sla_2` requests are directed to **server_1** through **my_port_1** only.
- `CONNECT TO @sla_4` requests are directed to **server_1** through **my_port_2** only.

SQLHOSTS Connection Manager configuration parameter

The SQLHOSTS parameter specifies where a Connection Manager can search for database servers that are specified by the ONEDB_SERVER parameter and DBSERVERS attribute.

Syntax

```
"SQLHOSTS={LOCAL | REMOTE | LOCAL+REMOTE}"
```

Table 63. Values for the SQLHOSTS Connection Manager configuration parameter

SQLHOSTS parameter value	Description
`LOCAL`	The Connection Manager searches the local `sqlhosts` file for requested database server instances.
`REMOTE`	The Connection Manager searches for requested database server instances in remote `sqlhosts` files. The Connection Manager searches the `sqlhosts` files of database servers that are specified by a connection-unit's ONEDB_SERVER parameter.
`LOCAL+REMOTE` (default)	The Connection Manager searches the local `sqlhosts` file for requested database server instances. If a database server cannot be found, the Connection Manager searches the `sqlhosts` files of database servers that are specified by a connection-unit's ONEDB_SERVER parameter.

Usage

The SQLHOSTS parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The SQLHOSTS option is useful when you want to restrict client applications from accessing one or more database servers in a high-availability cluster.

Example

Example

In the following example, the SQLHOSTS parameter is used to limit the servers that the Connection Manager connects to.

```
NAME my_connection_manager
SQLHOSTS LOCAL

CLUSTER my_cluster
{
   INFORMIXSERVERS my_servers
   SLA my_sla DBSERVERS=server_1,server_2,server_3,server_4
   FOC ORDER=ENABLED PRIORITY=1
}
```

The local `sqlhosts` file has the following entries:

```
#dbservername       nettype    hostname   servicename   options
 my_servers         group      -          -             c=1,e=server_3
 server_1           onsoctcp   host_1     port_1        g=my_servers
 server_2           onsoctcp   host_2     port_2        g=my_servers
 server_3           onsoctcp   host_3     port_3        g=my_servers
```

Remote database-server `sqlhosts` files have the following entries:

```
#dbservername       nettype    hostname   servicename   options
 my_servers         group      -          -             c=1,e=server_4
 server_1           onsoctcp   host_1     port_1        g=my_servers
 server_2           onsoctcp   host_2     port_2        g=my_servers
 server_3           onsoctcp   host_3     port_3        g=my_servers
 server_4           onsoctcp   host_3     port_3        g=my_servers
```

server_4 is not defined in the local `sqlhosts` file, and the Connection Manager is configured to not search remote `sqlhosts` files, so the Connection Manager does not send connection requests to **server_4**.

SSL_LABEL Connection Manager configuration parameter

The SSL_LABEL parameter specifies the certificate label used for authentication by the CM when listening for an SSL connection.

```
SSL_LABEL certificate_name
```

The certificate name can be a single value, combination of alphanumeric characters, symbols or numbers.

Usage

The SSL_LABEL parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The SSL_LABEL option is useful when you want the CM to use the name of certificate specified in SSL_LABEL to authenticate incoming SSL connections.

Example

Example

```
SSL_LABEL  cm1ListeningCertificate
```

Modifying Connection Manager configuration files

Use the oncmsm utility to load a modified configuration file into a Connection Manager and change the Connection Manager's configuration.

Before you begin

About this task

If you are using multiple Connection Managers, you can run the onstat -g cmsm command to display the names of Connection Manager instances.

1. Modify the existing Connection Manager configuration file.
   The default location of the configuration file is the `$ONEDB_HOME/etc` directory.
2. On the Connection Manager's host, run the oncmsm utility with the r parameter and the name of the Connection Manager instance.
   Example
   For example:
   ```
   oncmsm -r connection_manager_name
   ```
   Because multiple instances of the Connection Manager can be active at the same time, you must specify the name of the Connection Manager instance.

Results

After you reload a Connection Manager's configuration file, the new configuration immediately takes effect.

Converting older formats of the Connection Manager configuration file to the current format

The Connection Manager configuration file in versions of HCL OneDB™ Client Software Development Kit (Client SDK) before version 3.70.xC3 are incompatible with the current version of the Connection Manager. You must convert configuration files from versions before 3.70.xC3 by running the oncmsm utility.

Before you begin

If you have multiple configuration files to convert, combine related SLA definitions into a single configuration file.

About this task

You can convert Connection Manager configuration files, even if a Connection Manager is running.

To convert a Connection Manager file to the current format:

1. Log on to the computer on which the updated Client SDK is installed.
2. Run the oncmsm utility, specifying old and new configuration file names.

Results

After the configuration file is converted, it can be loaded into a Connection Manager.

Example

Example: Converting a configuration file and loading it into a Connection Manager

For the following example:

- You are using a UNIX™ operating system.
- The Connection Manager was using a configuration file that is named `configuration_file_old` and located in `$ONEDB_HOME/etc`.
- You want to convert the file to the new format and rename the file to `configuration_file_new`.

Run the following commands:

1. `oncmsm -c configuration_file_old -n configuration_file_new`
2. `oncmsm -c configuration_file_new`

Configuring environments and setting configuration parameters for connection management

Before you start a Connection Manager, you must configure its environment.

About this task

To set environment variables:

- For UNIX™ C use the appropriate shell command. The following examples use C shell (csh).
- For Windows™, use the **Environment** tab of the setnet32 utility.

1. If clients or Connection Managers are installed on hosts where database servers are not installed, set each host's **ONEDB_HOME** environment variable to the directory the client or Connection Manager is installed in.
   Example
   Run the following command:
   ```
   setenv ONEDB_HOME path
   ```
2. If clients or Connection Managers are installed on hosts where database servers are not installed, set each host's **ONEDB_ SQLHOSTS** environment variable to the location of the `sqlhosts` file that the host uses.
   Example
   Run the following command:
   ```
   setenv ONEDB_ SQLHOSTS path_and_filename
   ```
3. If a Connection Manager's configuration file is in a directory other than `$ONEDB_HOME/etc`, set the **CMCONFIG** environment variable to the directory.
   Example
   Run the following command:
   ```
   setenv CMCONFIG path_and_filename
   ```

4. If Connection Managers control failover for a high-availability cluster, set the `onconfig` file DRAUTO configuration parameter to `3` on all managed cluster database servers.
   Example
   For example:
   ```
   DRAUTO 3
   ```
5. If Connection Managers control failover for a high-availability cluster, set the `onconfig` file HA_FOC_ORDER configuration parameter on the primary server of each cluster to a failover order.
   Example
   For example:
   ```
   HA_FOC_ORDER SDS,HDR,RSS
   ```
6. On each database server that uses multiple ports, set the `onconfig` file DBSERVERALIASES configuration parameter to the alias names listed in Connection Manager and database server `sqlhosts` file entries.
   Example
   For example:

   A Connection Manager's host has the following `sqlhost` file entries:
   ```
   #dbservername   nettype    hostname   servicename   options
    my_servers     group      -          -             c=1,e=server_3
    server_1       onsoctcp   host_1     port_1        s=6,g=my_servers
    server_2       onsoctcp   host_2     port_2        s=6,g=my_servers
    server_3       onsoctcp   host_3     port_3        s=6,g=my_servers

    my_aliases     group      -          -             c=1,e=a_server_3
    a_server_1     onsoctcp   host_1     port_4        g=my_aliases
    a_server_2     onsoctcp   host_2     port_5        g=my_aliases
    a_server_3     onsoctcp   host_3     port_6        g=my_aliases
   ```
   In the `onconfig` file for **server_1**, set the following value:
   ```
   DBSERVERALIASES a_server_1
   ```
   In the `onconfig` file for **server_2**, set the following value:
   ```
   DBSERVERALIASES a_server_2
   ```
   In the `onconfig` file for **server_3**, set the following value:
   ```
   DBSERVERALIASES a_server_3
   ```

Defining sqlhosts information for connection management

You must define `sqlhosts` network-connectivity information for client applications that connect to Connection Managers, Connection Managers that connect to database servers, and database servers that are part of a Connection Manager connection unit.

Before you begin

If Connection Managers or clients are installed on hosts where database servers are not installed, you must create a `sqlhosts` file on each host.

About this task

Entries in an `sqlhosts` file can specify connection information for the following connection-unit components:

- Database servers
- Aliases for database servers that are using secure ports
- Connection Manager service-level agreements (SLAs)
- Groups that can contain database servers, database-server aliases, or SLAs.

All database servers that a Connection Manager connects to must be listed in the `sqlhosts` file that the Connection Manager uses. If the Connection Manager is monitoring a high-availability cluster, the `sqlhosts` file the Connection Manager uses must contain entries for all cluster servers.

1. Create entries in each database server's host `sqlhosts` file.
   You can modify one `sqlhosts` file, and then distribute it to the hosts of other database servers.
2. Create entries in each Connection Manager's host `sqlhosts` file.
   You can create one `sqlhosts` file, and then distribute it to the hosts of other Connection Managers.
3. Create entries in each client application's host `sqlhosts` file.
   You can create one `sqlhosts` file, and then distribute it to the hosts of other client applications.
4. If a host has multiple database servers that are installed on it, if the `sqlhosts` file is in a directory other than `$ONEDB_HOME/etc`, or if you are using a network-connectivity file other than `$ONEDB_HOME/etc/sqlhosts`, set the host's **ONEDB_ SQLHOSTS** environment variable to the location of the `sqlhosts` file.

What to do next
If `sqlhosts` file entries use the `s=6` option to define secure ports, use the information in the `sqlhosts` file to create a password file.

Defining sqlhosts information for connection management of high-availability clusters

You must define `sqlhosts` network-connectivity information for connection management of high-availability clusters.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server in the cluster.
   Example
   For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1
 server_2       onsoctcp   host_2     port_2
 server_3       onsoctcp   host_3     port_3
```

2. On the host of each Connection Manager, add a group entry that contains each database server in the cluster, and add group options to the database-server entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     -          -                        c=1,e=server_3
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
 server_3       onsoctcp   host_3     port_3        g=my_servers
```

3. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_1     DBSERVERS=PRI
   SLA sla_secondaries_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

   The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_2     DBSERVERS=PRI
   SLA sla_secondaries_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

   Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername       nettype    hostname   servicename   options
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2

 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4
```

4. On the host of each client application, create `sqlhosts` file entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of

group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

Example

```
#dbservername       nettype    hostname   servicename  options
 g_primary          group      -          -            c=1,e=sla_primary_2
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1    g=g_primary
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2    g=g_primary

 g_secondaries      group      -          -            c=1,e=sla_secondaries_2
 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3    g=g_secondaries
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4    g=g_secondaries
```

Results

Client connection requests to **@g_primary** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2** to provide the client application with connection information for the primary server.

Defining sqlhosts information for connection management of high-availability clusters that use secure ports

You must define `sqlhosts` network-connectivity information for connection management of high-availability clusters. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server in the cluster, and specify the `s=6` secure-port option.
   Example
   For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 server_2       onsoctcp   host_2     port_2        s=6
 server_3       onsoctcp   host_3     port_3        s=6
```

2. On the host of each Connection Manager and database server, create `sqlhosts` file alias entries for each database server.
   Example
   For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 a_server_1     onsoctcp   host_1     port_4

 server_2       onsoctcp   host_2     port_2        s=6
 a_server_2     onsoctcp   host_2     port_5

 server_3       onsoctcp   host_3     port_3        s=6
 a_server_3     onsoctcp   host_3     port_6
```

3. On the host of each Connection Manager, add a group entry the individual entries. Add group options to the database server and database server alias entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     group      -          -             c=1,e=a_server_3
 server_1       onsoctcp   host_1     port_1        s=6,g=my_servers
 a_server_1     onsoctcp   host_1     port_4        g=my_servers
 server_2       onsoctcp   host_2     port_2        s=6,g=my_servers
 a_server_2     onsoctcp   host_2     port_5        g=my_servers
 server_3       onsoctcp   host_3     port_3        s=6,g=my_servers
 a_server_3     onsoctcp   host_3     port_6        g=my_servers
```

   A password file that is encrypted through the onpassword utility is required for connectivity through secure ports. The entries in the previously shown `sqlhosts` file are represented in the following password file.

```
my_servers  a_server_1  user_1  my_password_1
my_servers  a_server_2  user_2  my_password_2
my_servers  a_server_3  user_3  my_password_3

server_1    a_server_1  user_1  my_password_1
server_2    a_server_2  user_2  my_password_2
server_3    a_server_3  user_3  my_password_3

a_server_1  a_server_1  user_1  my_password_1
a_server_2  a_server_2  user_2  my_password_2
a_server_3  a_server_3  user_3  my_password_3
```

4. In each database server's `onconfig` file, set the DBSERVERALIASES parameter to that database server's alias.

   Example

   The `onconfig` file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

   The `onconfig` file entry for **server_2**:

```
DBSERVERALIASES a_server_2
```

   The `onconfig` file entry for **server_3**:

```
DBSERVERALIASES a_server_3
```

5. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_1     DBSERVERS=PRI
   SLA sla_secondaries_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_2     DBSERVERS=PRI
   SLA sla_secondaries_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED PRIORITY=2
}
```

Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername       nettype    hostname   servicename   options
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2

 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4
```

6. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

```
#dbservername       nettype    hostname   servicename   options
 g_primary          group      -          -             c=1,e=sla_primary_2
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1     g=g_primary
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2     g=g_primary

 g_secondaries      group      -          -             c=1,e=sla_secondaries_2
 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3     g=g_secondaries
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4     g=g_secondaries
```

Results

Client connection requests to **@g_primary** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2** to provide the client application with connection information for the primary server.

Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture™ (DRDA®)

Connection Managers support Distributed Relational Database Architecture™ (DRDA®) connections for high-availability clusters. You must define `sqlhosts` network-connectivity information for connection management of high-availability clusters that use DRDA®.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server in the cluster.
   Example
   For example:

   ```
   #dbservername   nettype    hostname   servicename   options
    server_1       onsoctcp   host_1     port_1
    server_2       onsoctcp   host_3     port_2
    server_3       onsoctcp   host_5     port_3
   ```
2. In each database server's `onconfig` file, set the DBSERVERALIASES parameter to specify an alias for the server.
   Example

   The `onconfig` file entry for **server_1**:

   ```
   DBSERVERALIASES drda_1
   ```

   The `onconfig` file entry for **server_2**:

   ```
   DBSERVERALIASES drda_2
   ```

   The `onconfig` file entry for **server_3**:

   ```
   DBSERVERALIASES drda_3
   ```
3. On the host of each Connection Manager, add entries for the DRDA® aliases. Use a DRDA® protocol for the `nettype` value.
   Example
   For example:

   ```
   #dbservername   nettype    hostname   servicename   options
    server_1       onsoctcp   host_1     port_1
    server_2       onsoctcp   host_2     port_2
    server_3       onsoctcp   host_3     port_3

    drda_1         drsoctcp   host_1     port_4
    drda_2         drsoctcp   host_2     port_5
    drda_3         drsoctcp   host_3     port_6
   ```
4. On the host of each Connection Manager, add a group entry for the group of database server and add a group entry for the group of DRDA® aliases. Add group options to the database server and DRDA® alias entries. Use the `c=1`

group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

Example

For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     group      -          -             c=1,e=server_3
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
 server_3       onsoctcp   host_3     port_3        g=my_servers

 drda_aliases   group      -          -             c=1,e=drda_3
 drda_1         drsoctcp   host_1     port_4        g=drda_aliases
 drda_2         drsoctcp   host_2     port_5        g=drda_aliases
 drda_3         drsoctcp   host_3     port_6        g=drda_aliases
```

5. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

Example

The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_1          DBSERVERS=PRI
   SLA sla_primary_drda_1     DBSERVERS=PRI
   SLA sla_secondaries_1      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   ONEDB_SERVER my_servers
   SLA sla_primary_2          DBSERVERS=PRI
   SLA sla_primary_drda_2     DBSERVERS=PRI
   SLA sla_secondaries_2      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername            nettype    hostname   servicename  options
 sla_primary_1           onsoctcp   cm_host_1  cm_port_1
 sla_primary_2           onsoctcp   cm_host_2  cm_port_2

 sla_secondaries_2       onsoctcp   cm_host_1  cm_port_3
 sla_secondaries_2       onsoctcp   cm_host_2  cm_port_4

 sla_primary_1_drda      drsoctcp   cm_host_1  cm_port_5
 sla_primary_2_drda      drsoctcp   cm_host_2  cm_port_6

 sla_secondaries_2_drda  drsoctcp   cm_host_1  cm_port_7
 sla_secondaries_2_drda  drsoctcp   cm_host_2  cm_port_8
```

6. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.
   Example

```
#dbservername            nettype    hostname   servicename  options
 g_primary               group      -          -            c=1,e=sla_primary_2
 sla_primary_1           onsoctcp   cm_host_1  cm_port_1    g=g_primary
 sla_primary_2           onsoctcp   cm_host_2  cm_port_2    g=g_primary

 g_secondaries           group      -          -            c=1,e=sla_secondaries_2
 sla_secondaries_2       onsoctcp   cm_host_1  cm_port_3    g=g_secondaries
 sla_secondaries_2       onsoctcp   cm_host_2  cm_port_4    g=g_secondaries

 g_primary_drda          group      -          -            c=1,e=sla_primary_2_drda
 sla_primary_1_drda      drsoctcp   cm_host_1  cm_port_5    g=g_primary_drda
 sla_primary_2_drda      drsoctcp   cm_host_2  cm_port_6    g=g_primary_drda

 g_secondaries_drda      group      -          -            c=1,e=sla_secondaries_2_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_1  cm_port_7    g=g_secondaries_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_2  cm_port_8    g=g_secondaries_drda
```

Results

Client connection requests to **@g_primary_drda** are sent by **drsoctcp** protocol to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1_drda** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2_drda** to provide the client application with connection information for the primary server.

Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture™ (DRDA®) and secure ports

Connection Managers support Distributed Relational Database Architecture™ (DRDA®) connections for high-availability clusters. You must define `sqlhosts` network-connectivity information for connection management of high-availability clusters that use DRDA®. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

About this task

The Connection Manager's `sqlhosts` file must contain entries for all database servers that it connects to.

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server in the cluster, and specify the `s=6` secure-port option.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_3    port_2       s=6
 server_3      onsoctcp  host_5    port_3       s=6
```

2. On the host of each Connection Manager and database server, add DRDA® alias entries. Use a DRDA® protocol for the `nettype` value, and specify the `s=6` secure-port option.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_2    port_2       s=6
 server_3      onsoctcp  host_3    port_3       s=6

 drda_1        drsoctcp  host_1    port_4       s=6
 drda_2        drsoctcp  host_2    port_5       s=6
 drda_3        drsoctcp  host_3    port_6       s=6
```

3. On the host of each Connection Manager and database server, create `sqlhosts` file alias entries for each database server and each DRDA® alias.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 a_server_1    onsoctcp  host_1    port_7

 server_2      onsoctcp  host_2    port_2       s=6
 a_server_2    onsoctcp  host_2    port_8

 server_3      onsoctcp  host_3    port_3       s=6
 a_server_3    onsoctcp  host_3    port_9

 drda_1        drsoctcp  host_1    port_4       s=6
 a_drda_1      drsoctcp  host_1    port_10

 drda_2        drsoctcp  host_2    port_5       s=6
 a_drda_2      drsoctcp  host_2    port_11

 drda_3        drsoctcp  host_3    port_6       s=6
 a_drda_3      drsoctcp  host_3    port_12
```

4. On the host of each Connection Manager, add group entries for the groups of database servers and DRDA® entries. Add group options to the individual entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

For example:

```
#dbservername  nettype   hostname  servicename  options
 g_servers     group     -         -            c=1,e=a_server_3
 server_1      onsoctcp  host_1    port_1       s=6,g=g_servers
 a_server_1    onsoctcp  host_1    port_7       g=g_servers
 server_2      onsoctcp  host_2    port_2       s=6,g=g_servers
 a_server_2    onsoctcp  host_2    port_8       g=g_servers
 server_3      onsoctcp  host_3    port_3       s=6,g=g_servers
 a_server_3    onsoctcp  host_3    port_9       g=g_servers

 g_drda        group     -         -            c=1,e=a_drda_3
 drda_1        drsoctcp  host_1    port_4       s=6,g=g_drda
 a_drda_1      drsoctcp  host_1    port_10      g=g_drda
 drda_2        drsoctcp  host_2    port_5       s=6,g=g_drda
 a_drda_2      drsoctcp  host_2    port_11      g=g_drda
 drda_3        drsoctcp  host_3    port_6       s=6,g=g_drda
 a_drda_3      drsoctcp  host_3    port_12      g=g_drda
```

A password file that is encrypted through the onpassword utility is required for connectivity through secure ports. The entries in the previously shown `sqlhosts` file are represented in the following password file.

```
g_servers   a_server_1  user_1  password_1
g_servers   a_server_2  user_2  password_2
g_servers   a_server_3  user_3  password_3

server_1    a_server_1  user_1  password_1
server_2    a_server_2  user_2  password_2
server_3    a_server_3  user_3  password_3

a_server_1  a_server_1  user_1  password_1
a_server_2  a_server_2  user_2  password_2
a_server_3  a_server_3  user_3  password_3

g_drda      a_drda_1    user_1  password_1
g_drda      a_drda_2    user_2  password_2
g_drda      a_drda_3    user_3  password_3

drda_1      a_drda_1    user_1  password_1
drda_2      a_drda_2    user_2  password_2
drda_3      a_drda_3    user_3  password_3

a_drda_1    a_drda_1    user_1  password_1
a_drda_2    a_drda_2    user_2  password_2
a_drda_3    a_drda_3    user_3  password_3
```

5. In each database server's `onconfig` file, set the DBSERVERALIASES parameter to that database server's aliases.

Example

The `onconfig` file entry for **server_1**:

```
DBSERVERALIASES a_server_1,drda_1,a_drda_1
```

The `onconfig` file entry for **server_2**:

```
DBSERVERALIASES a_server_2,drda_2,a_drda_2
```

The `onconfig` file entry for **server_3**:

```
DBSERVERALIASES a_server_3,drda_3,a_drda_3
```

6. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

   The first Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_1

   CLUSTER my_cluster
   {
      ONEDB_SERVER g_servers,g_drda
      SLA sla_primary_1          DBSERVERS=PRI
      SLA sla_primary_drda_1     DBSERVERS=PRI
      SLA sla_secondaries_1      DBSERVERS=SDS,HDR
      SLA sla_secondaries_drda_1 DBSERVERS=SDS,HDR
      FOC ORDER=ENABLED \
          PRIORITY=1
   }
   ```

   The second Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_2

   CLUSTER my_cluster
   {
      ONEDB_SERVER g_servers,g_drda
      SLA sla_primary_2          DBSERVERS=PRI
      SLA sla_primary_drda_2     DBSERVERS=PRI
      SLA sla_secondaries_2      DBSERVERS=SDS,HDR
      SLA sla_secondaries_drda_2 DBSERVERS=SDS,HDR
      FOC ORDER=ENABLED \
          PRIORITY=2
   }
   ```

   Add the following entries to each client application's host `sqlhosts` file:

   ```
   #dbservername            nettype   hostname   servicename  options
    sla_primary_1           onsoctcp  cm_host_1  cm_port_1
    sla_primary_2           onsoctcp  cm_host_2  cm_port_2

    sla_secondaries_2       onsoctcp  cm_host_1  cm_port_3
    sla_secondaries_2       onsoctcp  cm_host_2  cm_port_4

    sla_primary_1_drda      drsoctcp  cm_host_1  cm_port_5
    sla_primary_2_drda      drsoctcp  cm_host_2  cm_port_6

    sla_secondaries_2_drda  drsoctcp  cm_host_1  cm_port_7
    sla_secondaries_2_drda  drsoctcp  cm_host_2  cm_port_8
   ```

7. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

```
#dbservername            nettype    hostname   servicename  options
 g_primary               group      -          -            c=1,e=sla_primary_2
 sla_primary_1           onsoctcp   cm_host_1  cm_port_1    g=g_primary
 sla_primary_2           onsoctcp   cm_host_2  cm_port_2    g=g_primary

 g_secondaries           group      -          -            c=1,e=sla_secondaries_2
 sla_secondaries_2       onsoctcp   cm_host_1  cm_port_3    g=g_secondaries
 sla_secondaries_2       onsoctcp   cm_host_2  cm_port_4    g=g_secondaries

 g_primary_drda          group      -          -            c=1,e=sla_primary_2_drda
 sla_primary_1_drda      drsoctcp   cm_host_1  cm_port_5    g=g_primary_drda
 sla_primary_2_drda      drsoctcp   cm_host_2  cm_port_6    g=g_primary_drda

 g_secondaries_drda      group      -          -            c=1,e=sla_secondaries_2_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_1  cm_port_7    g=g_secondaries_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_2  cm_port_8    g=g_secondaries_drda
```

Results

Client connection requests to **@g_primary_drda** are sent by **drsoctcp** protocol to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1_drda** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2_drda** to provide the client application with connection information for the primary server.

Defining sqlhosts information for connection management of grids and replicate sets

You must define `sqlhosts` network-connectivity information for connection management of replicate sets or grids.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and replication server, create `sqlhosts` file entries for each replication server.
   Example
   For example:

```
#dbservername  nettype   hostname   servicename  options
 server_1      onsoctcp  host_1     port_1
 server_2      onsoctcp  host_2     port_2
 server_3      onsoctcp  host_3     port_3
 server_4      onsoctcp  host_4     port_4
```

2. On the host of each Connection Manager and replication server, create a `sqlhosts` file group entry for each replication server. Add group options to each replication-server entry. Use the `i=`*`unique_number`* group-entry option to assign an identifier to the group for Enterprise Replication. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.
   Example
   For example:

```
#dbservername  nettype   hostname   servicename  options
 g_server_1    group     -          -            i=1,e=server_1
 server_1      onsoctcp  host_1     port_1       g=g_server_1

 g_server_2    group     -          -            i=2,e=server_2
 server_2      onsoctcp  host_2     port_2       g=g_server_2

 g_server_3    group     -          -            i=3,e=server_3
 server_3      onsoctcp  host_3     port_3       g=g_server_3

 g_server_4    group     -          -            i=4,e=server_4
 server_4      onsoctcp  host_4     port_4       g=g_server_4
```

3. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_1 DBSERVERS=ANY
}
```

   The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_2 DBSERVERS=ANY
}
```

   Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

4. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -             c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1     g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2     g=g_sla
```

Results

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.

Defining sqlhosts information for connection management of grids and replicate sets that use secure ports

You must define `sqlhosts` network-connectivity information for connection management of replicate sets or grids. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and replication server, create `sqlhosts` file entries for each replication server, and specify the `s=6` secure-port option.
   Example
   For example:
```
#dbservername  nettype   hostname   servicename  options
 server_1      onsoctcp  host_1     port_1       s=6
 server_2      onsoctcp  host_2     port_3       s=6
 server_3      onsoctcp  host_3     port_5       s=6
 server_4      onsoctcp  host_4     port_7       s=6
```
2. On the host of each Connection Manager and replication server, create a `sqlhosts` file alias entry for each replication server.
   Example
   For example:
```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 a_server_1    onsoctcp  host_1    port_2

 server_2      onsoctcp  host_2    port_3       s=6
 a_server_2    onsoctcp  host_2    port_4

 server_3      onsoctcp  host_3    port_5       s=6
 a_server_3    onsoctcp  host_3    port_6

 server_4      onsoctcp  host_4    port_7       s=6
 a_server_4    onsoctcp  host_4    port_8
```
   The aliases are used by the cdr utility, which cannot connect to a secure port.
3. On the host of each Connection Manager and replication server, create a `sqlhosts` file group entry for each replication server and alias pair. Use the `i=`*`unique_number`* group-entry option to assign an identifier to the group for

Enterprise Replication. Add group options to each replication server and alias entry. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

Example

For example:

```
#dbservername  nettype   hostname   servicename  options
 g_server_1    group     -          -            i=1,e=a_server_1
 server_1      onsoctcp  host_1     port_1       g=g_server_1,s=6
 a_server_1    onsoctcp  host_1     port_2       g=g_server_1

 g_server_2    group     -          -            i=2,e=a_server_2
 server_2      onsoctcp  host_2     port_3       g=g_server_2,s=6
 a_server_2    onsoctcp  host_2     port_4       g=g_server_2

 g_server_3    group     -          -            i=3,e=a_server_3
 server_3      onsoctcp  host_3     port_5       g=g_server_3,s=6
 a_server_3    onsoctcp  host_3     port_6       g=g_server_3

 g_server_4    group     -          -            i=4,e=a_server_4
 server_4      onsoctcp  host_4     port_7       g=g_server_4,s=6
 a_server_4    onsoctcp  host_4     port_8       g=g_server_4
```

A password file that is encrypted through the onpassword utility is required for connectivity through secure ports. The entries in the previously shown `sqlhosts` file are represented in the following password file.

```
 g_server_1  a_server_1  user_1  my_password_1
 server_1    a_server_1  user_1  my_password_1
 a_server_1  a_server_1  user_1  my_password_1

 g_server_2  a_server_2  user_2  my_password_2
 server_2    a_server_2  user_2  my_password_2
 a_server_2  a_server_2  user_2  my_password_2

 g_server_3  a_server_3  user_3  my_password_3
 server_3    a_server_3  user_3  my_password_3
 a_server_3  a_server_3  user_3  my_password_3

 g_server_4  a_server_4  user_4  my_password_4
 server_4    a_server_4  user_4  my_password_4
 a_server_4  a_server_4  user_4  my_password_4
```

4. In each replication server's `onconfig` file, set the DBSERVERALIASES parameter to that database server's aliases.

   Example

   The `onconfig` file entry for **server_1**:

   ```
   DBSERVERALIASES a_server_1
   ```

   The `onconfig` file entry for **server_2**:

   ```
   DBSERVERALIASES a_server_2
   ```

   The `onconfig` file entry for **server_3**:

   ```
   DBSERVERALIASES a_server_3
   ```

The `onconfig` file entry for **server_4**:

```
DBSERVERALIASES a_server_4
```

5. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
   Example

   The first Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_1

   REPLSET my_replset
   {
      ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
      SLA sla_1 DBSERVERS=ANY
   }
   ```

   The second Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_2

   REPLSET my_replset
   {
      ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
      SLA sla_2 DBSERVERS=ANY
   }
   ```

   Add the following entries to each client application's host `sqlhosts` file:

   ```
   #dbservername  nettype   hostname   servicename  options
    sla_1         onsoctcp  cm_host_1  cm_port_1
    sla_2         onsoctcp  cm_host_2  cm_port_2
   ```

6. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.
   Example
   For example:

   ```
   #dbservername  nettype   hostname   servicename  options
    g_sla         group     -          -            c=1,e=sla_2
    sla_1         onsoctcp  cm_host_1  cm_port_1    g=g_sla
    sla_2         onsoctcp  cm_host_2  cm_port_2    g=g_sla
   ```

Results

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for the primary server.

Defining sqlhosts information for connection management high-availability replication systems

You must define `sqlhosts` network-connectivity information for connection management of high-availability replication systems.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

For this example, you are setting up Enterprise Replication between the primary servers of three high-availability clusters.

Cluster 1:

- server_1 (primary)
- server_2 (SD secondary)
- server_3 (HDR secondary)
- server_4 (RS secondary)

Cluster 2:

- server_5 (primary)
- server_6 (SD secondary)
- server_7 (HDR secondary)
- server_8 (RS secondary)

Cluster 3:

- server_9 (primary)
- server_10 (SD secondary)
- server_11 (HDR secondary)
- server_12 (RS secondary)

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each server.
   Example
   For example:

```
#dbservername  nettype   hostname   servicename  options
server_1       onsoctcp  host_1     port_1
server_2       onsoctcp  host_1     port_2
server_3       onsoctcp  host_2     port_3
server_4       onsoctcp  host_3     port_4

server_5       onsoctcp  host_4     port_5
server_6       onsoctcp  host_4     port_6
server_7       onsoctcp  host_5     port_7
server_8       onsoctcp  host_6     port_8
```

```
server_9       onsoctcp  host_7     port_9
server_10      onsoctcp  host_7     port_10
server_11      onsoctcp  host_8     port_11
server_12      onsoctcp  host_9     port_12
```

2. On the host of each Connection Manager and database server, create a `sqlhosts` file group entry for each replication-server entry and each cluster. Add group options to each database server entry. Use the `i=unique_number` group-entry option to assign an identifier to the group for Enterprise Replication. Use the `c=1` group-entry option for cluster groups, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername  nettype   hostname   servicename  options
 cluster_1     group     -          -            i=1,c=1,e=server_4
 server_1      onsoctcp  host_1     port_1       g=cluster_1
 server_2      onsoctcp  host_1     port_2       g=cluster_1
 server_3      onsoctcp  host_2     port_3       g=cluster_1
 server_4      onsoctcp  host_3     port_4       g=cluster_1

 cluster_2     group     -          -            i=2,c=1,e=server_8
 server_5      onsoctcp  host_4     port_5       g=cluster_2
 server_6      onsoctcp  host_4     port_6       g=cluster_2
 server_7      onsoctcp  host_5     port_7       g=cluster_2
 server_8      onsoctcp  host_6     port_8       g=cluster_2

 cluster_3     group     -          -            i=3,c=1,e=server_12
 server_9      onsoctcp  host_7     port_9       g=cluster_3
 server_10     onsoctcp  host_7     port_10      g=cluster_3
 server_11     onsoctcp  host_8     port_11      g=cluster_3
 server_12     onsoctcp  host_9     port_12      g=cluster_3
```

3. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   ONEDB_SERVER cluster_1,cluster_2,cluster_3
   SLA sla_1 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   ONEDB_SERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_2
{
   ONEDB_SERVER cluster_2
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_3
{
   ONEDB_SERVER cluster_3
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   ONEDB_SERVER cluster_1,cluster_2,cluster_3
   SLA sla_2 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   ONEDB_SERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=2
}

CLUSTER my_cluster_2
{
   ONEDB_SERVER cluster_2
   FOC ORDER=ENABLED \
       PRIORITY=2
}

CLUSTER my_cluster_3
{
   ONEDB_SERVER cluster_3
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

4. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -            c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1    g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2    g=g_sla
```

Results

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.

Defining sqlhosts information for connection management of high-availability replication systems that use secure ports

You must define `sqlhosts` network-connectivity information for connection management of high-availability replication systems. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

About this task

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

For this example, you are setting up Enterprise Replication between the primary servers of two high-availability clusters.

Cluster 1:

- **server_1** (primary)
- **server_2** (SD secondary)
- **server_3** (HDR secondary)
- **server_4** (RS secondary)

Cluster 2:

- **server_5** (primary)
- **server_6** (SD secondary)
- **server_7** (HDR secondary)
- **server_8** (RS secondary)

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_1    port_2       s=6
 server_3      onsoctcp  host_2    port_3       s=6
 server_4      onsoctcp  host_3    port_4       s=6
 server_5      onsoctcp  host_4    port_5       s=6
 server_6      onsoctcp  host_4    port_6       s=6
 server_7      onsoctcp  host_5    port_7       s=6
 server_8      onsoctcp  host_6    port_8       s=6
```

2. On the host of each Connection Manager and database server, create a `sqlhosts` file alias entry for each database server.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 a_server_1    onsoctcp  host_1    port_9

 server_2      onsoctcp  host_1    port_2       s=6
 a_server_2    onsoctcp  host_1    port_10

 server_3      onsoctcp  host_2    port_3       s=6
 a_server_3    onsoctcp  host_2    port_11

 server_4      onsoctcp  host_3    port_4       s=6
 a_server_4    onsoctcp  host_3    port_12

 server_5      onsoctcp  host_4    port_5       s=6
 a_server_5    onsoctcp  host_4    port_13

 server_6      onsoctcp  host_4    port_6       s=6
 a_server_6    onsoctcp  host_4    port_14

 server_7      onsoctcp  host_5    port_7       s=6
 a_server_7    onsoctcp  host_5    port_15

 server_8      onsoctcp  host_6    port_8       s=6
 a_server_8    onsoctcp  host_6    port_16
```

   The aliases are used by the cdr utility, which cannot connect to a secure port.

3. On the host of each Connection Manager and database server, create a `sqlhosts` file group entry for each cluster. Add group options to each database server entry. Use the `i=`*`unique_number`* group-entry option to assign an identifier to the group for Enterprise Replication. Use the `c=1` group-entry option for cluster groups, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername  nettype   hostname  servicename  options
 cluster_1     group     -          -           i=1,c=1,e=a_server_4
 server_1      onsoctcp  host_1    port_1       s=6,g=cluster_1
 a_server_1    onsoctcp  host_1    port_9       g=cluster_1
 server_2      onsoctcp  host_1    port_2       s=6,g=cluster_1
 a_server_2    onsoctcp  host_1    port_10      g=cluster_1
 server_3      onsoctcp  host_2    port_3       s=6,g=cluster_1
 a_server_3    onsoctcp  host_2    port_11      g=cluster_1
 server_4      onsoctcp  host_3    port_4       s=6,g=cluster_1
 a_server_4    onsoctcp  host_3    port_12      g=cluster_1
 cluster_2     group     -          -           i=1,c=1,e=a_server_8
 server_5      onsoctcp  host_4    port_5       s=6,g=cluster_2
 a_server_5    onsoctcp  host_4    port_13      g=cluster_2
 server_6      onsoctcp  host_4    port_6       s=6,g=cluster_2
 a_server_6    onsoctcp  host_4    port_14      g=cluster_2
 server_7      onsoctcp  host_5    port_7       s=6,g=cluster_2
 a_server_7    onsoctcp  host_5    port_15      g=cluster_2
 server_8      onsoctcp  host_6    port_8       s=6,g=cluster_2
 a_server_8    onsoctcp  host_6    port_16      g=cluster_2
```

A password file that is encrypted through the onpassword utility is required for connectivity through secure ports. The entries in the previously shown `sqlhosts` file are represented in the following password file.

```
cluster_1   a_server_1  user_1  password_1
cluster_1   a_server_2  user_2  password_2
cluster_1   a_server_3  user_3  password_3
cluster_1   a_server_4  user_4  password_4

cluster_2   a_server_5  user_5  password_5
cluster_2   a_server_6  user_6  password_6
cluster_2   a_server_7  user_7  password_7
cluster_2   a_server_8  user_8  password_8

server_1    a_server_1  user_1  password_1
server_2    a_server_2  user_2  password_2
server_3    a_server_3  user_3  password_3
server_4    a_server_4  user_4  password_4
server_5    a_server_5  user_5  password_5
server_6    a_server_6  user_6  password_6
server_7    a_server_7  user_7  password_7
server_8    a_server_8  user_8  password_8

a_server_1  a_server_1  user_1  password_1
a_server_2  a_server_2  user_2  password_2
a_server_3  a_server_3  user_3  password_3
a_server_4  a_server_4  user_4  password_4
a_server_5  a_server_5  user_5  password_5
a_server_6  a_server_6  user_6  password_6
a_server_7  a_server_7  user_7  password_7
a_server_8  a_server_8  user_8  password_8
```

4. In each database server's `onconfig` file, set the DBSERVERALIASES parameter to that database server's aliases.

   Example

   For example:

   The `onconfig` file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

5. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.

   Example

   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   ONEDB_SERVER cluster_1,cluster_2
   SLA sla_1 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   ONEDB_SERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_2
{
   ONEDB_SERVER cluster_2
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   ONEDB_SERVER cluster_1,cluster_2
   SLA sla_2 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   ONEDB_SERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=2
}

CLUSTER my_cluster_2
{
   ONEDB_SERVER cluster_2
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

6. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=`*`last_member`* group-entry option so that the entire `sqlhosts` is not scanned for group members.

Example

For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -             c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1     g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2     g=g_sla
```

Results

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.

Defining sqlhosts information for connection management of server sets

You must define `sqlhosts` network-connectivity information for connection management of server sets.

About this task

The Connection Manager's `sqlhosts` file must contain entries for all database servers that it connects to.

To use a file other than `$ONEDB_HOME/etc/sqlhosts` on a specific host, set the host's **ONEDB_ SQLHOSTS** environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create `sqlhosts` file entries for each database server.
   Example
   For example:

```
#dbservername   nettype    hostname   servicename   options
 standalone_1   onsoctcp   host_1     port_1

 standalone_2   onsoctcp   host_2     port_2

 standalone_3   onsoctcp   host_3     port_3
```

2. On the host of each client application, create an `sqlhosts` file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
   Example

   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1
MACRO servers=standalone_1,standalone_2,standalone_3

SERVERSET ${servers}
{
   ONEDB_SERVER ${servers}
   SLA sla_1a DBSERVERS=standalone_1
   SLA sla_2a DBSERVERS=standalone_2
   SLA sla_3a DBSERVERS=standalone_3
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2
MACRO servers=standalone_1,standalone_2,standalone_3

SERVERSET ${servers}
{
   ONEDB_SERVER ${servers}
   SLA sla_1b DBSERVERS=standalone_1
   SLA sla_2b DBSERVERS=standalone_2
   SLA sla_3b DBSERVERS=standalone_3
}
```

Add the following entries to each client application's host `sqlhosts` file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1a        onsoctcp  cm_host_1  cm_port_1
 sla_1b        onsoctcp  cm_host_2  cm_port_2

 sla_2a        onsoctcp  cm_host_1  cm_port_3
 sla_2b        onsoctcp  cm_host_2  cm_port_4

 sla_3a        onsoctcp  cm_host_1  cm_port_5
 sla_3b        onsoctcp  cm_host_2  cm_port_6
```

3. On the host of each client application, create `sqlhosts` file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire `sqlhosts` is not scanned for group members.

   Example

   For example:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         group     -          -            c=1,e=sla_1b
 sla_1a        onsoctcp  cm_host_1  cm_port_1    g=sla_1
 sla_1b        onsoctcp  cm_host_2  cm_port_2    g=sla_1

 sla_2         group     -          -            c=1,e=sla_2b
 sla_2a        onsoctcp  cm_host_1  cm_port_3    g=sla_2
 sla_2b        onsoctcp  cm_host_2  cm_port_4    g=sla_2

 sla_3         group     -          -            c=1,e=sla_3b
 sla_3a        onsoctcp  cm_host_1  cm_port_5    g=sla_3
 sla_3b        onsoctcp  cm_host_2  cm_port_6    g=sla_3
```

Results

Client connection requests to **@sla_1** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1a** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_1b** to provide the client application with connection information for a replication server.

Creating a password file for connecting to database servers on untrusted networks

If a client, Connection Manager, or any of the database servers that a Connection Manager connects to are on an untrusted network, you can create encrypted password files to verify connection requests.

About this task

In certain situations, an encrypted password file is required for trusted network environments, such as when a local system account attempts to connect to a database server in a high-availability cluster or Enterprise Replication domain, or when the user ID does not exist on a database server. The password file provides the correct system-level access, so that a local system account or a Windows™ account can connect directly to a remote server.

The password file has separate entries for the following items:

- Each Enterprise Replication group
- Each High-availability cluster group
- Each High-availability cluster server
- Each Enterprise Replication server that is in a group that is also configured for high-availability
- Each database server's alternative server alias, if the database server is using a secure port for communication

A password file entry contains the following information:

- The name of an alternative server to connect to if a connection cannot be made to the listed server or group. For example, *alternative_server_name* is used when *server_or_group_name* uses a secure port, as specified by the `s=6` option in an `sqlhosts` file entry.
- The user ID for a database server or the database servers in a group. User IDs must have the following privileges:
    - Permission to connect to the **sysadmin** database
    - CONNECT permission on the remote servers
    - On UNIX™ operating systems, membership in the group **informix** DBSA group
    - On Windows™ operating systems, membership in the **Informix-Admin** DBSA group

  Only user **informix** has all of these privileges by default
- The password for a server

1. On a Connection Manager host, use a text editor to create an ASCII text file to be used as a password file. Save the file to the `$ONEDB_HOME/tmp` directory.
   If you have a high-availability replication system, your password file contains password information for replication servers and cluster servers.

   Note: The password file must not contain comments.

   Example

   The replication-server entries of the password file have the following format:

```
group_name              database_server_alias    user_name    database_server_password
database_server_name    database_server_alias    user_name    database_server_password
database_server_alias   database_server_alias    user_name    database_server_password
```

For example:

```
group_1   unsecure_server_alias_1  user_1  password_1
server_1  unsecure_server_alias_1  user_1  password_1
alias_1   unsecure_server_alias_1  user_1  password_1

group_2   unsecure_server_alias_2  user_2  password_2
server_2  unsecure_server_alias_2  user_2  password_2
alias_2   unsecure_server_alias_2  user_2  password_2

group_n   unsecure_server_alias_n  user_n  password_n
server_n  unsecure_server_alias_n  user_n  password_n
alias_n   unsecure_server_alias_n  user_n  password_n
```

The cluster-server entries of the password file have the following format:

```
alias_group_name    db_server_alias    user_name    db_server_password

db_server_name      db_server_alias    user_name    db_server_password
```

For example:

```
alias_group_1  unsecure_alias_1  user_1  password_1
alias_group_1  unsecure_alias_2  user_2  password_2
alias_group_1  unsecure_alias_n  user_n  password_n

alias_group_2  unsecure_alias_1  user_1  password_1
alias_group_2  unsecure_alias_2  user_2  password_2
alias_group_2  unsecure_alias_n  user_n  password_n

alias_group_n  unsecure_alias_1  user_1  password_1
alias_group_n  unsecure_alias_2  user_2  password_2
alias_group_n  unsecure_alias_n  user_n  password_n

server_1       unsecure_alias_1  user_1  password_1
server_2       unsecure_alias_2  user_2  password_2
server_n       unsecure_alias_n  user_n  password_n
```

2. Encrypt the password file with the onpassword utility and an encryption key.

   Example

   For example, if your password file is `$ONEDB_HOME/tmp/my_passwords.txt`, and the encryption key you want to use is **my_secret_encryption_key_efgh**, run the following command:

   ```
   onpassword -k my_secret_encryption_key_efgh -e my_passwords.txt
   ```

   This example creates the encrypted `passwd_file` file in the `$ONEDB_HOME/etc` directory.

   To later decrypt the password file, you must enter the same key that was used to encrypt the password file. If you lose the encryption key that was used to encrypt a password file, re-encrypt the original ASCII text password file. If the ASCII text password file was deleted, you must create a new one.

3. Distribute `$ONEDB_HOME/etc/passwd_file` to all the database servers that Connection Managers or the cdr utility connects to, and to all Connection Managers.

   Note: An encrypted password file that is created on one type of operating system is not supported on a different type of operating system. On each operating system, you must run the onpassword utility with the same text file and encryption key.

Modifying encrypted password information

Modify the information in the encrypted `passwd_file` file by running the onpassword utility.

About this task

Modify the encrypted `passwd_file` file when the following events occur:

- Database servers are added to or removed from a high-availability cluster or replication domain
- `sqlhosts` file server aliases or groups change
- User IDs or server passwords change
- You want to change your encryption key

1. Decrypt the `passwd_file` file by running onpassword utility, specifying the previously used encryption key and a name for the output file.
   Example
   For example, if you previously encrypted the file, and used **my_secret_encryption_key_asdf** as the encryption key, run the following command:
   ```
   onpassword -k my_secret_encryption_key_asdf -d my_passwords.txt
   ```
   The onpassword utility creates the ASCII text `my_passwords.txt` output file in the `$ONEDB_HOME/etc` directory.
2. **Optional:** Open the file with a text editor, and modify the information in the file.
3. Encrypt the password file with the onpassword utility, specifying an encryption key and the name of the text file.
   Example
   For example:
   ```
   onpassword -k my_secret_encryption_key_lmnop -e my_passwords.txt
   ```
   This example uses the new encryption key, **my_secret_encryption_key_lmnop**, and creates the encrypted `passwd_file` file in the `$ONEDB_HOME/etc` directory.
4. Redistribute `passwd_file` to all the database servers that the Connection Manager or cdr utility connects to, replacing the previous `$ONEDB_HOME/etc/passwd_file` files.
   If you update the `passwd_file` on multiple operating systems, you must run the onpassword utility on each type of operating system, and use the same text file and encryption key.

Starting Connection Managers on UNIX™ and Linux™

Use the oncmsm utility to start a Connection Manager.

About this task

A Connection Manager can be started before the database servers it manages are started. If a connection unit is managed by a Connection Manager, the Connection Manager connects to database servers in the connection unit after the database servers start.

If the database servers of a connection unit are online before a Connection Manager is initialized, start the Connection Manager, and then direct client application requests to the Connection Manager instead of the database servers.

1. Start the Connection Manager by running the oncmsm utility with the c parameter and the name of the Connection Manager configuration file.
   Example
   ```
   oncmsm -c configuration_file
   ```
   The default location for the configuration file is the `$ONEDB_HOME/etc` directory.
2. If you enabled Connection Manager logging by setting the LOG and LOGFILE parameters in the Connection Manager's configuration file, check the log to verify that the Connection Manager successfully started.
   Result
   The following message displays is the Connection Manager started:
   ```
   Connection Manager started successfully
   ```

Results
After a Connection Manager initializes, it attempts to connect to the database servers or groups of database servers that are specified by the ONEDB_SERVER parameter in the Connection Manager's configuration file. The Connection Manager then searches for and connects to all the database servers that are in each specified server's cluster, grid, or replicate set.

Starting Connection Managers on Windows™

Use the oncmsm utility to start a Connection Manager.

About this task

A Connection Manager can be started before the database servers it manages are started. If a connection unit is managed by a Connection Manager, the Connection Manager connects to database servers in the connection unit after the database servers start.

If the database servers of a connection unit are online before a Connection Manager is initialized, start the Connection Manager, and then direct client application requests to the Connection Manager instead of the database servers.

1. Install the Connection Manager as a service by running the oncmsm utility with the i parameter, the c parameter, and the name of the Connection Manager configuration file.
   Example
   ```
   oncmsm -i -c configuration_file
   ```
   The default location for the configuration file is the `%ONEDB_HOME%\etc` directory.

2. Initialize the Connection Manager by running the oncmsm utility with the name of the Connection Manager that is specified in the Connection Manager configuration file.

   Example

   ```
   oncmsm connection_manager_name
   ```

   Result

   The following message is displayed:

   ```
   Specify the user and password to run this service.
   Press <ENTER> to run Connection Manager as 'localsystem'
   ```

3. Enter the **informix** user ID and password, or press **Enter** to automatically create an **informix-admin** group and assign it access rights.
4. If you enabled Connection Manager logging by setting the LOG and LOGFILE parameters in the Connection Manager's configuration file, check the log to verify that the Connection Manager successfully started.

   Result

   The following message displays if the Connection Manager started:

   ```
   Connection Manager started successfully
   ```

Results

After a Connection Manager initializes, it attempts to connect to the database servers or groups of database servers that are specified by the ONEDB_SERVER parameter in the Connection Manager's configuration file. The Connection Manager then searches for and connects to all the database servers that are in each specified server's cluster, grid, or replicate set.

Stopping connection management

When you no longer want a Connection Manager to manage connection units, run the oncmsm utility to stop the Connection Manager instance.

About this task

If you are using multiple Connection Managers, you can run onstat -g cmsm to display the names of Connection Manager instances.

1. Log on to the computer on which the Connection Manager instance is running.
2. Run the oncmsm utility with the -k parameter.

   Example

   For example:

   ```
   oncmsm -k connection_manager_name
   ```

3. If quality of data (QOD) monitoring is turned on, and you want to stop it, run the cdr stop qod command on the master server.

   Example

   For example:

   ```
   cdr stop qod -c server_1
   ```

Results

The Connection Manager stops.

What to do next

To unistall a stopped Connection Manager from a Windows™ operating system, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -u connection_manager_name
```

Monitoring and troubleshooting connection management

Tools are available to monitor connection management, and help you diagnose potential problems.

The following options are available for connection management monitoring and troubleshooting:

- Set the LOG and LOGFILE parameters in Connection Manager configuration files. Log files contain information about service level agreements, failover configuration, and status information. The location of the log file is displayed when the Connection Manager is started.
- Run onstat -g cmsm to display information about Connection Manager instances.
- Set the CMALARMPROGRAM parameter in Connection Manager configuration files, and configure the `cmalarmprogram` script to handle event alarms.
- If the Connection Manager raises an event alarm:
  - The **ONEDB_ CMNAME** environment variable stores the name of the Connection Manager instance that raised the alarm.
  - The**ONEDB_ CMCONUNITNAM** environment variable stores the name of the Connection Manager connection unit that raised the alarm.

Strategies for increasing availability with Connection Managers

You can increase the resiliency of your client/server communication environment.

Install multiple network-interface cards (NICs) on client, Connection Manager, and database-server hosts

You can prevent a network-connectivity failure by installing multiple NICs on each host in a connection-management domain. If a NIC fails, the host can use other available NICs.

Install multiple Connection Managers

You can prevent a Connection Manager from becoming a single point of failure in your system by installing multiple Connection Managers to manage a domain.

Install Connection Managers separate from database servers

To prevent simultaneous Connection Manager and database server failure if a host fails, install Connection Managers on hosts that are not running database servers.

Use keywords in Connection Manager service-level-agreements

If a Connection Manager manages a high-availability cluster, you can use the cluster keywords to maintain connection consistency after failover. If database servers in a cluster switch roles after failover, the Connection Manager can use cluster keywords to continue directing client connection requests to the appropriate cluster-server type.

The cluster keywords are:

- HDR - High-availability data replication server
- RSS - Remote standalone secondary server
- SDS - Shared-disk secondary server
- PRI - Primary server
- PRIMARY - Primary server

Use group entries in sqlhosts files and Connection Manager configuration files

Create database-server groups in the host `sqlhosts` file of each Connection Manager that manages a high-availability cluster. Then, specify `sqlhosts` groups, rather than individual server names, in Connection Manager configuration files. If you specify database server names for a connection unit's ONEDB_SERVER parameter, and those specified servers are offline when a Connection Manager restarts, the Connection Manager is unable to reconnect to the cluster. If you specify a `sqlhosts` group for a connection unit's ONEDB_SERVER parameter, and at least one of the group's database servers is online when a Connection Manager restarts, the Connection Manager can reconnect to the cluster.

Specify the `c=1` option for `sqlhosts` group entries so that the connection-attempt starting point for a list of group members is random. For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     -          -                        c=1
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
 server_3       onsoctcp   host_3     port_3        g=my_servers
```

Configuration examples for connection management

The following examples show steps for setting up connection management for various connection units and various systems.

You can use these examples as a basis for developing your own connection-management system.

Example of configuring connection management for a high-availability cluster

This example shows steps that are required to configure connection management for a high-availability cluster.

About this task

For this example, you have a high-availability cluster on a trusted network. The cluster consists of three servers:

- A primary server (**server_1**)
- A shared-disk secondary server (**server_2**)
- An HDR secondary server (**server_3**)

The cluster supports the following application services:

- Online transaction processing (OLTP), which runs only on the primary server
- Payroll services, which can run on the primary server or HDR secondary server
- Reporting services, which can run on any of the secondary servers

Your system has the following needs:

- The database servers' workloads are balanced.
- The Connection Managers control failover.
- If failover occurs, the SD secondary server takes priority over the HDR secondary server.
- If the primary server fails, the Connection Managers can still connect to the cluster after restarting.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.

To configure connection management:

1. Install at least two network interface cards on each host.
   This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on.
   This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.

   You can install Connection Managers on application-server hosts if you want to prioritize an application server's connectivity to the primary cluster server.
3. On each host Connection Manager host, set the **ONEDB_HOME** environment to the directory the Connection Manager was installed into.
   Example
   Run the following command:

   ```
   setenv ONEDB_HOME path
   ```
4. Create a configuration file in each Connection Manager installation's `$ONEDB_HOME/etc` directory.
   Example

   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm1_log.log
   LOCAL_IP 192.0.2.0,192.0.2.1
   ```

```
CLUSTER cluster_1
{
   ONEDB_SERVER servers_1
   SLA oltp_1    DBSERVERS=primary
   SLA payroll_1 DBSERVERS=(PRI,HDR) \
                 POLICY=WORKLOAD
   SLA report_1  DBSERVERS=(SDS,HDR) \
                 POLICY=WORKLOAD
   FOC ORDER=ENABLED \
       PRIORITY=1
   CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
}
```

The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

```
NAME connection_manger_2
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm2_log.log
LOCAL_IP 192.0.2.2,192.0.2.3

CLUSTER cluster_1
{
   ONEDB_SERVER cluster_1
   SLA oltp_2    DBSERVERS=primary
   SLA payroll_2 DBSERVERS=(PRI,HDR)\
                 POLICY=WORKLOAD
   SLA report_2  DBSERVERS=(SDS,HDR) \
                 POLICY=WORKLOAD
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
}
```

The configuration file specifies the following information and behavior:

- Logging is enabled, and the log files are `$ONEDB_HOME/tmp/my_cm1_log.log` and `$ONEDB_HOME/tmp/my_cm2_log.log`.
- **connection_manager_1** monitors 192.0.2.0 and 192.0.2.1 and **connection_manager_2** monitors 192.0.2.2 and 192.0.2.3 for network failure.
- When the Connection Managers start, they each search their `sqlhosts` files for **cluster_1** entry, and then connect to the servers in that group.
- `CONNECT TO @oltp_1` and `CONNECT TO @oltp_2` connection requests are directed to the primary server.
- `CONNECT TO @payroll_1` and `CONNECT TO @payroll_2` connection requests are directed to whichever of the primary and HDR secondary servers has the lowest workload.
- `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the secondary server that has the lowest workload.
- The connection between **connection_manager_1** and the primary server is prioritized over the connection between **connection_manager_2** and the primary server. Failover that would break the connectivity between **connection_manager_1** and the primary server is blocked.
- If failover processing fails after eight attempts, `$ONEDB_HOME/etc/CMALARMPROGRAM.sh` is called.

Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:

- The EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for primary-server events before failover processing begins.
- The MODE attributes of the SLA parameters are not set, so the Connection Managers return connection information for **server_1**, **server_2**, and **server_3** to client applications, rather than acting as proxy servers.
- The SECONDARY_EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for secondary-server events before the Connection Manager disconnects from the secondary server.
- The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote `sqlhosts` files.
- The SQLHOSTS parameter is not set, so each Connection Manager first searches its local `sqlhosts` file, and then remote database server `sqlhosts` files for connectivity information related to **server_1**, **server_2**, and **server_3**.
- The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.

5. Set the `onconfig` file DRAUTO configuration parameter on all database servers to `3`

   Example

   ```
   DRAUTO 3
   ```

   This setting specifies that a Connection Manager controls failover arbitration.
6. Set the `onconfig` file HA_FOC_ORDER configuration parameter on **server_1** to `SDS,HDR`

   Example

   ```
   HA_FOC_ORDER SDS,HDR
   ```

   After the Connection Managers start, and connect to **server_1**, the HA_FOC_ORDER value replaces the value of the `ORDER` attributes in each Connection Manager's configuration file.

   If **server_1** fails, the Connection Managers attempt failover to the SD secondary server. If the SD secondary server is also unavailable, the Connection Managers attempt failover to the HDR secondary server.
7. **Optional:** Configure the `cmalarmprogram` script on each Connection Manager host.
   Event alarms can be sent to specified email addresses.
8. Add entries to the `sqlhosts` files on **server_1** and **server_2**'s host and on **server_3**'s host.

   Example

   ```
   #dbservername  nettype    hostname   servicename   options
    server_1      onsoctcp   host_1     port_1
    server_2      onsoctcp   host_1     port_2
    server_3      onsoctcp   host_2     port_3
   ```
9. Create a `sqlhosts` file on each Connection Manager.

   Example

   ```
   #dbservername  nettype    hostname   servicename   options
    cluster_1     group      -          -             c=1,e=server_3
    server_1      onsoctcp   host_1     port_1        g=cluster_1
    server_2      onsoctcp   host_1     port_2        g=cluster_1
    server_3      onsoctcp   host_2     port_3        g=cluster_1
   ```

If a Connection Manager restarts after a primary-server failure, it is able to connect to other database servers in the cluster because the **cluster_1** group is defined.

10. Create a `sqlhosts` file on each client host.

    Example

    ```
    #dbservername  nettype    hostname   servicename   options
     oltp          group      -          -             c=1,e=oltp_2
     oltp_1        onsoctcp   cm_host_1  cm_port_1     g=oltp
     oltp_2        onsoctcp   cm_host_2  cm_port_2     g=oltp

     report        group      -          -             c=1,e=report_2
     report_1      onsoctcp   cm_host_1  cm_port_3     g=report
     report_2      onsoctcp   cm_host_2  cm_port_4     g=report

     payroll       group      -          -             c=1,e=payroll_2
     payroll_1     onsoctcp   cm_host_1  cm_port_5     g=payroll
     payroll_2     onsoctcp   cm_host_2  cm_port_6     g=payroll
    ```

    If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **oltp**, **report**, and **payroll** groups are defined.

    - `CONNECT TO @oltp` connection requests are directed through one of the Connection Managers to the primary server.
    - `CONNECT TO @payroll` connection requests are directed through one of the Connection Managers to whichever of the primary and HDR secondary servers has the lowest workload.
    - `CONNECT TO @report` connection requests are directed through one of the Connection Managers to the secondary server that has the lowest workload.
11. Set each **ONEDB_ SQLHOSTS** environment variable to the `sqlhosts` file location by running the setenv command on each Connection Manager and client host.

    Example

    ```
    setenv ONEDB_ SQLHOSTS path_and_file_name
    ```
12. Run the oncmsm utility on each Connection Manager host, to start each Connection Manager.

    Example

    On the host of **connection_manager_1**:

    ```
    oncmsm -c cm_1.cfg
    ```

    On the host of **connection_manager_2**:

    ```
    oncmsm -c cm_2.cfg
    ```
13. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Example of configuring connection management for a grid or replicate set

You can use a Connection Manager to route client connections for the replication servers of a grid or replicate set.

About this task

For this example, you have a replicate set that consists of four replication servers:

- **server_1**
- **server_2**
- **server_3**
- **server_4**

The replication set supports reporting services, which can run on any of the replication servers.

Your system has the following needs

- Client requests are directed to the replication server with the fewest apply failures.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.

To configure connection management:

1. Install at least two network interface cards on each host.
   This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on.
   This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.
3. On each Connection Manager host, set the **ONEDB_HOME** environment to the directory the Connection Manager was installed into.
   Example
   Run the following command:
   ```
   setenv ONEDB_HOME path
   ```
4. Create a configuration file in each Connection Manager installation's `$ONEDB_HOME/etc` directory.
   Example

   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:
   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm1_log.log

   REPLSET replicate_set_1
   {
    ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
    SLA report_1 DBSERVERS=ANY \
                POLICY=FAILURE
   }
   ```
   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:
   ```
   NAME connection_manger_2
   LOG 1
   ```

```
LOGFILE $ONEDB_HOME/tmp/my_cm2_log_.log

REPLSET replicate_set_1
{
 ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
 SLA report_2 DBSERVERS=ANY \
             POLICY=FAILURE
}
```

The configuration file specifies the following information and behavior:

- Logging is enabled, and the log files are `$ONEDB_HOME/tmp/my_cm1_log.log` and `$ONEDB_HOME/tmp/my_cm2_log.log`.
- When the Connection Managers start, they each search their `sqlhosts` files for **g_server_1**, **g_server_2**, **g_server_3**, and **g_server_4** entries, and then connect to the servers **server_1**, **server_2**, **server_3**, and **server_4** that are in those groups.
- `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the replication server that has the fewest apply failures.

Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:

- The MODE attributes of the SLA parameters are not set, so the Connection Managers return connection information for **server_1**, **server_2**, **server_3**, and **server_4** to client applications, rather than acting as proxy servers.
- The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote `sqlhosts` files.
- The SQLHOSTS parameter is not set, so each Connection Manager first searches its local `sqlhosts` file, and then remote database server `sqlhosts` files for connectivity information related to **server_1**, **server_2**, **server_3**, and **server_4**.
- The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.

5. Add entries to the`sqlhosts` files on the hosts of each database server, **connection_manger_1**, and **connection_manger_2**.

Example

```
#dbservername  nettype   hostname   servicename  options
 g_server_1    group     -          -            i=1,e=server_1
 server_1      onsoctcp  host_1     port_1       g=g_server_1

 g_server_2    group     -          -            i=2,e=server_2
 server_2      onsoctcp  host_2     port_2       g=g_server_2

 g_server_3    group     -          -            i=3,e=server_3
 server_3      onsoctcp  host_3     port_3       g=g_server_3

 g_server_4    group     -          -            i=4,e=server_4
 server_4      onsoctcp  host_4     port_4       g=g_server_4
```

6. Create a `sqlhosts` file on each client host.

Example

```
#dbservername  nettype    hostname   servicename   options
 report        group      -          -             c=1,e=report_2
 report_1      onsoctcp   cm_host_1  cm_port_3     g=report
 report_2      onsoctcp   cm_host_2  cm_port_4     g=report
```

If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **report** group is defined.

`CONNECT TO @report` connection requests are directed through one of the Connection Managers to the replication server that has the fewest apply failures.

7. Set each **ONEDB_ SQLHOSTS** environment variable to the `sqlhosts` file location by running the setenv command on each Connection Manager and client host.
   Example
   ```
   setenv ONEDB_ SQLHOSTS path_and_file_name
   ```
8. Turn on quality of data (QOD) monitoring by running the cdr define qod command.
   Example
   ```
   cdr define qod -c server_1 --start
   ```
   The command connects to **server_1**, defines **server_1** as a master server for monitoring data, and then turns on quality of data monitoring.

   server_1 maintains a failed-transaction count for the servers in the replicate set. The failed-transaction count determines which replication server the Connection Managers send a client connection requests to.
9. Run the oncmsm utility on each Connection Manager host, to start each Connection Manager.
   Example

   On the host of **connection_manager_1**:
   ```
   oncmsm -c cm_1.cfg
   ```
   On the host of **connection_manager_2**:
   ```
   oncmsm -c cm_2.cfg
   ```
10. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Example of configuring connection management for a high-availability replication system

You can use a Connection Manager to route client connections for the participants of a replicate set and to control failover for high-availability clusters that participate in Enterprise Replication.

About this task

For this example, you have a grid that consists of four nodes. One of the nodes is a primary server in a high-availability cluster that consists of a primary server, an SD secondary server, an HDR secondary server, and an RS secondary server:

- **server_1a** - ER Node 1, primary server
- **server_1b** - SD secondary server

- **server_1c** - HDR secondary server
- **server_1d** - RS secondary server
- **server_2** - ER Node 2
- **server_3** - ER Node 3
- **server_4** - ER Node 4

The grid supports reporting services, which can run on any of the ER nodes.

Your system has the following needs

- Client requests are directed to the ER node with the lowest transaction latency.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.
- The Connection Managers control failover for the cluster.
- If failover occurs, it the SD secondary server takes priority over the HDR secondary server. The HDR secondary server takes priority over the RS secondary server.
- If the primary server fails, the Connection Managers can still connect to the cluster after restarting.

To configure connection management:

1. Install at least two network interface cards on each host.
   This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on.
   This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.
3. On each Connection Manager host, set the **ONEDB_HOME** environment to the directory the Connection Manager was installed into.
   Example
   Run the following command:
   ```
   setenv ONEDB_HOME path
   ```
4. Create a configuration file in each Connection Manager installation's `$ONEDB_HOME/etc` directory.
   Example

   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:
   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm1_log.log
   LOCAL_IP 192.0.0.2,192.0.2.1

   REPLSET replicate_set_1
   {
    ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
    SLA report_1 DBSERVERS=ANY \
                POLICY=LATENCY
   ```

```
}

CLUSTER cluster_1
{
   ONEDB_SERVER g_server_1
   FOC ORDER=ENABLED \
       PRIORITY=1
   CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
}
```

The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

```
NAME connection_manger_2
LOG 1
LOGFILE $ONEDB_HOME/tmp/my_cm2_log_.log
LOCAL_IP 192.0.2.2,192.0.2.3

REPLSET replicate_set_1
{
 ONEDB_SERVER g_server_1,g_server_2,g_server_3,g_server_4
 SLA report_2 DBSERVERS=ANY \
             POLICY=LATENCY
}
CLUSTER cluster_1
{
   ONEDB_SERVER g_server_1
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
}
```

The configuration file specifies the following information and behavior:

- Logging is enabled, and the log files are `$ONEDB_HOME/tmp/my_cm1_log.log` and `$ONEDB_HOME/tmp/my_cm2_log.log`.
- **connection_manager_1** monitors 192.0.2.0 and 192.0.2.1 and **connection_manager_2** monitors 192.0.2.2 and 192.0.2.3 for network failure.
- When the Connection Managers start, they each search their `sqlhosts` files for **g_server_1**, **g_server_2**, **g_server_3**, and **g_server_4** entries, and then connect to the servers **server_1a**, **server_1b**, **server_1c**, **server_1d**,**server_2**, **server_3**, and **server_4** that are in those groups.
- `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the replication server that has the lowest transaction latency.
- The connection between **connection_manager_1** and the primary server is prioritized over the connection between **connection_manager_2** and the primary server. Failover that would break the connectivity between **connection_manager_1** and the primary server is blocked.
- If failover processing fails after eight attempts, `$ONEDB_HOME/etc/CMALARMPROGRAM.sh` is called.

Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:

- The EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for primary-server events before failover processing begins. The SECONDARY_EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for secondary-server events before the Connection Manager disconnects from the secondary server.

    - The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote `sqlhosts` files.
    - The SQLHOSTS parameter is not set, so each Connection Manager first searches its local `sqlhosts` file, and then remote database server `sqlhosts` files for connectivity information related to **server_1**, **server_2**, **server_3**, and **server_4**.
    - The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.

5. Set the `onconfig` file DRAUTO configuration parameter on **server_1a**, **server_1b**, **server_1c**, and **server_1d** to `3`
   Example
   ```
   DRAUTO 3
   ```
   This setting specifies that a Connection Manager controls failover arbitration.
6. Set the `onconfig` file HA_FOC_ORDER configuration parameter on **server_1a** to `SDS,HDR,RSS`
   Example
   ```
   HA_FOC_ORDER SDS,HDR,RSS
   ```
   After the Connection Managers start, and connect to **server_1a**, the HA_FOC_ORDER value replaces the value of the `ORDER` attributes in each Connection Manager's configuration file.

   If **server_1a** fails, the Connection Managers attempt failover to the SD secondary server. If the SD secondary server is also unavailable, the Connection Managers attempt failover to the HDR secondary server. If the HDR secondary server is also unavailable, the Connection Managers attempt failover to the RS secondary server.
7. Add entries to the`sqlhosts` files on the hosts of each database server and Connection Manager.
   Example
   ```
   #dbservername  nettype   hostname  servicename  options
    g_server_1    group     -         -            i=1,c=1,e=server_1d
    server_1a     onsoctcp  host_1    port_1       g=g_server_1
    server_1b     onsoctcp  host_1    port_2       g=g_server_1
    server_1c     onsoctcp  host_2    port_3       g=g_server_1
    server_1d     onsoctcp  host_3    port_4       g=g_server_1

    g_server_2    group     -         -            i=2,e=server_2
    server_2      onsoctcp  host_4    port_5       g=g_server_2

   g_server_3    group     -          -            i=3,e=server_3
    server_3      onsoctcp  host_5    port_6       g=g_server_3

    g_server_4    group     -         -            i=4,e=server_4
    server_4      onsoctcp  host_6    port_7       g=g_server_4
   ```
8. Create a `sqlhosts` file on each client host.
   Example
   ```
   #dbservername  nettype    hostname   servicename   options
    report        group      -          -             c=1,e=report_2
    report_1      onsoctcp   cm_host_1  cm_port_3     g=report
    report_2      onsoctcp   cm_host_2  cm_port_4     g=report
   ```

If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **report** group is defined.

`CONNECT TO @report` connection requests are directed through one of the Connection Managers to the replication server that has the lowest transaction latency.

9. Set each **ONEDB_ SQLHOSTS** environment variable to the `sqlhosts` file location by running the setenv command on each Connection Manager and client host.
   Example

   ```
   setenv ONEDB_ SQLHOSTS path_and_file_name
   ```
10. Turn on quality of data (QOD) monitoring by running the cdr define qod command.
   Example

   ```
   cdr define qod -c server_1a --start
   ```

   The command connects to **server_1a**, defines **server_1a** as a master server for monitoring data, and then turns on quality of data monitoring.

   server_1a monitors transaction latency for the replication servers in the grid.
11. Run the oncmsm utility on each Connection Manager host, to start each Connection Manager.
   Example

   On the host of **connection_manager_1**:

   ```
   oncmsm -c cm_1.cfg
   ```

   On the host of **connection_manager_2**:

   ```
   oncmsm -c cm_2.cfg
   ```
12. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Example: Configuring connection management for untrusted networks

This example shows steps that are required to configure connection management for an untrusted network.

About this task

For this example, you have a high-availability cluster on an untrusted network. All hosts use UNIX™ operating systems. The cluster consists of four servers:

- A primary server (**server_1**)
- A shared-disk secondary server (**server_2**)
- An HDR secondary server (**server_3**)
- An RS secondary server (**server_4**)

To configure connection management:

1. Install at least two network interface cards on each host.
2. Install at least two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on.
3. On each host Connection Manager host, set the **ONEDB_HOME** environment to the directory the Connection Manager was installed into.

   Example

   Run the following command:

   ```
   setenv ONEDB_HOME path
   ```
4. Create a configuration file in each Connection Manager installation's `$ONEDB_HOME/etc` directory.

   Example

   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm1_log.log
   LOCAL_IP 192.0.2.0,192.0.2.1

   CLUSTER cluster_1
   {
      ONEDB_SERVER cluster_1
      SLA oltp_1    DBSERVERS=primary
      SLA payroll_1 DBSERVERS=(PRI,HDR) \
                    POLICY=WORKLOAD
      SLA report_1  DBSERVERS=(SDS,HDR,RSS) \
                    POLICY=WORKLOAD
      FOC ORDER=ENABLED \
          PRIORITY=1
      CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
   }
   ```

   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

   ```
   NAME connection_manger_2
   LOG 1
   LOGFILE $ONEDB_HOME/tmp/my_cm2_log.log
   LOCAL_IP 192.0.2.2,192.0.2.3

   CLUSTER cluster_1
   {
      ONEDB_SERVER cluster_1
      SLA oltp_2    DBSERVERS=primary
      SLA payroll_2 DBSERVERS=(PRI,HDR)\
                    POLICY=WORKLOAD
      SLA report_2  DBSERVERS=(SDS,HDR,RSS) \
                    POLICY=WORKLOAD
      FOC ORDER=ENABLED \
          PRIORITY=2
      CMALARMPROGRAM $ONEDB_HOME/etc/CMALARMPROGRAM.sh
   }
   ```
5. Set the `onconfig` file DRAUTO configuration parameter on all database servers to `3`, to specify that Connection Managers control failover arbitration.

Example

```
DRAUTO 3
```

6. Set the `onconfig` file HA_FOC_ORDER configuration parameter on **server_1** to `SDS,HDR,RSS`

   Example

```
HA_FOC_ORDER SDS,HDR,RSS
```

7. **Optional:** Configure the `cmalarmprogram` script on each Connection Manager host.
8. Add entries to the`sqlhosts` files on **server_1** and **server_2**'s host, **server_3**'s host, and **server_4**'s host.

   Example

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 a_server_1     onsoctcp   host_1     port_2

 server_2       onsoctcp   host_1     port_3        s=6
 a_server_2     onsoctcp   host_1     port_4

 server_3       onsoctcp   host_2     port_5        s=6
 a_server_3     onsoctcp   host_2     port_6

 server_4       onsoctcp   host_3     port_7        s=6
 a_server_4     onsoctcp   host_3     port_8
```

9. Create a `sqlhosts` file on each Connection Manager's host.

   Example

```
#dbservername   nettype    hostname   servicename   options
 cluster_1      group      -          -             c=1,e=a_server_4
 server_1       onsoctcp   host_1     port_1        s=6,g=cluster_1
 a_server_1     onsoctcp   host_1     port_2        g=cluster_1
 server_2       onsoctcp   host_1     port_3        s=6,g=cluster_1
 a_server_2     onsoctcp   host_1     port_4        g=cluster_1
 server_3       onsoctcp   host_2     port_5        s=6,g=cluster_1
 a_server_3     onsoctcp   host_2     port_6        g=cluster_1
 server_4       onsoctcp   host_3     port_7        s=6,g=cluster_1
 a_server_4     onsoctcp   host_3     port_8        g=cluster_1
```

10. In each database server's `onconfig` file, set the DBSERVERALIASES parameter to that database server's alias.

    Example

    The `onconfig` file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

The `onconfig` file entry for **server_2**:

```
DBSERVERALIASES a_server_2
```

The `onconfig` file entry for **server_3**:

```
DBSERVERALIASES a_server_3
```

The `onconfig` file entry for **server_4**:

```
DBSERVERALIASES a_server_4
```

11. On one of the Connection Manager hosts, use a text editor to create an ASCII-text password file that contains security information. Save the file to the `$ONEDB_HOME/tmp` directory.

    Example

    For example, `my_passwords.txt` has the following entries:

    ```
    cluster_1   a_server_1  user_1  password_1
    cluster_1   a_server_2  user_2  password_2
    cluster_1   a_server_3  user_3  password_3
    cluster_1   a_server_4  user_4  password_4

    server_1    a_server_1  user_1  password_1
    server_2    a_server_2  user_2  password_2
    server_3    a_server_3  user_3  password_3
    server_4    a_server_4  user_4  password_4

    a_server_1  a_server_1  user_1  password_1
    a_server_2  a_server_2  user_2  password_2
    a_server_3  a_server_3  user_3  password_3
    a_server_4  a_server_4  user_4  password_4
    ```

12. On the host where the password file is saved, run the onpassword utility with a specified encryption key to encrypt the password and create `passwd_file` in the `$ONEDB_HOME/etc` directory.

    Example

    For example, run the following command, specifying **my_secret_encryption_key_456** as your encryption key:

    ```
    onpassword -k my_secret_encryption_key_456 -e my_passwords.txt
    ```

13. Store the original text file and encryption key in a safe place.
14. Distribute `$ONEDB_HOME/etc/passwd_file` to all the database servers that Connection Managers connect to, and to all Connection Managers.

    For systems that use Enterprise Replication, also distribute `$ONEDB_HOME/etc/passwd_file` to all the database servers that the cdr utility connects to.
15. Create a `sqlhosts` file on each client host.

    Example

    ```
    #dbservername  nettype    hostname   servicename   options
     oltp          group      -          -             c=1,e=oltp_2
     oltp_1        onsoctcp   cm_host_1  cm_port_1     g=oltp
     oltp_2        onsoctcp   cm_host_2  cm_port_2     g=oltp

     report        group      -          -             c=1,e=report_2
     report_1      onsoctcp   cm_host_1  cm_port_3     g=report
     report_2      onsoctcp   cm_host_2  cm_port_4     g=report

     payroll       group      -          -             c=1,e=payroll_2
     payroll_1     onsoctcp   cm_host_1  cm_port_5     g=payroll
     payroll_2     onsoctcp   cm_host_2  cm_port_6     g=payroll
    ```

16. Set each **ONEDB_ SQLHOSTS** environment variable to the `sqlhosts` file location by running the setenv command on each Connection Manager and client host.

    Example

    ```
    setenv ONEDB_ SQLHOSTS path_and_file_name
    ```

17. Run the oncmsm utility on each Connection Manager host, to start each Connection Manager.
    Example

    On the host of **connection_manager_1**:

    ```
    oncmsm -c cm_1.cfg
    ```

    On the host of **connection_manager_2**:

    ```
    oncmsm -c cm_2.cfg
    ```
18. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Example of configuring connection management for prioritizing connections and network monitoring

You can install Connection Managers on the hosts of application servers, and then prioritize the connections between specific application servers and the primary server of a high-availability cluster. This configuration allows the highest priority application server to maintain its connection to the cluster's primary server if a portion of the network fails.

About this task

You can configure failover for the following conditions:

- When the primary server becomes inoperative.
- When an application server loses network connectivity with the primary server.

If network monitoring and failover priority are enabled and a network failure occurs, an application server that loses connectivity to the primary server but maintains connectivity to a secondary server can, through a shared-host Connection Manager, initiate failover to the secondary server. If, however, failover would cause an application server with more priority to lose connectivity to the primary server, the Connection Managers block failover.

Network monitoring and failover priority are enabled by setting the following parameters and attributes in each Connection Manager's configuration file:

- The Connection Manager's LOCAL_IP parameter
- A CLUSTER connection-unit's SLA parameters
- A CLUSTER connection-unit's FOC parameter
- The FOC parameter's ORDER attribute
- The FOC parameter's PRIORITY attribute

1. Install at least two network interface cards on each host.
   This method prevents the failure of a network interface card from causing client, Connection Manager, or database server connectivity failure.
2. Install and configure Connection Managers on each application server's host.

a. Set the LOCAL_IP parameter in each Connection Manager configuration file to the IP addresses of the host's NIC cards.
   Example
   For example:

   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manager_1
   LOCAL_IP 192.0.2.0,192.0.2.1
   ```

   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

   ```
   NAME connection_manager_2
   LOCAL_IP 192.0.2.2,192.0.2.3
   ```

b. Create service-level agreements for each Connection Manager.
   Example
   For example:

   cm_1.cfg now has the following entries:

   ```
   NAME connection_manager_1
   LOCAL_IP 192.0.2.0,192.0.2.1

   CLUSTER my_cluster

   {
      ONEDB_SERVER my_servers
      SLA sla_1 DBSERVERS=PRI
      SLA sla_2 DBSERVERS=SDS,HDR,RSS
   }
   ```

   cm_2.cfg now has the following entries:

   ```
   NAME connection_manager_1
   LOCAL_IP 192.0.2.2,192.0.2.3

   CLUSTER my_cluster

   {
      ONEDB_SERVER my_servers
      SLA sla_3 DBSERVERS=PRI
      SLA sla_4 DBSERVERS=SDS,HDR,RSS
   }
   ```

c. Specify each application server's priority by setting the FOC parameter's PRIORITY attribute for each shared-host Connection Manager.
   A PRIORITY value must be a positive integer and unique among all the Connection Managers that are configured to manage a specific cluster. If you specify a PRIORITY value in a connection-unit definition, you must set the ORDER attribute to ENABLED, and specify the failover order in the primary server's HA_FOC_ORDER configuration parameter.
   Example

For example:

The primary server has the following `onconfig` file entry:

```
HA_FOC_ORDER SDS,HDR,RSS
```

cm_1.cfg now has the following entries:

```
NAME connection_manager_1
LOCAL_IP 192.0.2.0,192.0.2.1

CLUSTER my_cluster

{
   ONEDB_SERVER my_servers
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=SDS,HDR,RSS
   FOC ORDER=ENABLED PRIORITY=1
}
```

cm_2.cfg now has the following entries:

```
NAME connection_manager_2
LOCAL_IP 192.0.2.2,192.0.2.3

CLUSTER my_cluster

{
   ONEDB_SERVER my_servers
   SLA sla_3 DBSERVERS=PRI
   SLA sla_4 DBSERVERS=SDS,HDR,RSS
   FOC ORDER=ENABLED PRIORITY=2
}
```

3. Create `sqlhosts` files that contain network connectivity information for each Connection Manager and database server host.
4. **Optional:** Create a password file if you configure secure ports for database servers.
5. Run the oncmsm utility on each Connection Manager host, to start each Connection Manager.
   Example

   On the host of **connection_manager_1**:

   ```
   oncmsm -c cm_1.cfg
   ```

   On the host of **connection_manager_2**:

   ```
   oncmsm -c cm_2.cfg
   ```

6. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Results

The Connection Managers now initiate failover if a network failure occurs, and failover would not cause a higher-priority application server to lose its connectivity to the cluster's primary server.

Example of configuring for an SSL connection

The examples in this section shows the steps to configure Connection Manager to listen for SSL connections from database clients.

About this task

Connection Manager uses OpenSSL to implement the SSL functionality. The following example demonstrates how to use the OpenSSL utility to create the needed keystores and how to configure CM to use them.

Configuring a CM to connect to OneDB server using SSL

To configure a CM to connect to OneDB server using SSL, follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections on page .

Configuring a client to connect to a CM using SSL

A client can be configured to connect to a CM via SSL by following the same steps used to configure a client to connect to an OneDB server via SSL. In this case, the CM acts like a server. Follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections on page .

Example: Using the OpenSSL encryption library

This example shows the steps to configure CM to listen for SSL connection using the OpenSSL encryption library.

About this task

Use the openssl utility of your OpenSSL installation.

1. Create a self-signed certificate and corresponding private key in a PEM file:

   Create a private key

   Example

   ```
   $ openssl genrsa -out cm1key.pem
   ```

   Create the self-signed certificate using the private key

   Example

   ```
   $ openssl req -new -x509 -key cm1key.pem -subj "CN=`hostname`" -days 3650 -out cm1cert.pem
   ```

   Put the private key and the self-signed certificate into a single PEM file

   Example

   ```
   $ cat cm1key.pem cm1cert.pem > filewithcertificatetoimport.pem
   ```

2. Create the keystore file to contain the private key and certificate that are contained in a PEM file:

   Example

   ```
   $ openssl pkcs12 -export -in filewithcertificatetoimport.pem -name cm1ListeningCert -passout pass:test
    -out cm1.p12
   ```

3. Create the stash file to contain the encrypted keystore password:

   Example

```
onkstash cm1.p12 test
```

4. In cm1's config file set "SSL_LABEL" to the certificate's label:

   Example

   ```
   SSL_LABEL cm1ListeningCert
   ```

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 for high-availability clusters

A failure in a high-availability cluster means that one of the servers is no longer available. If a primary server fails, you must promote a secondary server to the role of primary server. Connection Managers can be configured to perform automatic failover.

You must set the DRAUTO configuration parameter to specify how failover is performed, and then use one of the following failover options:

- Connection Manager
- ISV cluster management software
- Manual switchover

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 `$ONEDB_HOME/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 *HCL OneDB™ 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 HCL OneDB™ 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.

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.

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:

- Equipment failure or destruction
- A network failure
- An excessive processing delay on one of the database servers

The database server interprets either of the following conditions as a cluster failure:

- The DRTIMEOUT configuration parameter value was exceeded without confirmation of communication with other cluster servers.
- A database server in the cluster does not respond to the periodic messaging attempts over the network. Cluster servers ping each other even if the primary server does not send records to the secondary database servers.

  A cluster server pings other cluster servers at the interval specified by its DRTIMEOUT configuration parameter.

After a database server detects a cluster failure, it writes a message to its message log (for example, `DR: receive error`) and turns off data replication. If a cluster failure occurs, the connection between the two database servers is dropped and the secondary database server remains in read-only mode.

You can configure automatic switchover for HDR replication pairs by setting the DRAUTO configuration parameter to 1 or 2.

You can configure automatic failover for a high-availability cluster by configuring Connection Managers. Connection Managers have many advantages over automatic switchover, and can manage failover to SD and RS secondary servers, as well.

Automatic switchover

If the primary server in a HDR-availability system fails, the HDR secondary server can be automatically converted to either a standard or primary database server.

If automatic switchover occurs, the HDR secondary server to rolls back open transactions, and then switches to online mode as either a primary or standard database server. Automatic switchover does not redirect client applications the new primary or standard database server.

Automatic switchover can be a better option than manual failover, but using Connection Managers to control failover for a high-availability cluster has even more advantages.

Automatic switchover requires a very stable network, and works only with primary servers and HDR secondary servers. Connection Managers can be configured to be tolerant of network instability, and can perform failover to HDR, RS, and SD secondary servers. Connection Managers can also prioritize connections between application servers and the primary server if a network failure occurs.

Automatic switchover does not redirect client connections from the old primary server to the new standard or primary server. Connection Managers can redirect clients to whichever secondary server becomes the primary server.

If you use automatic switchover, you must ensure that logical-log files are backed up, and that the HDR server has enough logical-log disk space to allow processing to continue without backing up logical-log files.

To configure automatic switchover or Connection Manager failover, you must set the DRAUTO configuration parameter on all database servers of a high-availability cluster.

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. 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.

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.

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 HCL OneDB™ Client Software Development Kit (Client SDK) have specific mechanisms for automating redirection. For details, see the HCL OneDB™ 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 **ONEDB_SERVER** environment variable

The mechanism that you employ determines which CONNECT syntax you can use in your application.

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 **ONEDB_SERVER** 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 **ONEDB_SERVER** environment variable if their **DBPATH** environment variable is set to the other database server in the pair. Their **ONEDB_SERVER** 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:

```
ONEDB_SERVER  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 *HCL OneDB™ 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 537, 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 **ONEDB_SERVER** 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 **ONEDB_ SQLHOSTS** environment variable specifies the full path name and file name of the connection information in `$ONEDB_HOME/etc/sqlhosts`. For more information about **ONEDB_ SQLHOSTS**, see the *HCL OneDB™ Guide to SQL: Reference*.

On Windows™, the connectivity information is in a key in the Windows™ registry.

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.

About this task

To change the connectivity information about the client computer:

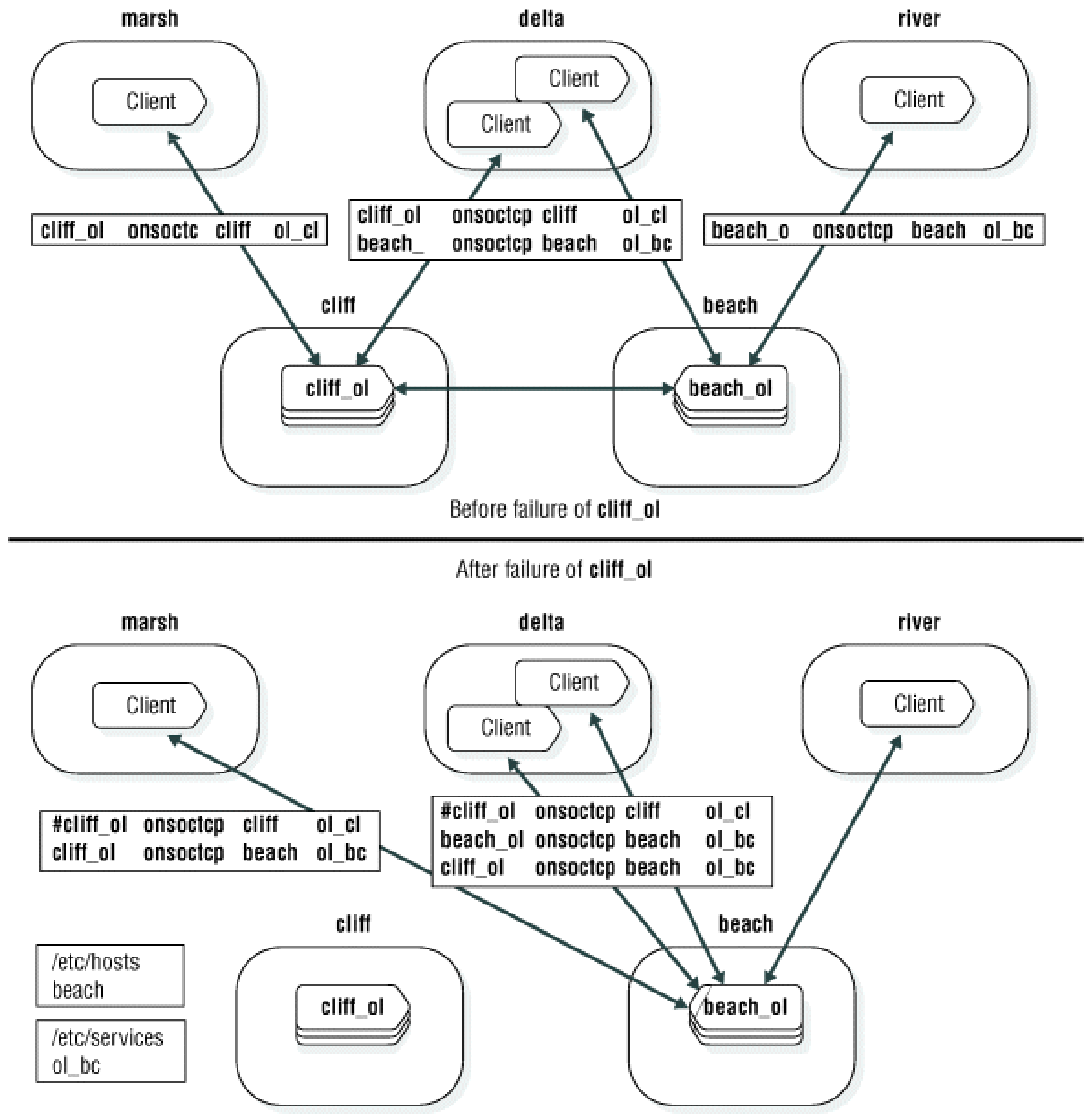
1. In the `sqlhosts` file or registry, comment out the entry for the failed database server.
2. Add an entry that specifies the dbservername of the failed database server in the **servername** field and information for the database server to which you are redirecting clients in the **nettype**, **hostname**, 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.

Results

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.

Figure 80. Connectivity values before and after a failure of the cliff_ol database server


marsh
delta
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
beach
cliff_ol
beach_ol
Before failure of cliff_ol
After failure of cliff_ol
#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
/etc/hosts
beach
/etc/services
ol_bc

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 ONEDB_SERVER environment variable

The **ONEDB_SERVER** environment variable redirection method can be used when an application does not explicitly specify a database server in the CONNECT statement, so that the client connects to the database server that the **ONEDB_SERVER** environment variable specifies.

If one of the database servers in a cluster is unusable, applications that use that database server can reset their **ONEDB_SERVER** environment variable to the another database server in the cluster to access the same data.

Applications read the value of the **ONEDB_SERVER** 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 **ONEDB_SERVER** 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 **ONEDB_SERVER** environment variable.

To redirect clients with the **ONEDB_SERVER** environment variable:

1. Quit their applications.
2. Change their **ONEDB_SERVER** 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 ONEDB_SERVER 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 tables summarize the differences among redirection mechanisms:

Table 64. Redirection methods for DBPATH connectivity

DBPATH	Automatic redirection	User redirection
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

Table 65. Redirection methods for Connectivity information

Connectivity information	Automatic redirection	User redirection
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

Table 66. Redirection methods for ONEDB_HOME connectivity

ONEDB_HOME	Automatic redirection
When is a client redirected?	When the client restarts and reads a new value for the **ONEDB_SERVER** 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

Table 67. Redirection methods for Connection Manager connectivity

Connection Manager	Automatic redirection
When is a client redirected?	When the configured service level agreement (SLA) 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.

Recovering a cluster after critical data is damaged

If one of the database servers in a high-availability 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 as is being started 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`, then convert the server to the primary server by running the onmode -d make primary command.
   - If it is set to `2`, 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 545.

Table 68. 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

Table 68. Steps for restarting HDR after a critical media failure on the primary database server (continued)

Step	On the primary database server (svr_A)	On the secondary database server (svr_B)
2.	ON-Bar command onbar -r -p	
3.	onmode command onmode -d secondary srv_B	
4.	command 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 a high-availability 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 high-availability cluster for the first time.

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
```

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 69. 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.		command 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.

About this task

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:

   ```
   $ONEDB_HOME/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:

   ```
   $ONEDB_HOME/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 70. 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.	

Table 70. Steps to restart if you changed the secondary database server to standard (continued)

Step	On the primary database server	On the secondary database server
	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).	

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 71. 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).	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. ON-Bar command onbar -r -l	

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.

Critical data is damaged

About this task
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 utility: onbar -r
   Result
   The primary server restarts after the restore is complete.
2. Restart the SD secondary servers.

Results
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

About this task
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 utility: onbar -r 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.

About this task

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 388 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**.

Distributed data

Multiphase commit protocols

A *two-phase commit protocol* ensures that transactions are uniformly committed or rolled back across multiple database servers. You can use HCL® OneDB® database servers with transaction managers to manipulate data in non-HCL OneDB™ databases. Distributed queries across HCL® OneDB® database servers support two-phase commit.

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 Manually recovering from failed two-phase commit on page 571.

These topics also contain information about using transaction support for XA-compliant, external data sources, which can participate in two-phase commit transactions. See HCL OneDB transaction support for XA-compliant, external data sources on page 550.

Transaction managers

Transaction managers support two-phase commit and roll back. For example, if your database is HCL® OneDB®, 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 HCL® OneDB® or non-HCL OneDB™ 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 (HCL® OneDB® 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 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 HCL® OneDB® and non-HCL OneDB™ 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 HCL OneDB™ Client Software Development Kit, the latest version of HCL OneDB™ ODBC Driver, and MTS/XA. MTS/XA works on Windows™. For more information, contact HCL® OneDB® Technical Support, and see the *HCL OneDB™ Client Products Installation Guide* and the MTS/XA documentation.

HCL OneDB™ transaction support for XA-compliant, external data sources

The HCL OneDB™ Transaction Manager, which is an integral part of HCL OneDB™, 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 extension is an example of a set of UDRs that provide this type of external data access.
- Register XA-compliant, external data sources with HCL OneDB™.
- 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 HCL® OneDB® 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
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 *HCL OneDB™ Guide to SQL: Syntax*.

The interaction between HCL® OneDB® 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 *HCL OneDB™ 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 *HCL OneDB™ DataBlade® API Function Reference* and the *HCL OneDB™ 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 HCL® OneDB® MQ extension provides external data access mechanisms for XA data sources.

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 553).

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 HCL OneDB™ XA implementation:

- All XA interface requests are available on updatable secondary servers (see Database updates on secondary servers on page 424).
- 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 Connection management through the Connection Manager on page 437).
- Redirection of XA requests from secondary servers to the primary server is supported.
- Transaction survival is supported for XA transactions, with the exception of transaction completion after failover (see Transaction completion during cluster failover on page 436).
- 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 424.
- 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()

Important: If you are using the .NET Framework with the Microsoft™ Transaction Server to manage XA transactions on a high availability cluster, you must use the TransactionScope class instead of the ServiceConfig class. The TransactionScope class is available in .NET Framework 3.5.

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.

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.

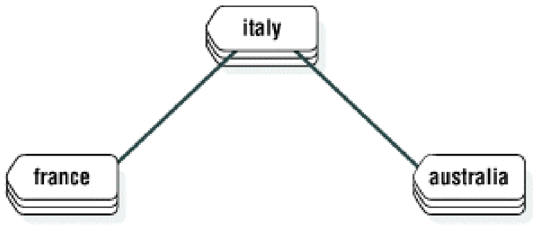
If any database server is unable to commit its portion of the transaction, all database servers participating in the transaction must be prevented from committing their work.

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 81. 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 554, 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 564.

- 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 554, 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 566.

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 557.)

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 Manually recovering from failed two-phase commit on page 571.)

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 on page 560)
- Heuristic end transaction (described in The heuristic end-transaction scenario on page 562)

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 *HCL OneDB™ 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 571.)

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 *HCL OneDB™ Administrator's Reference*.

The *address* parameter is obtained from onstat -x output. For more information about the onstat -x option, see the *HCL OneDB™ 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 *HCL OneDB™ 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 *HCL OneDB™ Performance Guide*. For a description of the fields that onstat -x displays, see the *HCL OneDB™ 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.

Example

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 561. 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 561. 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 559.

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 Manually recovering from failed two-phase commit on page 571.)

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

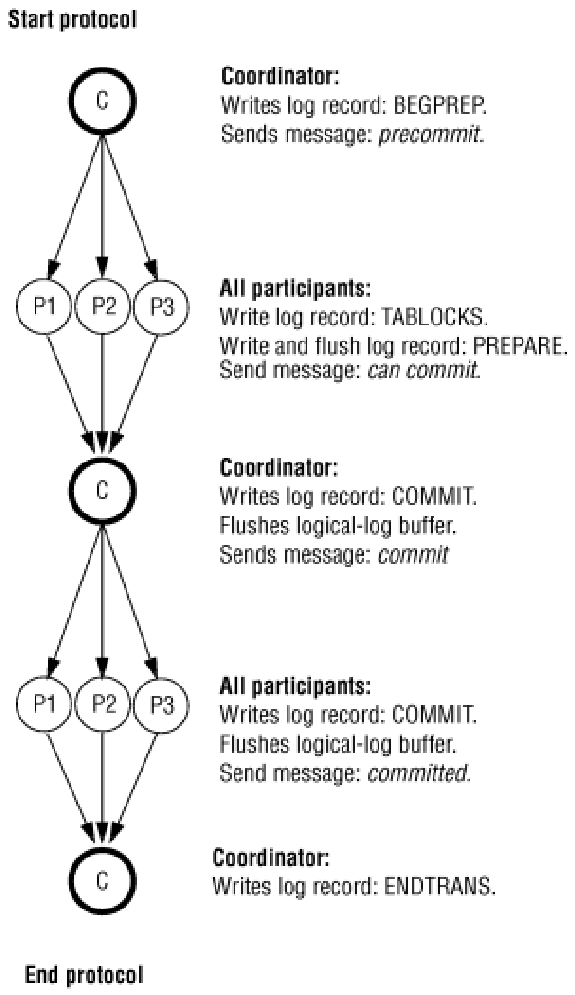
For information about these logical-log records, see the chapter on interpreting the logical log in the *HCL OneDB™ Administrator's Reference*.

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.

Figure 82. Logical-log records written during a committed transaction


Start protocol
C
Coordinator:
Writes log record: BEGPREP.
Sends message: *precommit*.
P1
P2
P3
All participants:
Write log record: TABLOCKS.
Write and flush log record: PREPARE.
Send message: *can commit*.
C
Coordinator:
Writes log record: COMMIT.
Flushes logical-log buffer.
Sends message: *commit*
P1
P2
P3
All participants:
Writes log record: COMMIT.
Flushes logical-log buffer.
Send message: *committed*.
C
Coordinator:
Writes log record: ENDTRANS.
End protocol

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.

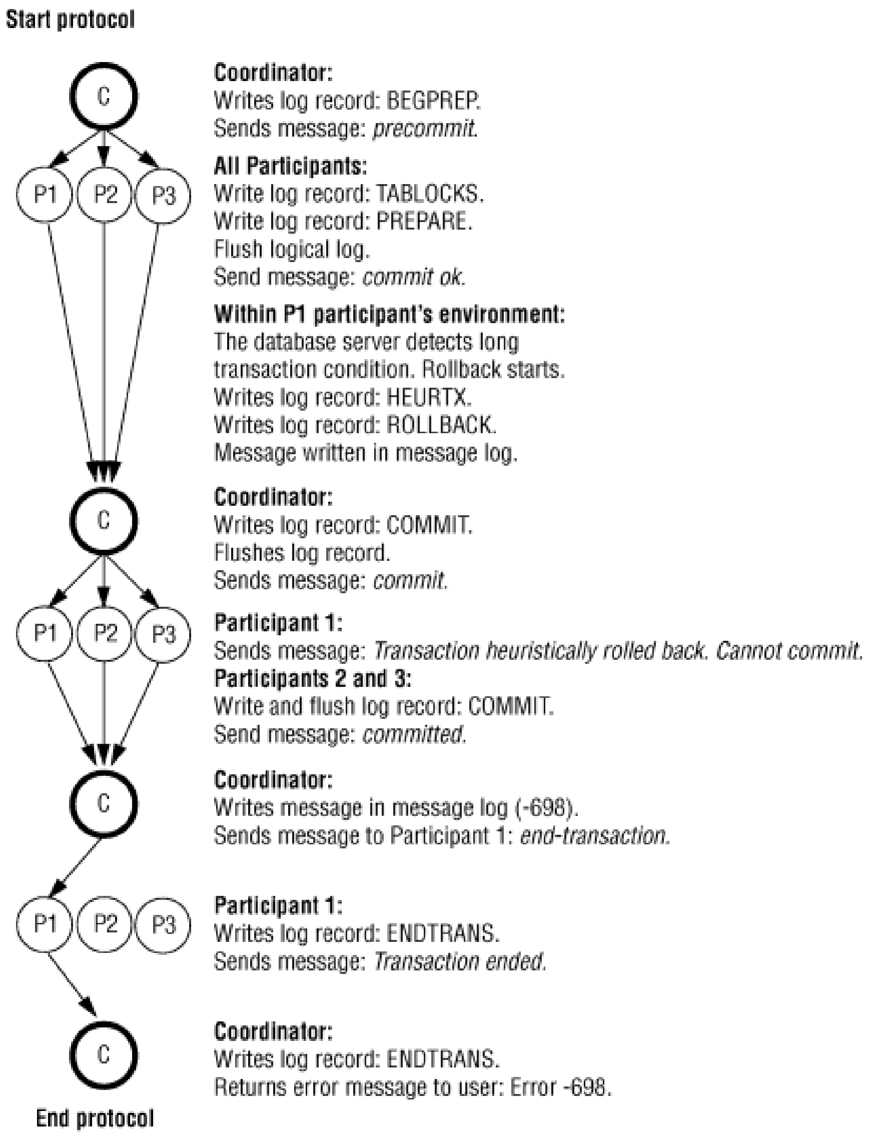
The participants in the preceding figure must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. As part of 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 82: Logical-log records written during a committed transaction on page 567. 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 560.

Figure 83. Logical-log records written during a heuristic rollback



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.

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.*

Coordinator:
Writes message in message log (-698).
Sends message to Participant 1: *end-transaction.*

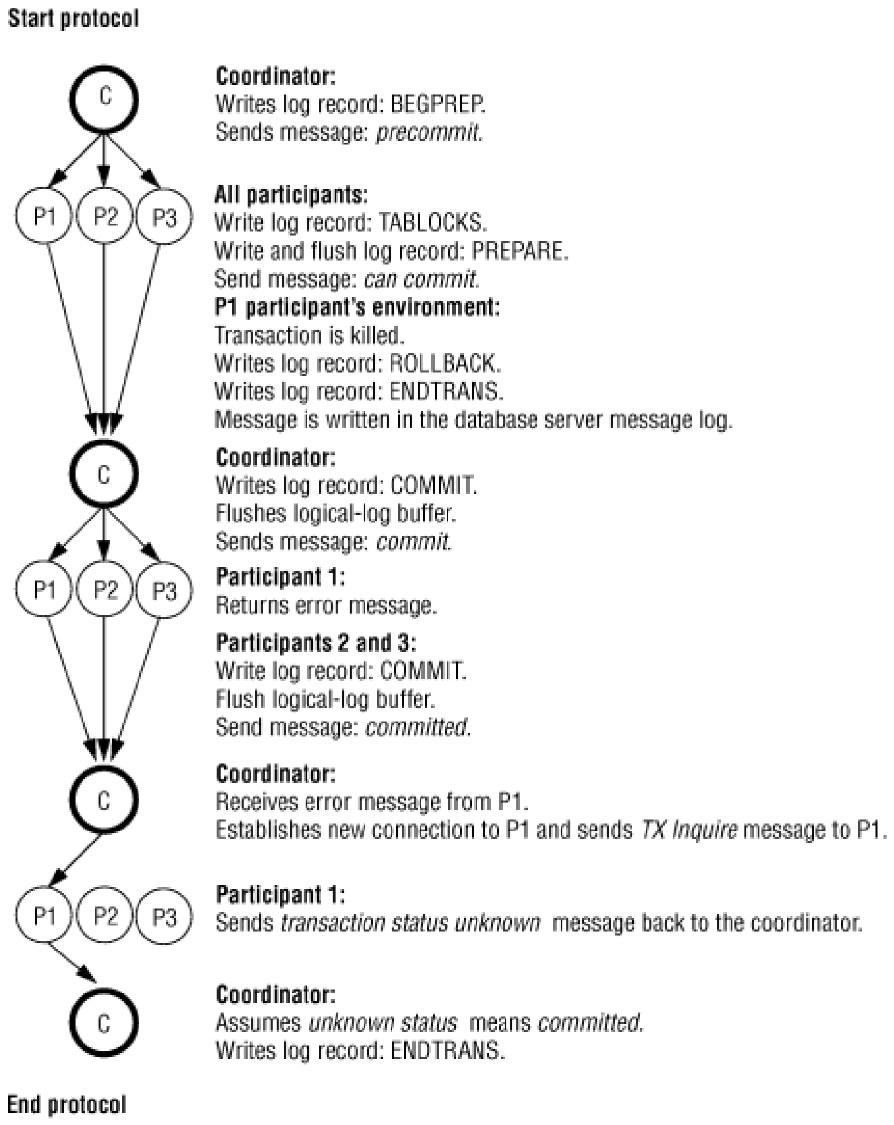
Participant 1:
Writes log record: ENDTRANS.
Sends message: *Transaction ended.*

Coordinator:
Writes log record: ENDTRANS.
Returns error message to user: Error -698.

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 *HCL OneDB™ 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 562.

Figure 84. Logical-log records written during a heuristic end transaction



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.

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 *HCL OneDB™ 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 556.

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 558.) 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 559.) 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 562.) 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

About this task

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 on page 574.
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).

Results

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

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    --------

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
```

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 HCL OneDB™ 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.

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 *HCL OneDB™ 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 72. 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 *HCL OneDB™ 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 73. Built-in tasks and sensors

Task or sensor	Description	Results table	Frequency	Retention
add_storage	This task add more storage space automatically when		As needed	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	automatic space management is configured.			
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_compress	This task compresses tables that are configured for automatic compression.			
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 Scheduling data optimization on page 263.		Once a week	
autoreg exe	This task registers database extensions when they are first used.		As necessary	
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		At server startup	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	database server. This task creates sub-tasks for individual databases, as necessary.			
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 virtual processors on page .		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.		Once a day	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	• AUS_SMALL_TABLES: tables containing fewer than 100 rows always have their statistics updated. For more information, see Automatic statistics updating on page .			
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 on page 356.		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		3 A.M. every 30 days (with a maximum of 12 log files)	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	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.			
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		Once a day	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	If a backup has not occurred, a warning alert is added to the **ph_alert** table.			
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 57.		Every 2 hours	
ifx_ha_monitor_log_replay_task	This task monitors the high-availability cluster replay position.		Not set	
mon_checkpoint	This sensor saves information about checkpoints.	**mon_checkpoint**	Every hour	7 days
mon_chunk	This sensor saves general information about chunk usage and I/O chunk performance.	**mon_chunk**	Every hour	30 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_compression_estimates	This sensor saves information about how much space	**mon_compression_ estimates**	Once a week	30 days

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
	might be saved if the data is compressed.			
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_iohistory	This sensor saves performance information about chunk I/O. You can change the IO_SAMPLES_PER_HOUR parameter in the **ph_threshold** table to collect information more frequently.		Every hour	30 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 222.	**mon_low_storage**	Every hour	7 days
mon_memory_system	This sensor collects information about the amount of memory the server uses.	mon_memory_system	Every hour	7 days
mon_page_usage	This sensor saves information about the pages that are used and free in storage spaces.	**mon_page_usage**	Once a day	7 days

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
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	

Table 73. Built-in tasks and sensors (continued)

Task or sensor	Description	Results table	Frequency	Retention
purge_tables	This task identifies rolling window tables whose purge policies have been exceeded. It discards or detaches qualifying fragments, according to each purge policy, until that policy is satisfied, or until no more fragments can be removed.		Daily at 00:45	
SET tk_enable	This task enables the tasks that rotate message log files.		3 A.M. every 30 days	
sync_registry	This task automatically converts the connection information between the `sqlhosts` file format and the Windows™ registry format.		Every 15 minutes	

Creating a task

You can create a Scheduler task to perform a specific action at specific times.

Before you begin

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

About this task

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:
    Choose from:

- **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.

Results

The task runs at the specified start time and subsequently at the time calculated from the frequency.

Example

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"
);
```

Creating a sensor

You can create a Scheduler sensor to collect and store data about the database server.

Before you begin

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

About this task

To create a sensor, use an INSERT statement to add a row into the **ph_task** table:

1. Include values for the following columns:
   Choose from:
   - **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:
   Choose from:
   - **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.
-

Results
The sensor runs at the specified start time and subsequently at the time calculated from the frequency.

Example

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
);
```

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.

Example

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;
```

Creating a group

You can create a group to organize Scheduler tasks and sensors.

Before you begin
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.

About this task
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

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.

Before you begin

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.

About this task

To create a threshold:

1. Use an INSERT statement to add values for the following columns in the **ph_threshold** table:

   Choose from:

   - **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

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.

Creating an alert

You can create an alert as part of the action of a Scheduler task or sensor.

Before you begin

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

About this task

To create an alert:

Use an INSERT statement to add a row to the **ph_alert** table. Include values for the following columns:

Choose from:

- **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

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
);
```

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.

Example

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.

Before you begin

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

About this task

To modify Scheduler properties:

Use an UPDATE statement for the appropriate Scheduler table in the **sysadmin** database.

Example

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";
```

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 *HCL OneDB™ 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.

Run the task() or admin() function in a transaction that does not include any other statements.

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:

```
BEGIN WORK;
EXECUTE FUNCTION admin("check extents");
END WORK;
```

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");
```

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.

Before you begin

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.

About this task

To display command history:

Use a SELECT statement to return the data from the **command_history** table.

Example

The following example displays all command history for the past 30 days:

```
SELECT * FROM command_history;
```

What to do next

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 *HCL OneDB™ Administrator's Reference*.

Table 74. 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.`

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.

Before you begin

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.

About this task

To reduce the retention period:

Use an UPDATE statement to modify the value of the COMMAND HISTORY RETENTION row in the **ph_threshold** table.

Example

The following example sets the retention period to 25 days:

```
UPDATE ph_threshold
SET value = "25"
WHERE name = "COMMAND HISTORY RETENTION";
```

What to do next

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);
```

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 not 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 *HCL OneDB™ Administrator's Reference*.

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.

Before you begin

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.

About this task

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

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 605.

For more information about using task() and admin() functions, see the *HCL OneDB™ Administrator's Reference*.

For more information about the SQLTRACE configuration parameter, including minimum and maximum values for some fields, see the *HCL OneDB™ 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.

Example

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 *HCL OneDB™ 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

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", sid)
  FROM sysmaster:syssessions
  WHERE username not in ("root","informix");
END
```

For more information about the task() and admin() functions, see the *HCL OneDB™ 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

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 605.

For more information about the task() and admin() functions, see the *HCL OneDB™ Guide to SQL: Syntax*.

HCL OneDB server licensing

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

The HCL OneDB™ server product incorporates a resource based license model, in contrast to a system based on hardware processor value units (PVUs). This allows you to run a small OneDB instance on a large system without incurring the penalty of paying for licensing all the hardware. Once you have chosen the CPU resources required for a particular operating environment they can be shared dynamically across multiple instances. This is controlled by utilizing the FlexNet licensing system from Flexera® software.

FlexNet Licensing Overview

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

The FlexNet licensing system is used to control the total number of CPU Virtual Processors (CPU VPs) that are available to an operating environment.

OneDB instances are required to obtain a OneDB license for each CPU VP that they wish to run.

A FlexNet License Server needs to be configured to hold a fixed number of OneDB licenses. It is then a matter of configuring the onconfig parameter **LICENSE_SERVER** to point to the FlexNet server device. All instances configured to use this FlexNet server will be able to share CPU VP resources. The Server Device will have a fixed number of licenses associated with it, each license equates to a CPU VP that can be claimed by an OneDB instance. This allows sharing of licensed resources across all systems in a production environment. OneDB licenses are also required for tenant virtual processors.

Note: When OneDB server is configured for multitenancy, it is possible to have the session threads run on tenant VPs instead of CPU VPs. The tenant VPs are treated like CPU VPs and licensing is applicable.

Licensing configuration

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

The value of LICENSE_SERVER parameter will be determined and controlled through the customers interaction with the FlexNet Operations (FNO) portal. It is referred to as a Cloud License Server (CLS) and its format is a simple URL: *https://hclsoftware.compliance.flexnetoperations.com/instances/<ServerDevice>/request*. All instances specifying this URL as its LICENSE_SERVER will be able to claim CPU resources from the pool available. Once the pool limit has been reached, then no further instances can be started. At this point it is possible to dynamically reduce the number of CPU VPs in one running instance and those licenses will then be made available to another instance.

ONCONFIG parameters

When sharing a FlexNet license server device among multiple instances, care needs to be taken that resources are available for all instances as required by their configuration. It is recommended that the AUTO_TUNE parameter is set to 0 to prevent the possibly damaging effect of having CPU VPs automatically added by HCL OneDB server itself. The VPCLASS parameters should also be reviewed to ensure that the starting number (num=N) and the maximum number (max=N) of CPU VPs is configured appropriately for all HCL OneDB server instances using the same FlexNet license Server Device.

For more information on LICENSE_SERVER configuration parameter, refer LICENSE_SERVER configuration parameter on page .

Related reference

LICENSE_TIMING configuration parameter

System startup

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

When an instance starts, the online log file will contain some messages indicating the current state of the licensing system. Consider an instance configured with a VPCLASS parameter set for num=2 CPU VPs. When the instance starts, assuming that there are 2 licenses available, then the following messages will be seen in the message log.

```
08:24:56  License Server configuration succeeded.
18:25:00  Server is holding 2 license(s).
```

If the license server cannot be contacted, or there are insufficient licenses available, you will see the following messages in the log.

```
11:46:12 Cannot acquire or validate licenses.

WARNING: Server might be running without enough licenses.

HCL OneDB Server requires licenses for the CPU/TENANT VP
resources to operate and it should be able to acquire or validate
them by connecting to the License Server.
You are receiving this warning because either the License Server
cannot be connected or there are not enough licenses available to
acquire for the VP resources.

Please verify the License Server connectivity and/or make sure there
are licenses available to prevent a server shutdown.
```

Dynamic CPU allocation

Every 10 seconds the server instance will check the number of CPU VPs and adjust licenses up or down as required. If you start the instance with 3 CPU VPs and dynamically add another, then you will see the licenses adjusted.

```
09:50:06  License Server configuration succeeded.
09:50:08  Server is holding 3 license(s).
09:50:08  Defragmenter cleaner thread now running.
09:50:08  Defragmenter cleaner thread cleaned:0 partitions
09:50:30  Dynamically added 1 cpu VP.
09:50:38  Server is holding 4 license(s).
```

If there are not enough licenses to fulfill the request for additional CPU VP resources then the VPs will be adjusted back down. If it is not possible to drop the VPs then the server willprint the following message.

```
10:00:52  Not enough licenses. VP resources were adjusted.
         HCL OneDB Server requires licenses for the CPU/TENANT
         VP resources to operate and it should be able to acquire them
         from the License Server.
         You are receiving this warning because there were not enough
         licenses available for the number of CPU/TENANT VPs. OneDB
         server tried to adjust the VP resources for the available number
         of licenses.
         Please make sure there are licenses available.
10:00:54 Warning: Only 1 out of 3 requested cpu VPs can be dropped
10:00:54 Dynamically dropped 1 cpu VP
10:00:54 Server is holding 1 license(s).
```

This instance is now in the state of having more CPU VPs allocated than licenses available and will be shutdown if the licenses are not made available.

Returning licenses

When CPU VPs are dropped or the instance is shutdown, the licenses will be returned to the FlexNet server. These licenses can then be claimed by another instance or by the same instance when it restarts.

In case of an abnormal shutdown, it may be impossible for the server instance to return the licenses and they will remain allocated to the instance even though it is not running. When the server instance is restarted, a verification check is performed that allows the previously allocated licenses to be reclaimed.

Note: When OneDB server is configured for multitenancy, it is possible to have the session threads run on tenant VPs instead of CPU VPs. The tenant VPs are treated like CPU VPs and licensing is applicable.

If it is not possible to restart the instance in a timely manner then the onclean utility can be used to return the licenses. Running onclean must be done from the context of the machine that hosted the original instance. You will see the licenses being returned:

```
$ onclean -yk
onclean: Cleaning up processes and resources for 'ol_onedb1210_2'...
- Looking for the master daemon process: 13904
Connected to License Server:
https://hclsoftware.compliance.flexnetoperations.com/instances/5BYD9U9NF7P2/request
2 license(s) returned successfully.
```

If the problem causing the abnormal termination has also made the physical server installation inaccessible, then recovery of the licenses will require the assistance of HCL Technical Support.

System alerts when licenses are not available

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

It is possible for the instance to get into a state where it is still running but there are insufficient licenses for the current CPU configuration. This may happen if there is a network issue that prevents contacting the license server. To understand how this situation can occur, and the remedial action required, the sequence of events for license checks is detailed below:

Every 10 seconds, the instance compares the acquired licenses with the number of CPU VPs. If the number of CPU VPs has not changed, then this license check does not require contacting the license server and a Trusted Storage area on disk, caches the licenses locally. If the number of CPU VPs has changed, or after 23 hours, when the license is within 1 hour of the expiration period, the instance will contact the license server to renew the license. If it cannot renew an expiring license, or acquire more licenses because of a change in the number of CPU VPs, then it will start issuing warnings. While being unable to reconcile the license count with the licence server, the number of held licenses will be set to zero.

The first warning will appear as a message in the **online.log** and will also trigger an alarm event, severity 3 (ATTENTION), class 29 (LICENSING), through the ALARMPROGRAM script. The instance will continue to try to obtain the necessary licenses every 10 seconds but it will not constantly emit warnings. The next warning will be 5 minutes after the license expired or became unavailable, then 10 minutes, then 20, and then 40 minutes. After these initial warnings, the alerts will continue to be produced on an hourly basis until 24 hours has passed from the time the licensing became non-compliant. If no licenses have been obtained during that time, then the server will write a message to the log to indicate it is shutting down, issue a severity 4 alarm (EMERGENCY), and then shutdown the instance.

In order to bring the server back online, it will be necessary to make licenses available again. If this requires changing the LICENSE_SERVER onconfig parameter, then the Trusted Storage cache may not be able to sync up with the FNO system and the instance will be unable to start. To recover from this state, it is necessary to remove the Trusted Storage area. This is a directory in *$ONEDB_HOME/etc* called *.lic.<SNUM>* - replacing <SNUM> with the SERVERNUM of the of the instance.

To aid troubleshooting of licensing issues, the environment variable IFX_FLEXLOG can be set (to any value between 1 and 4) before starting the instance. This will cause extra information regarding the licensing sub-system to be printed to the file pointed to by the instance's CONSOLE parameter. For example, if there is a network issue preventing access to the license server then you would see the following messages:

```
2021-05-27 09:21:12 LICENSING: <ERROR> Send Binary Message
2021-05-27 09:21:12 LICENSING: <ERROR> Unable to get license information from server
2021-05-27 09:22:12 LICENSING: <ERROR> server
 http://hclsoftware.compliance.flexnetoperations.com/instances/5BYD9U9NF7P2/request: curl error - 28:
Connection timed out after 60001 milliseconds
```

High Availability server instances

Important: Flexnet is deprecated in 2.0.1.3 and will not be supported in future versions.

The licensing of CPU VPs is also applicable to secondary servers in an HA configuration. The number of HA licenses available from the license server will be the same as the regular CPU entitlement. If the secondary instance is read only and has only 2 concurrent sessions connected, then HA specific, HID_CPU_HA licenses will be requested from the license server. If the secondary server is configured as updatable, or has more than 2 concurrent sessions connected, then regular HID_CPU licenses will be required to run that instance.

If a read only Secondary has to switch to primary mode, then it will return the HA licenses and acquire regular CPU licenses. This should be available as the primary is no longer online and should return its entitlement.

Monitoring the currently held licenses

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

onstat -g lic option is used to print out the current status of the licensing system

```
HCL OneDB Server Version 1.0.1.0 -- On-Line -- Up 00:13:21 -- 322188 Kbytes
2021-05-27 11:54:17

FlexNet DeviceID: ifxprod:005056850DFE:224
Licence count: 6
HID_CPU         : holding     6 : available    19
HID_CPU_HA      : holding     0 : available    19

Available license counts are as of 2021-05-27 11:27:19
Expected license expire time: 2021-06-24 11:27:19
```

The FlexNet DeviceID can be mapped to the device allocations in the FNO portal.

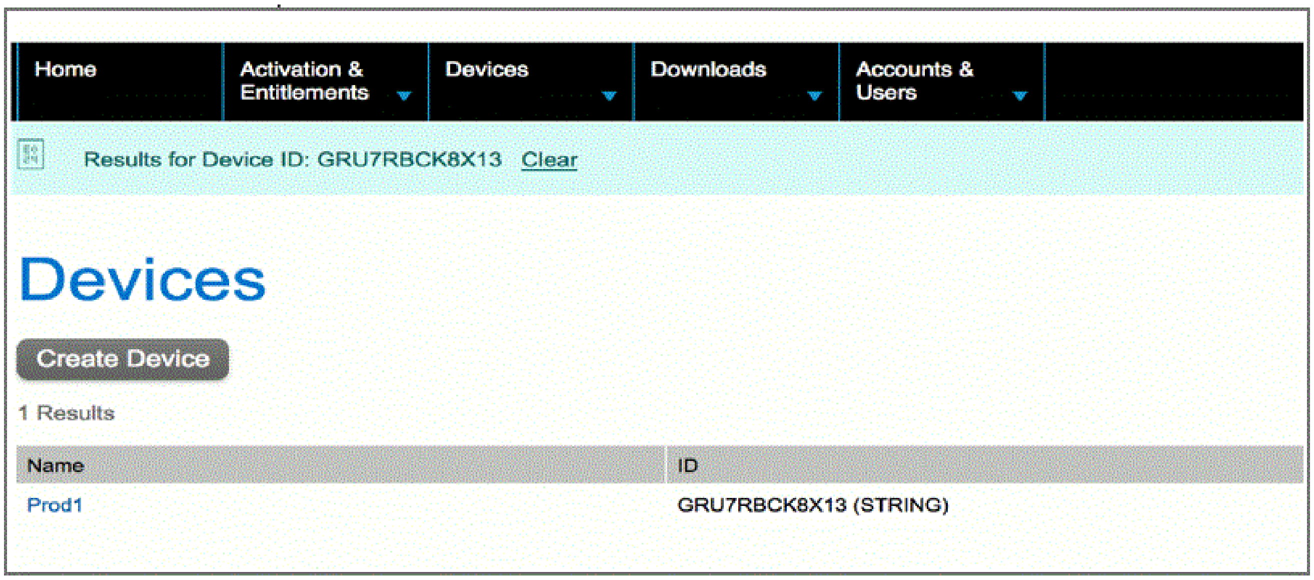
Note: There can be some delay in the update of the portal when licenses are acquired or released.

If the instance is in the warning state, unable to acquire the necessary licenses, then the projected shutdown time will be indicated. The **onstat -g** lic output is also included in the alert message from the ALARMPROGRAM script

FlexNet Operations portal

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

When the OneDB server instances are online it is possible to see their license holdings in the FlexNet Operations customer portal. You can drill down to get the details, starting with the Server Device. This is the device name that is in the LICENSE_SERVER URL. So for a LICENSE_SERVER value of https://hclsoftware.compliance.flexnetoperations.com/instances/GRU7RBCK8X13/ request, you would search in the portal for the Server Device ID **GRU7RBCK8X13**.



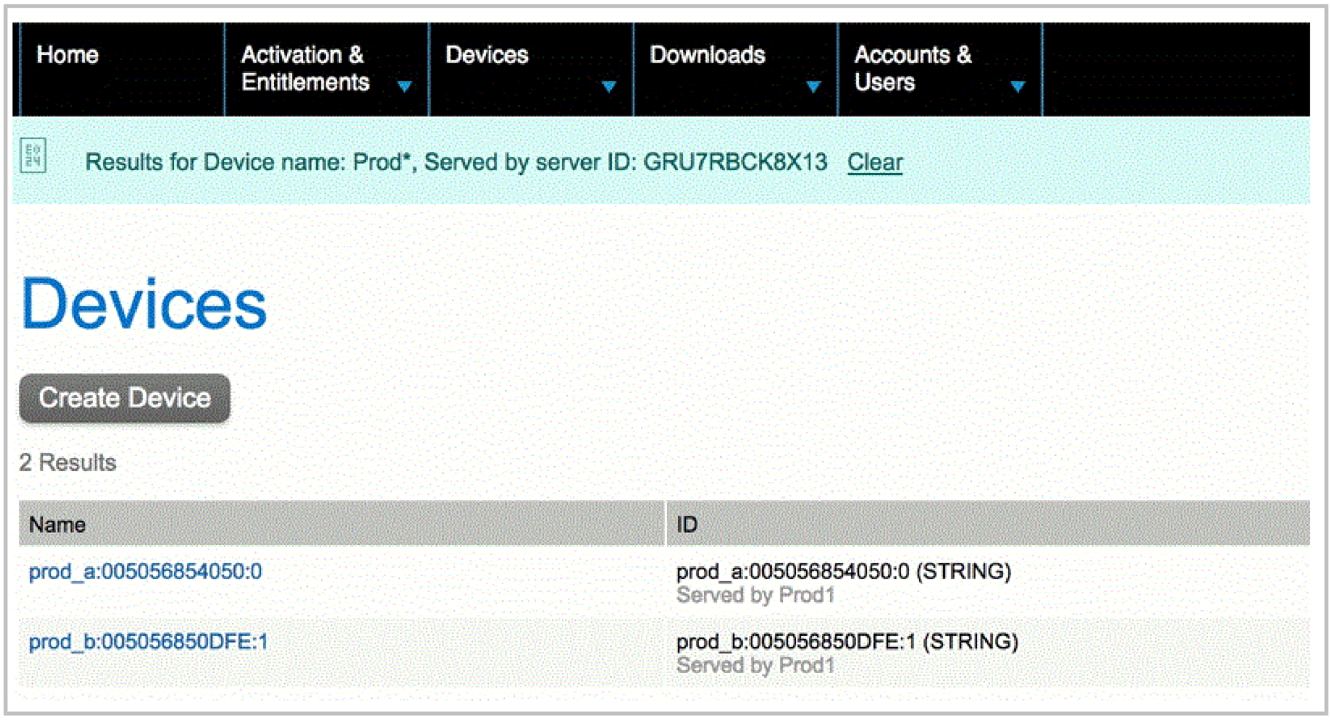
Results for Device ID: GRU7RBCK8X13 Clear

Devices

1 Results

Name	ID
Prod1	GRU7RBCK8X13 (STRING)

Click on the returned Device Name, Prod1, and you will see the details of the server device.



Results for Device name: Prod*, Served by server ID: GRU7RBCK8X13 Clear

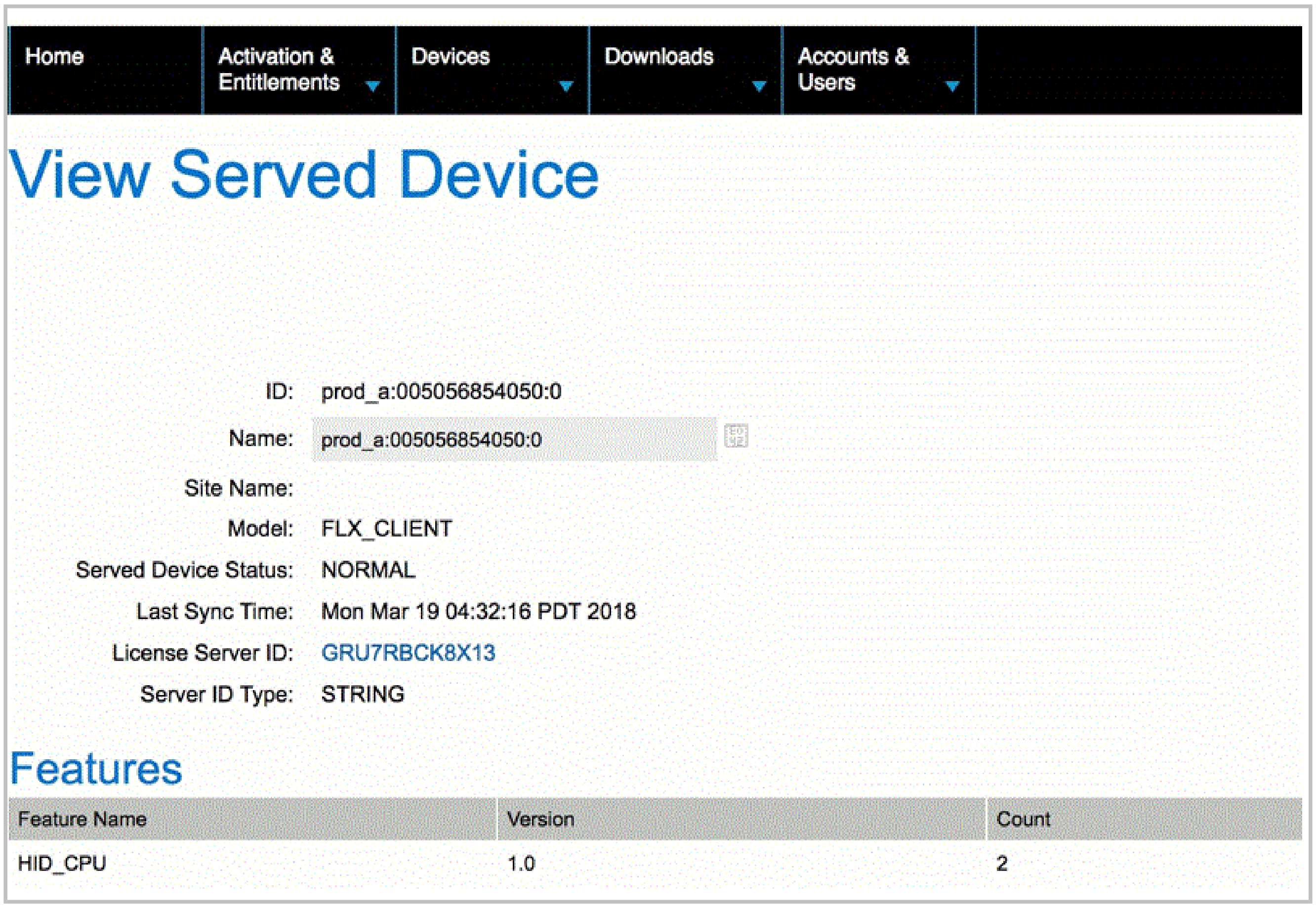
Devices

2 Results

Name	ID
prod_a:005056854050:0	prod_a:005056854050:0 (STRING) Served by Prod1
prod_b:005056850DFE:1	prod_b:005056850DFE:1 (STRING) Served by Prod1

From the View menu, select **View Served Devices** and you will see all the OneDB instances that have licenses assigned to them.

The device name that identifies the OneDB instance is made up of the host name, a unique identifier string and the SERVERNUM of the instance. Selecting the Device shows the individual license allocation.



View Served Device

ID:	prod_a:005056854050:0
Name:	prod_a:005056854050:0
Site Name:	
Model:	FLX_CLIENT
Served Device Status:	NORMAL
Last Sync Time:	Mon Mar 19 04:32:16 PDT 2018
License Server ID:	GRU7RBCK8X13
Server ID Type:	STRING

Features

Feature Name	Version	Count
HID_CPU	1.0	2

This count should tally with the number of licenses held that is reported in the server instances online.log. It should also be the same as the number of running CPU VPs.

Local License Server

The Local License Server (LLS) runs on a machine within the local network and serves out licenses just like the CLS that is visible on the internet. It is linked to the FNO back office through the use of a DeviceID that is specified using the LLS hostIdent as an ETHERNET device.

There are two possible configurations:

1. An online server that has a connection to the internet and will periodically sync up with the FNO back office server to keep track of the currently held licenses. This allows for the LLS to run on dedicated hardware with secure firewalls to isolate production instances of OneDB from the internet. The default sync time is 24 hourswhich can be changed through the REST interface.
2. An offline server that requires a manual data transfer operation to notify the FNO back office server of the current license allocations. The notification mechanism involves the downloading and uploading of files between the LLS and the backoffice server. Using this method allows for a complete air-gap to be maintained between the LLS and the internet.

Once the LLS is configured, it is accessed by the client device using the same LICENSE_SERVER configuration parameter. But instead of specifying the cloud based server, hclsoftware.compliance.flexnetoperations.com, the local network hostname for the machine running the LLS is used, along with the port number allocated to the server. It is possible to run the LLS on the same server as the OneDB instance and the LICENSE_SERVER URL, using the default port number, would then typically be *http://example.com:7070/instances/<DeviceID>/request*

Note: LLS requests for licenses are served over the HTTP protocol, unlike a CLS which uses the HTTPS.

The LLS will accept REST requests in the same way that the CLS does. The default REST configuration requires the set up of a secure HTTPS listener within the LLS to protect the exchange of authentication tokens. The Java utility keytool can be used to generate a self signed certificate for the server to use, although it is recommended that a valid corporate certificate is used. The admin user password needed to authenticate REST requests for the LLS is set to an initial value of "**HCLDefault1***".

For more information on LICENSE_SERVER configuration parameter, refer LICENSE_SERVER configuration parameter.

The REST interface

Important: FlexNet licensing system is deprecated in 2.0.1.3 and will not be supported in future versions.

Although changes made by server to acquire and release licenses happens almost instantaneously, the FlexNet Operations portal is not updated immediately and could take some time to sync up and provide the correct view of the system. For the most up to date view, it is possible to use the REST interface. You will need the appropriate Admin User privileges on the FlexNet Operation portal to be able to set a password for the server before the REST interface can be used.

Index